working on building a project in a role of product manager - detail out step by step breakdown of activities for day 5 in attached project plan

Day 5 is about delivering a working de-identification pipeline for clinical text with basic tests and producing the required artifacts: deid_pipeline.py, sample before/after outputs, and passing unit tests for names, phone, and dates. The detailed schedule below expands the Day 5 plan into a practical, time-boxed checklist aligned to the exact tools and outcomes specified in the project plan. [1]

Objectives

- Build a Python de-identification pipeline that detects PHI with Presidio Analyzer, preserves clinical sections with a sectionizer, replaces PHI with tags, and emits an encrypted PHI span map using Fernet. [2] [3] [4] [1]
- Produce artifacts: app/deid_pipeline.py, /secure_store/ encrypted span maps, before/after samples, and unit tests covering names, phone numbers, and dates. [1]

Prereqs check (10 min)

- Confirm Day 4 environment: Python 3.10+, dependencies installed (presidio-analyzer, presidio-anonymizer, spacy, medspacy, cryptography), and synthetic notes under /data/synthetic_notes/ as the input corpus. [1]
- Verify a Git repo with /app and /docs exists so Day 5 artifacts can be committed with traceable diffs and screenshots for the portfolio. [1]

Folder scaffolding (5 min)

- Create folders: /app for code, /secure_store for encrypted PHI span maps (write-only for the app), and /tests for unit tests to keep PHI maps physically segregated and auditable. [1]

Step 1: Pipeline interface (15 min)

- Define a minimal DeidPipeline class interface to load a note, analyze for PHI entities, apply replacements, and return both de-identified text and an encrypted JSON mapping for spans per the Day 5 objective. [1]

Step 2: PHI detection with Presidio (25 min)

- Initialize Presidio AnalyzerEngine and select common entities (e.g., PERSON/NAME, PHONE_NUMBER, DATE, SSN, EMAIL) to match Day 5 coverage, using Presidio's built-in recognizers.[2] [1]
- Validate one synthetic note end-to-end by printing recognizer results and confirming start/end character spans align with expected PHI regions before moving on.[2] [1]

Step 3: Preserve sections with a sectionizer (25 min)

- Add a sectionizer that tags clinical sections so replacements do not destroy section structure; use a medspaCy/clinical-sectionizer component or equivalent patterns for typical EHR headers (e.g., HPI, PMH, Assessment & Plan).[3] [1]
- Ensure the pipeline returns section-aware output, retaining headers and content boundaries to aid later RAG chunking and evaluation.[3] [1]

Step 4: Apply anonymization operators (20 min)

- Replace detected spans with neutral tags like [NAME], [PHONE], and [DATE], aligning with the plan's instruction to use anonymizer operators for consistent masking semantics.[1]
- Confirm that replacements do not overlap or break section headers, and that the de-identified text is safe to index in Week 2.[1]

Step 5: Encrypt PHI span map with Fernet (25 min)

- Build a PHI span map JSON: for each entity, record tag type, original text, start/end indices, and section context; then encrypt this JSON using Fernet and write to /secure_store/ as the authorized re-link payload.[4] [1]
- Store and load the Fernet key securely (env variable or local key file kept out of version control) and document rotation readiness via MultiFernet to align with future Day 10 key-rotation notes.[4] [1]

Step 6: CLI harness for quick runs (10 min)

- Add a small CLI entry point that takes an input file path and emits two outputs: de-identified .txt and an encrypted .bin (or .enc) for the span map under /secure_store/, matching the artifact expectations.[1]

Step 7: Unit tests (30 min)

- Write tests for names, phone, and dates to assert that tags appear in the de-identified output and that the encrypted span map decrypts to the correct original spans when a clinician role is authorized in later steps.[4] [2] [1]
- Include a section header preservation test to ensure the sectionizer maintains boundaries and titles during PHI replacement for clinical fidelity.[3] [1]

Evidence capture and artifacts (10 min)

- Save three representative before/after pairs and the corresponding encrypted span maps for the portfolio and for later RAG evaluation examples. [1]
- Commit: app/deid_pipeline.py, tests/test_deid_pipeline.py, sample inputs/outputs, and a README snippet explaining how the pipeline aligns with HIPAA de-identification and at-rest encryption. [1]

Acceptance criteria

- Running the pipeline on sample notes produces de-identified text suitable for indexing and an encrypted span map under /secure_store/ with Fernet. [4] [1]
- Tests pass for names, phone numbers, and dates, and section headers are preserved; artifacts exist and are committed per Day 5's deliverable list. [2] [3] [1]

Risks and mitigations

- False negatives in PHI detection: bias for higher recall entities initially and include manual spot checks; expand recognizers later if gaps appear. [2] [1]
- Structural drift: ensure replacements do not alter section headers or token boundaries critical to downstream chunking and retrieval. [3] [1]

Handoff to Day 6

- Record any custom entity patterns to carry into embedding metadata, and confirm de-identified text is organized for FAISS/Chroma indexing next day. [1]
- Note encryption key handling so re-linking workflows and audit logging can be validated during Week 2 and Day 10 hardening. [4] [1]

⁂

i am not technical, hence you have provide in easy language detailed steps as if I am shadowing you

Here is a simple, shadow-style Day 5 walkthrough that explains what happens, why it matters, and what gets captured, without requiring hands-on coding or prior technical knowledge. The goal is to get a first working de-identification flow for clinical notes: detect personal details, mask them with safe tags, keep clinical sections intact, and save an encrypted "map" for approved re-link later. [22] [23] [24] [25]

What Day 5 delivers

Day 5 ends with a basic de-identification pipeline, sample before/after outputs, and quick tests confirming names, phone numbers, and dates are masked while clinical section headers are preserved. The pipeline also saves an encrypted map of what was masked so authorized clinicians could re-link later if policy allows, using Fernet encryption for at-rest protection. [25] [22]

Simple explanation

Think of it like a smart highlighter and mask: a detector finds PHI like names, phone numbers, and dates; a masker swaps those with safe tags like [NAME] and [PHONE]; a "section keeper" makes sure clinical headers like HPI and Assessment remain as-is; and an encrypted map safely records what changed. Presidio does the detecting and masking, medspaCy's sectionizer keeps clinical sections intact, and Fernet locks the PHI map so only the right key can unlock it.[23] [24] [22] [25]

Morning checks

First, confirm the project folder exists with subfolders /app and /docs, and that a /data/synthetic_notes/ folder has sample notes created on Day 4 for testing. Also confirm the plan's core tools are installed (Presidio, medspaCy, and cryptography), which are used today for PHI detection, clinical section tagging, and encryption.[24] [22] [23] [25]

- Make sure /app, /docs, and /data/synthetic_notes/ are present so work can be saved in the right places for portfolio evidence and later steps.[22]
- Confirm Presidio works for detecting PII/PHI, medspaCy is available for section detection, and cryptography's Fernet is installed for encryption.[23] [24] [25] [22]

Build detector pass

The first pass is to prove detection works: run Presidio on one sample note to see it spotting items like person names, phone numbers, and dates with start–end positions in the text. Ask for a screenshot of these detections so it can be pasted into docs as proof the recognizers are active and returning spans.[22] [23]

- Request a single-note run and collect the Presidio results list showing each detected item's type and position, which demonstrates the detector is functioning.[23] [22]
- Verify at least names, phone numbers, and dates appear in detections, matching the Day 5 test focus.[22] [23]

Keep sections

Next, ensure clinical sections stay recognizable: enable the sectionizer so it tags headers like HPI, PMH, and Assessment & Plan, making it easier to keep clinical structure in later RAG steps. Confirm a quick dump of sections shows the titles and boundaries, then note that replacements should not remove or alter these headers.[26] [24] [22]

- Ask for a printout or log of detected sections from the same note, verifying headers like "History of Present Illness" or "Assessment and Plan" appear as separate sections.[27] [24] [22]
- Record that section titles must remain intact after masking, as downstream retrieval depends on them.[24] [22]

Mask with tags

Now convert detected PHI into safe tags: instruct masking so names become [NAME], phone numbers become [PHONE], and dates become [DATE], producing a clean text ready for indexing. Quickly inspect the masked output to ensure obvious PHI is gone and section headers still look the same.[23] [22]

- Review a side-by-side of original vs masked for one note, verifying PHI fields were replaced and sections were not altered.[22] [23]
- Confirm consistent tags are used so tests and documentation can reliably check outcomes. [23] [22]

Encrypt the map

Behind the scenes, also save a “map” that says what was replaced where, then encrypt that map with Fernet and store it in /secure_store so it cannot be read or changed without the key. Confirm the key is kept outside the repo and that an encrypted file was created for the test note.[25] [22]

- Ask for confirmation that an encrypted .bin (or .enc) file was written to /secure_store for the sample note.[25] [22]
- Note where the key is stored and that it is excluded from version control to prevent accidental exposure.[25] [22]

Run via a command

Have a simple command set up to point at a single input note and produce two outputs: the de-identified text and the encrypted map file under /secure_store. Use this once on three different sample notes to build the artifact set for the portfolio.[22]

- Request a demo run on three notes and collect de-identified outputs plus the corresponding encrypted maps as evidence.[22]
- Save the filenames and paths to paste into the README and to facilitate Day 6 indexing.[22]

Test quickly

Keep tests simple and readable: check that [NAME], [PHONE], and [DATE] appear in the masked output when expected; optionally decrypt one map file using the key to confirm the original values match the masked positions. Also include a check that section headers remain unchanged after masking.[25] [23] [22]

- Ask for a short test log showing these checks passed on at least one note and that decrypted map contents match what was masked.[25] [22]
- Capture a note that section titles before vs after are identical, proving structure preservation.[24] [22]

Capture evidence

Create a small evidence pack for Day 5: one screenshot of Presidio detections, one side-by-side before/after snippet, and a list of the encrypted files created. Commit the pipeline file, tests, and samples, and paste a short paragraph into the README explaining HIPAA-aligned choices (de-identified indexing; encrypted PHI map).[22]

- Save “before/after” pairs and the encrypted filenames to /docs or a screenshots folder for later case-study use.[22]
- Commit app/deid_pipeline.py, tests, and samples with a concise message mapping to the Day 5 artifact list.[22]

Acceptance criteria

A quick run on sample notes should produce de-identified text suitable for indexing plus an encrypted span map written to /secure_store using Fernet. Tests for names, phone, and dates pass, section headers remain intact, and artifacts are committed.[24] [23] [25] [22]

- Confirm masked outputs do not contain raw PHI and maps are unreadable without the key. [25] [22]
- Confirm unit tests pass and that preserved sections are visible in the final text.[24] [22]

Risks to watch

False negatives can occur in early detection; bias toward higher recall, perform spot checks, and expand recognizers later if gaps are observed. Structure drift is a risk if masking breaks headers, so always verify headers appear the same before and after.[23] [24] [22]

- Note any missed PHI types for future backlog items and custom recognizers as needed.[23] [22]
- Flag any section header anomalies immediately since they affect downstream RAG quality. [24] [22]

Handoff to Day 6

Document where de-identified outputs live so indexing can start, and note where the encryption key is kept for future authorized re-link tests. Include a one-liner in the README that indexing must only touch de-identified text, never the encrypted PHI map.[22]

- Provide the folder path for de-identified texts as Day 6 input along with the note IDs and sections.[22]
- Record key-handling notes so Day 10’s encryption hygiene and rotation checks are straightforward.[25] [22]

⁂

reshare above steps with detailed breakdown of steps with codes how to do it and every possible step

Here is a complete, copy-ready Day 5 playbook with step-by-step actions, exact commands, and code blocks, written so a non-technical product manager can execute by following along line-by-line. The objective is to build a basic de-identification pipeline that detects PHI in clinical text, masks it with tags, preserves section headers, and writes an encrypted “span map” for approved re-link later using Fernet encryption.[43] [44] [45] [46]

What will be created

By end of Day 5, there will be a working app/deid_pipeline.py, encrypted span maps in /secure_store, three before/after examples in /data/outputs, and unit tests proving names, phone numbers, and dates are masked while section headers remain unchanged. All artifacts will be committed to the repo with a README snippet explaining HIPAA-aligned masking and at-rest encryption using Fernet.[44] [45] [46] [43]

0) Prerequisites checklist

- Confirm folders exist: project root with /app, /docs, and /data/synthetic_notes containing sample notes from Day 4; if not, create them now.[43]
- Confirm Python 3.10+ is installed and the following packages are available: presidio-analyzer, presidio-anonymizer, spacy, medspacy (or clinical-sectionizer), and cryptography; if missing, install in a virtual environment.[45] [46] [44] [43]

Commands to run in terminal:

- On Windows PowerShell, Mac Terminal, or Linux shell, from the project root:[43]

```
python -m venv .venv
# Activate: Windows
. .venv/Scripts/Activate.ps1
# Activate: Mac/Linux
source .venv/bin/activate

pip install --upgrade pip
pip install presidio-analyzer presidio-anonymizer spacy cryptography
# For section detection: choose one
pip install medspacy
# OR the lighter pattern-only option
pip install clinical-sectionizer==1.0.0.0
```

Download a small English model for spaCy if prompted:

```
python -m spacy download en_core_web_sm
```

1) Create folders for Day 5

- Create structured locations: /app for code, /data/synthetic_notes for inputs, /data/outputs for de-identified text, /secure_store for encrypted maps, and /tests for unit tests.[43]

Commands:

```
mkdir -p app data/synthetic_notes data/outputs secure_store tests
```

If on Windows without -p support, run:

```
mkdir app
mkdir data
mkdir data\synthetic_notes
mkdir data\outputs
mkdir secure_store
mkdir tests
```

2) Put 2–3 sample notes into data/synthetic_notes

- If not already present, create simple synthetic sample files with obvious PHI to check masking; do not use real PHI.[43]

Example files:

- data/synthetic_notes/note1.txt[43]

```
HPI: John Smith, a 54-year-old male, presented on 03/15/2024 with chest pain.
PMH: Hypertension.
Contact: 555-123-4567.
Assessment and Plan: Rule out ACS.
```

- data/synthetic_notes/note2.txt[43]

```
History of Present Illness: Jane Doe reports migraine since 02/01/2023.
Medications: Ibuprofen.
Phone: (212) 555-7890. Email: jane.doe@example.com.
Assessment & Plan: Start prophylaxis.
```

3) Configure encryption key for Fernet

- Fernet is a symmetric encryption method; one secret key encrypts and decrypts the PHI span map.[46]
- Generate a key file one time and keep it out of Git (gitignore) for safety.[46] [43]

Create app/gen_key.py:

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()
with open("secure_store/fernet.key", "wb") as f:
    f.write(key)
print("Key written to secure_store/fernet.key")
```

Run it:

```
python app/gen_key.py
```

Add secure_store/fernet.key to .gitignore:

```
echo "secure_store/fernet.key" >> .gitignore
```

4) Build the de-identification pipeline code

- This script will detect PHI with Presidio, preserve sections, mask PHI with tags, and write an encrypted JSON span map to /secure_store.[45] [44] [46] [43]
- It also exposes a command-line interface for easy runs per file.[43]

Create app/deid_pipeline.py:

```
import json
import os
from dataclasses import dataclass
from typing import List, Dict, Any, Tuple

# Presidio
from presidio_analyzer import AnalyzerEngine
from presidio_analyzer.recognizer_registry import RecognizerRegistry
from presidio_anonymizer import AnonymizerEngine

# NLP for optional section detection
import spacy

# If using medspacy, uncomment (preferred for clinical):
# import medspacy
# from medspacy.sectionizer import Sectionizer

# If not using medspacy, optional lightweight section tagging:
# We'll use regex on common headers as a fallback
import re

# Encryption
from cryptography.fernet import Fernet

@dataclass
class PHISpan:
    entity_type: str
    start: int
    end: int
    text: str
    section: str
```

```
SECTION_HEADERS = [
    # Common clinical sections; customize as needed
    "HPI", "History of Present Illness",
    "PMH", "Past Medical History",
    "Medications", "Allergies",
    "Assessment and Plan", "Assessment & Plan", "Assessment",
    "Plan", "ROS", "Review of Systems",
    "Physical Exam"
]

SECTION_PATTERN = re.compile(
    r"^(?P<header>(" + "|".join([re.escape(h) for h in SECTION_HEADERS]) + r"))\s*:\s*$",
    re.IGNORECASE | re.MULTILINE
)

TAG_MAP = {
    "PERSON": "[NAME]",
    "PHONE_NUMBER": "[PHONE]",
    "DATE_TIME": "[DATE]",
    "DATE": "[DATE]",
    "EMAIL_ADDRESS": "[EMAIL]",
    "US_SSN": "[SSN]"
}

class DeidPipeline:
    def __init__(self, fernet_key_path: str = "secure_store/fernet.key"):
        # Load encryption key
        with open(fernet_key_path, "rb") as f:
            self.fernet = Fernet(f.read())

        # Presidio setup
        self.registry = RecognizerRegistry()
        self.registry.load_predefined_recognizers()
        self.analyzer = AnalyzerEngine(registry=self.registry)
        self.anonymizer = AnonymizerEngine()

        # NLP pipeline for section detection (spaCy base)
        self.nlp = spacy.load("en_core_web_sm")

    def _detect_sections(self, text: str) -> List[Tuple[str, int, int]]:
        """
        Lightweight section finder:
        Return list of (section_title, start_idx, end_idx_of_section_block)
        """
        # Find headers by regex, map their start positions
        headers = []
        for m in SECTION_PATTERN.finditer(text):
            headers.append((m.group("header"), m.start()))
        # Add end sentinel
        headers.append(("[END]", len(text)))

        sections = []
        for i in range(len(headers) - 1):
            title, start_pos = headers[i]
            next_title, next_pos = headers[i+1]
            sections.append((title.strip(), start_pos, next_pos))
        if not sections:
            # Single default section if none found
            sections = [("DOCUMENT", 0, len(text))]
        return sections

    def _find_section_for_span(self, sections, start_idx) -> str:
        for title, s, e in sections:
            if s <= start_idx < e:
                return title
        return "DOCUMENT"

    def analyze(self, text: str) -> List[Dict[str, Any]]:
        # Detect entities
        results = self.analyzer.analyze(text=text, language="en")
        # Convert to dict for consistency
        detections = []
        for r in results:
            detections.append({
                "entity_type": r.entity_type,
                "start": r.start,
                "end": r.end,
                "score": r.score
            })
        return detections

    def mask(self, text: str, detections: List[Dict[str, Any]]) -> Tuple[str, List[PHISpa
        """
        Replace spans with tags safely (right-to-left to maintain indices).
        """
        # Determine sections for context
        sections = self._detect_sections(text)

        # Build PHI span records
        spans: List[PHISpan] = []
        for d in detections:
            entity = d["entity_type"]
            start = d["start"]
            end = d["end"]
            original = text[start:end]
            section = self._find_section_for_span(sections, start)
            spans.append(PHISpan(entity_type=entity, start=start, end=end, text=original,

        # Replace from the end to avoid index shifting
        masked = text
        for d in sorted(detections, key=lambda x: x["start"], reverse=True):
            entity = d["entity_type"]
            start = d["start"]
            end = d["end"]
            tag = TAG_MAP.get(entity, f"[{entity}]")
            masked = masked[:start] + tag + masked[end:]

        return masked, spans

    def encrypt_span_map(self, spans: List[PHISpan], meta: Dict[str, Any]) -> bytes:
        payload = {
            "meta": meta,
            "spans": [s.__dict__ for s in spans]
        }
        blob = json.dumps(payload).encode("utf-8")
        token = self.fernet.encrypt(blob)
        return token

    def run_on_text(self, text: str, note_id: str) -> Dict[str, Any]:
        detections = self.analyze(text)
        masked, spans = self.mask(text, detections)

        # Encrypt span map
        token = self.encrypt_span_map(
            spans=spans,
            meta={"note_id": note_id}
        )

        return {
            "masked_text": masked,
            "encrypted_span_map": token
        }

def run_file(input_path: str, outputs_dir: str = "data/outputs", secure_dir: str = "secu
    os.makedirs(outputs_dir, exist_ok=True)
    os.makedirs(secure_dir, exist_ok=True)

    note_id = os.path.splitext(os.path.basename(input_path))[^3_0]
    with open(input_path, "r", encoding="utf-8") as f:
        text = f.read()

    pipeline = DeidPipeline()
    result = pipeline.run_on_text(text, note_id=note_id)

    # Save masked text
    out_txt = os.path.join(outputs_dir, f"{note_id}.deid.txt")
    with open(out_txt, "w", encoding="utf-8") as f:
        f.write(result["masked_text"])

    # Save encrypted span map
    out_bin = os.path.join(secure_dir, f"{note_id}.spanmap.enc")
    with open(out_bin, "wb") as f:
        f.write(result["encrypted_span_map"])

    print(f"De-identified text -> {out_txt}")
    print(f"Encrypted span map -> {out_bin}")

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="De-identify a clinical note and save en
    parser.add_argument("--input", required=True, help="Path to input .txt note")
    parser.add_argument("--outputs_dir", default="data/outputs", help="Output folder for
    parser.add_argument("--secure_dir", default="secure_store", help="Folder for encrypte
    args = parser.parse_args()
    run_file(args.input, args.outputs_dir, args.secure_dir)
```

Why this works:

- Presidio Analyzer detects common PII/PHI like names, phones, dates, emails, and SSN using predefined recognizers; these give start–end spans in the text for masking.[44]
- Section detection preserves clinical structure by identifying headers; the script uses a simple regex approach, and can be upgraded to medspaCy's Sectionizer for richer section rules later.[45]
- Fernet provides authenticated encryption; the encrypted span map can only be decrypted with the same secret key, satisfying at-rest protection requirements.[46]

5) Run the pipeline on sample notes

- Execute the script for each sample note to generate de-identified outputs and encrypted span maps.[43]

Commands:

```
python app/deid_pipeline.py --input data/synthetic_notes/note1.txt
python app/deid_pipeline.py --input data/synthetic_notes/note2.txt
```

Expected files:

- data/outputs/note1.deid.txt and data/outputs/note2.deid.txt containing masked content like [NAME], [PHONE], and [DATE] while section headers remain intact.[45] [44] [43]
- secure_store/note1.spanmap.enc and secure_store/note2.spanmap.enc which are unreadable without the key, proving at-rest encryption.[46] [43]

6) Quick verification by reading outputs

- Open data/outputs/note1.deid.txt in a text editor and visually confirm that names, phone numbers, and dates are replaced with tags while headers like HPI or Assessment & Plan are unchanged.[44] [45] [43]
- Do the same check for note2.deid.txt and confirm consistent tagging.[44] [43]

7) Minimal unit tests for confidence

- Create tests that assert the presence of tags, preservation of section headers, and decrypt one encrypted span map to confirm the original values were captured correctly.[44] [46] [43]

Create tests/test_deid_pipeline.py:

```
import os
import json
from cryptography.fernet import Fernet
from app.deid_pipeline import DeidPipeline, run_file

def test_masking_basic(tmp_path):
    # Arrange: create a temp note with clear PHI and sections
    note = """HPI: John Smith presented on 03/15/2024.
Contact: (212) 555-7890.
Assessment and Plan: Start aspirin."""
```

```
    note_path = tmp_path / "temp_note.txt"
    note_path.write_text(note, encoding="utf-8")

    # Copy key into temp secure dir
    os.makedirs(tmp_path / "secure_store", exist_ok=True)
    with open("secure_store/fernet.key","rb") as f:
        (tmp_path / "secure_store" / "fernet.key").write_bytes(f.read())

    # Act
    os.makedirs(tmp_path / "data" / "outputs", exist_ok=True)
    os.makedirs(tmp_path / "secure_store", exist_ok=True)
    # Instantiate pipeline with temp key
    pipeline = DeidPipeline(fernet_key_path=str(tmp_path / "secure_store" / "fernet.key")
    result = pipeline.run_on_text(note, note_id="temp_note")

    # Assert masked tags present
    masked = result["masked_text"]
    assert "[NAME]" in masked or "[PERSON]" in masked
    assert "[PHONE]" in masked
    assert "[DATE]" in masked

    # Assert sections preserved (headers remain)
    assert "HPI:" in masked
    assert "Assessment and Plan:" in masked

    # Decrypt span map and sanity check fields
    token = result["encrypted_span_map"]
    key = (tmp_path / "secure_store" / "fernet.key").read_bytes()
    f = Fernet(key)
    payload = json.loads(f.decrypt(token).decode("utf-8"))
    assert payload["meta"]["note_id"] == "temp_note"
    assert len(payload["spans"]) > 0
```

Run tests (if pytest is installed):

```
pip install pytest
pytest -q
```

If pytest is not desired, a manual sanity check is sufficient:

- Confirm tags appear in masked output; confirm headers still present; and decrypt with f = Fernet(open("secure_store/fernet.key","rb").read()) then f.decrypt(open("secure_store/note1.spanmap.enc","rb").read()) to visually inspect JSON fields.[46] [43]

8) Capture evidence artifacts for portfolio

- Save three artifacts: a screenshot of the terminal showing detections running and output files written, a side-by-side before/after snippet from one note, and a list of encrypted files present in /secure_store.[43]
- Add a short README section explaining that only de-identified text is indexed in later steps and that the encrypted map is retained for authorized re-link under policy.[46] [43]

README snippet to paste into docs/README.md:

```
Day 5: De-identification pipeline
- PHI detection via Microsoft Presidio with predefined recognizers (names, phones, dates,
- Section preservation via regex headers (upgradeable to medspaCy Sectionizer).
- Masking with tags: [NAME], [PHONE], [DATE], [EMAIL], [SSN].
- Encrypted span map written to /secure_store using Fernet (key stored outside Git).
- Only de-identified text in /data/outputs is used for indexing (Weeks 2+).
```

9) Git commit

- Commit code, tests, outputs, and documentation so Day 6 can start with indexing; the key file remains excluded.[43]

Commands:

```
git add app/deid_pipeline.py tests/test_deid_pipeline.py data/outputs docs/README.md .git
git commit -m "Day 5: de-id pipeline with PHI masking, section preservation, encrypted sp
```

Troubleshooting tips

- If spaCy model error appears, run python -m spacy download en_core_web_sm again and retry.[44] [43]
- If no sections detected, it's fine; the fallback "DOCUMENT" label keeps processing safe; consider adding medspaCy Sectionizer later for richer sectioning.[45] [43]
- If Presidio misses an entity, proceed with Day 5 as-is and log missed types for adding custom recognizers in a later hardening day to improve recall.[44] [43]

Acceptance criteria (self-check)

- Running on sample notes produces masked outputs and encrypted span maps; masked text shows [NAME], [PHONE], and [DATE], and headers like HPI remain unchanged.[45] [44] [43]
- Encrypted files exist in /secure_store and cannot be read without the key; tests or manual decrypt confirm original spans and metadata were captured correctly.[46] [43]

References used for this step-by-step build:

- Presidio "Getting started" and basic usage to detect and anonymize PII/PHI in text, which provides recognizers and examples for PERSON, PHONE_NUMBER, DATE, EMAIL, and SSN entities.[44]
- medspaCy sectionizer repository and documentation providing a rule-based way to detect clinical section headers, used here as a future upgrade from simple regex headers.[45]
- Python cryptography's Fernet documentation, which defines authenticated symmetric encryption used to protect the PHI span map at rest and support safe re-link workflows.[46]

⁂

getting following error at 5 point:

(new_env_rag) C:\Users\hipaa-compliant-rag-clinical>python app/deid_pipeline.py --input data/synthetic_notes/note2.txt
Traceback (most recent call last):
File "C:\Users\hipaa-compliant-rag-clinical\app\deid_pipeline.py", line 7, in <module>
from presidio_analyzer import AnalyzerEngine
File "C:\Users\hipaa-compliant-rag-clinical\new_env_rag\Lib\site-packages\presidio_analyzer_*init*_.py", line 9, in <module>
from presidio_analyzer.entity_recognizer import EntityRecognizer
File "C:\Users\hipaa-compliant-rag-clinical\new_env_rag\Lib\site-packages\presidio_analyzer\entity_recognizer.py", line 6, in <module>
from presidio_analyzer.nlp_engine import NlpArtifacts
File "C:\Users\hipaa-compliant-rag-clinical\new_env_rag\Lib\site-packages\presidio_analyzer\nlp_engine_*init*_.py", line 7, in <module>
from .stanza_nlp_engine import StanzaNlpEngine
File "C:\Users\hipaa-compliant-rag-clinical\new_env_rag\Lib\site-packages\presidio_analyzer\nlp_engine\stanza_nlp_engine.py", line 16, in <module>
from spacy import Language, blank
ImportError: cannot import name 'Language' from 'spacy' (C:\Users\hipaa-compliant-rag-clinical\new_env_rag\Lib\site-packages\spacy_*init*_.py)

This error means Presidio is pulling a spaCy integration path that expects specific spaCy symbols, but the installed spaCy version in the virtual environment doesn't match what Presidio expects on Windows. The quickest fix is to align versions and explicitly tell Presidio to use its spaCy engine with a supported spaCy release, then reinstall cleanly in the active venv.[48] [49]

What's happening

Presidio imports its NLP engines at import time and detects spaCy/Stanza backends; on Windows, mismatched spaCy versions can trigger "cannot import name 'Language' from 'spacy'" because underlying API names moved across spaCy versions or incompatible plugin shims were present. The stack trace shows Presidio's stanza_nlp_engine touching spaCy symbols, indicating a cross-dependency mismatch even if Stanza isn't used directly.[50] [51] [52] [48]

Fast, safe resolution

Follow these steps inside the same "new_env_rag" environment:

- Ensure the venv is active, then uninstall and pin compatible packages :[49] [48]

```
pip uninstall -y spacy presidio-analyzer presidio-anonymizer spacy-stanza stanza
pip cache purge

pip install "spacy==3.7.4"
python -m spacy download en_core_web_sm
```

```
pip install "presidio-analyzer==2.2.352" "presidio-anonymizer==2.2.352"
```

This combo uses spaCy 3.7.x (stable on Windows) and Presidio 2.2.x which targets spaCy NLP by default and avoids older stanza bridges that caused import errors on spaCy 3+.[53] [48] [49]

- Avoid Stanza unless explicitly needed; if ever switching to Stanza, install spacy-stanza and stanza together and configure Presidio's NlpEngineProvider accordingly, otherwise Presidio tries to discover engines and may load a conflicting path on import in some setups.[51] [52] [53]

Minimal code tweak to force spaCy engine

Add a few lines at the top of app/deid_pipeline.py to force Presidio to initialize with spaCy, bypassing auto-detection paths that might probe Stanza codepaths on import :[51] [48]

```
from presidio_analyzer.nlp_engine import NlpEngineProvider

# Force spaCy engine explicitly to avoid stanza import paths
NLP_CONFIG = {
    "nlp_engine_name": "spacy",
    "models": [{"lang_code": "en", "model_name": "en_core_web_sm"}],
}

provider = NlpEngineProvider(nlp_configuration=NLP_CONFIG)
spacy_engine = provider.create_engine()

from presidio_analyzer import AnalyzerEngine, RecognizerRegistry
from presidio_anonymizer import AnonymizerEngine

registry = RecognizerRegistry()
registry.load_predefined_recognizers()
analyzer = AnalyzerEngine(nlp_engine=spacy_engine, registry=registry)
anonymizer = AnonymizerEngine()
```

Then update your pipeline class to accept an injected analyzer/anonymizer so these are used, or adapt the class init to build from this exact config, ensuring Presidio uses spaCy 3.7 with en_core_web_sm.[48] [51]

Clean run checklist

- Verify the environment is active and the versions are correct:

```
python -c "import spacy, presidio_analyzer; import pkgutil, sys; print('spaCy', spacy.__v
```

If that prints versions and "model ok," the spaCy install and model are compatible.[54] [50]

- Re-run Day 5 command:

```
python app/deid_pipeline.py --input data/synthetic_notes/note2.txt
```

This should write masked text to data/outputs and the encrypted span map to secure_store without import errors, as Presidio now binds to spaCy with the pinned versions and explicit engine configuration.[49] [48]

If the error persists

- Check for stray global spaCy installs shadowing the venv; run "where python" on Windows to confirm using the venv interpreter, and "pip show spacy" to confirm it matches the venv path.[50]
- Remove spacy-stanza and stanza unless truly required; those bridges can trigger Presidio to import stanza_nlp_engine even when unused if present in the environment on some versions.[52] [51]
- As a last resort in Windows, create a fresh venv and only install the three essentials in order: spaCy 3.7.4, en_core_web_sm, then Presidio analyzer/anonymizer pinned as above, to avoid leftover metadata from previous attempts.[50] [48] [49]

⁂

getting this error:

(new_env_rag) C:\Users\hipaa-compliant-rag-clinical>python -m spacy download en_core_web_sm
Collecting https://github.com/explosion/spacy-models/releases/download/-en_core_web_sm/-en_core_web_sm.tar.gz
ERROR: HTTP error 404 while getting https://github.com/explosion/spacy-models/releases/download/-en_core_web_sm/-en_core_web_sm.tar.gz
ERROR: Could not install requirement https://github.com/explosion/spacy-models/releases/download/-en_core_web_sm/-en_core_web_sm.tar.gz because of HTTP error 404 Client Error: Not Found for url: https://github.com/explosion/spacy-models/releases/download/-en_core_web_sm/-en_core_web_sm.tar.gz for URL https://github.com/explosion/spacy-models/releases/download/-en_core_web_sm/-en_core_web_sm.tar.gz

The 404 means the automatic model downloader is constructing a bad URL for the English model version, which commonly happens with certain spaCy versions on Windows or behind network restrictions; the fix is to install a known-good wheel URL for the model that matches the installed spaCy version, or to pin a compatible pair and install the model via a direct wheel link instead of the download helper.[69] [70] [71]

Quick fix: install a matching wheel directly

- Keep spaCy at a stable 3.7.x, then install the model wheel by URL, which avoids the failing "download" helper and resolves the 404 immediately on Windows. Use this pair that's known to work: spaCy 3.7.5 with model 3.7.1.[70] [72]
- In the activated venv, run:

```
pip install "spacy==3.7.5"
pip install "en_core_web_sm @ https://github.com/explosion/spacy-models/releases/downloac
```

```
python -c "import spacy, en_core_web_sm; nlp=en_core_web_sm.load(); print('OK', spacy.__v
```

This bypasses the downloader and pins a compatible pair per the community workaround for recent 404/compatibility table issues with spaCy 3.7.x, after which spaCy.load('en_core_web_sm') will work in the pipeline.[71] [69] [70]

If a corporate proxy/firewall blocks GitHub

- Download the wheel once outside the restricted network and install from a local file path with pip install path\to\en_core_web_sm-3.7.1-py3-none-any.whl, which spaCy supports for models, or mirror it to an internal artifact repo and reference that URL in pip install, both of which avoid the failing online fetch step.[73] [74]
- After installing from file or internal mirror, verify with python -c "import en_core_web_sm; en_core_web_sm.load(); print('model ok')", which confirms the model is importable without using the downloader API that returned 404 earlier.[74] [69]

Alternative: use spaCy's CLI inside Python

- If command-line download fails sporadically, calling the same function via spacy.cli.download can sometimes succeed within the active interpreter session; this is still subject to the same backend URLs, so prefer the wheel method above, but it can be tried quickly:

```
python -c "import spacy; from spacy.cli import download; download('en_core_web_sm')"
```

This mirrors python -m spacy download behavior and is documented in community answers, but again can hit the same 404 depending on the spaCy/model version table at the moment, so the direct wheel remains the most reliable path on Windows.[75] [71]

Why the error occurs

- The downloader first fetches a "compatibility table" and constructs a model URL for the current spaCy version; recent reports show certain 3.7.x combinations produce missing or moved model assets, so the resolved URL is 404 even though the model exists for a slightly different sub-version, which is why 3.7.5 + model 3.7.1 is the recommended pairing that installs cleanly via a direct wheel URL.[69] [70]
- The official docs confirm models are separate packages installed per version and can be installed directly by pip from the models repo or local files, which is the supported workaround when the helper fails with 404/403 or when networks block GitHub redirects. [76] [71]

After installation: re-run Day 5

- Once en_core_web_sm loads, re-run the earlier Presidio steps, ideally with Presidio explicitly bound to spaCy as noted previously, and the de-identification script should initialize successfully and process the sample notes without import or model errors.[77] [78]
- If any residual errors appear, print the versions in the venv to confirm the environment is consistent: python -c "import spacy; print(spacy.**version**); import en_core_web_sm; en_core_web_sm.load(); print('ok')" and then proceed to run the pipeline command for note2 again, which should now succeed.[79] [69]

⁂

getting error: (new_env_rag) C:\Users\hipaa-compliant-rag-clinical>python app/deid_pipeline.py --input data/synthetic_notes/note1.txt

Traceback (most recent call last):
File "C:\Users\hipaa-compliant-rag-clinical\app\deid_pipeline.py", line 197, in <module>
run_file(args.input, args.outputs_dir, args.secure_dir)
File "C:\Users\hipaa-compliant-rag-clinical\app\deid_pipeline.py", line 172, in run_file
text = f.read()
^^^^^^^^
File "<frozen codecs>", line 322, in decode
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb0 in position 535: invalid start byte

The file note1.txt isn’t actually UTF-8 text; Windows saved it with a different encoding, so Python fails when reading it as UTF-8 by default. The fix is to: 1) detect and read with a Windows-friendly encoding fallback, and 2) normalize output to UTF-8 for downstream steps so everything stays consistent.[88]

What to change

Update app/deid_pipeline.py so file reading is robust on Windows by trying UTF-8 first, then falling back to cp1252 (common for Notepad/.docx copy-pastes), and as a last resort using errors="replace" to avoid runtime crashes while still allowing masking to proceed.[88]

Replace your run_file function with this version:

```
def _read_text_with_fallback(path: str) -> str:
    # 1) Try UTF-8 (preferred for cross-platform)
    try:
        with open(path, "r", encoding="utf-8") as f:
            return f.read()
    except UnicodeDecodeError:
        pass
    # 2) Try Windows-1252 (common for Notepad/docx copy-paste on Windows)
    try:
        with open(path, "r", encoding="cp1252") as f:
            return f.read()
    except UnicodeDecodeError:
        pass
    # 3) Last resort: decode with replacement to avoid crashing; preserves structure
    with open(path, "r", encoding="utf-8", errors="replace") as f:
        return f.read()

def run_file(input_path: str, outputs_dir: str = "data/outputs", secure_dir: str = "secu
    os.makedirs(outputs_dir, exist_ok=True)
    os.makedirs(secure_dir, exist_ok=True)

    note_id = os.path.splitext(os.path.basename(input_path))[^6_0]
    text = _read_text_with_fallback(input_path)

    pipeline = DeidPipeline()
    result = pipeline.run_on_text(text, note_id=note_id)

    # Save masked text normalized to UTF-8
    out_txt = os.path.join(outputs_dir, f"{note_id}.deid.txt")
    with open(out_txt, "w", encoding="utf-8", newline="\n") as f:
        f.write(result["masked_text"])

    # Save encrypted span map (binary)
    out_bin = os.path.join(secure_dir, f"{note_id}.spanmap.enc")
    with open(out_bin, "wb") as f:
        f.write(result["encrypted_span_map"])

    print(f"De-identified text -> {out_txt}")
    print(f"Encrypted span map -> {out_bin}")
```

Why this works

- Some Windows editors save plain .txt files as cp1252 or ANSI, which produces bytes like 0xB0 that are invalid in strict UTF-8, causing UnicodeDecodeError when opened with encoding="utf-8".[88]
- The fallback sequence ensures reading succeeds while preserving clinical sections and most characters; output is then written in UTF-8, giving stable downstream behavior for indexing and tests.[88]

Optional: quickly fix the source file

If preferred, convert note1.txt to UTF-8 once so future runs are clean:

- Open in Notepad, choose “Save As…”, set Encoding to “UTF-8”, overwrite the file, then re-run the command; or use PowerShell:

```
Get-Content data\synthetic_notes\note1.txt | Set-Content -Encoding UTF8 data\synthetic_nc
```

This normalizes the file to UTF-8 and avoids needing the fallback for that file going forward.[88]

Re-run

Run the pipeline again after updating run_file or converting the file:

```
python app/deid_pipeline.py --input data/synthetic_notes/note1.txt
```

This should complete without the UnicodeDecodeError, write a UTF-8 de-identified output to data/outputs, and the encrypted span map to secure_store, keeping Day 5 on track.[88]

⁂

1. COMPLETE-WEEK-BY-WEEK-TASKS.docx
2. https://microsoft.github.io/presidio/analyzer/
3. https://pypi.org/project/clinical-sectionizer/1.0.0.0/
4. https://cryptography.io/en/latest/fernet/
5. https://pypi.org/project/presidio-structured/
6. https://github.com/medspacy/sectionizer
7. https://www.youtube.com/watch?v=jewJln5Ecfk
8. https://www.slideshare.net/slideshow/an-introduction-to-medspacy/240099104
9. https://www.geeksforgeeks.org/python/fernet-symmetric-encryption-using-cryptography-module-in-python/
10. https://www.kaggle.com/code/shweta0812/pii-data-detection-using-presidio-analyzer
11. https://pmc.ncbi.nlm.nih.gov/articles/PMC8861690/
12. https://gist.github.com/palevell/bb034b559bb9dde9911c97219b6889ad
13. https://gitlab-scm.partners.org/bw720/satspacy/-/blob/main/medspacy/section_detection/sectionizer.py
14. https://stackoverflow.com/questions/66667705/python-cryptography-fernet-reduce-length-of-encrypted-text-generated
15. https://www.kaggle.com/code/haithamaliryan/nbme-extract-features-using-medspacy-lib
16. https://www.comparitech.com/blog/information-security/what-is-fernet/
17. https://pypi.org/project/presidio-analyzer/
18. https://microsoft.github.io/presidio/samples/python/presidio_notebook/
19. https://www.marktechpost.com/2025/06/24/getting-started-with-microsofts-presidio-a-step-by-step-guide-to-detecting-and-anonymizing-personally-identifiable-information-pii-in-text/
20. https://blog.stackademic.com/presidio-in-action-detecting-and-securing-pii-in-text-451711e3c544
21. https://www.tutorialspoint.com/fernet-symmetric-encryption-using-a-cryptography-module-in-python
22. COMPLETE-WEEK-BY-WEEK-TASKS.docx
23. https://microsoft.github.io/presidio/getting_started/
24. https://github.com/medspacy/sectionizer
25. https://cryptography.io/en/latest/fernet/
26. https://pypi.org/project/clinical-sectionizer/1.0.0.0/
27. https://pmc.ncbi.nlm.nih.gov/articles/PMC8861690/

28. https://blog.stackademic.com/presidio-in-action-detecting-and-securing-pii-in-text-451711e3c544
29. https://www.geeksforgeeks.org/python/fernet-symmetric-encryption-using-cryptography-module-in-python/
30. https://www.statcan.gc.ca/en/data-science/network/identifying-personal-identifiable-information
31. https://www.tutorialspoint.com/fernet-symmetric-encryption-using-a-cryptography-module-in-python
32. https://developer.mamezou-tech.com/en/blogs/2025/01/04/presidio-intro/
33. https://www.slideshare.net/slideshow/an-introduction-to-medspacy/240099104
34. https://www.hideipvpn.com/privacy/what-is-python-fernet/
35. https://arize.com/blog/pii-removal-microsoft-presidio-chatbot/
36. https://dev.to/sreeni5018/microsoft-presidio-and-langgraph-enhancing-ai-agents-with-robust-pii-protection-and-data-14oo
37. https://spacy.io/universe/project/medspacy
38. https://www.comparitech.com/blog/information-security/what-is-fernet/
39. https://gitlab-scm.partners.org/bw720/satspacy/-/blob/main/medspacy/section_detection/sectionizer.py
40. https://microsoft.github.io/presidio/samples/python/presidio_notebook/
41. https://www.marktechpost.com/2025/06/24/getting-started-with-microsofts-presidio-a-step-by-step-guide-to-detecting-and-anonymizing-personally-identifiable-information-pii-in-text/
42. https://www.youtube.com/watch?v=4GuK9sPUvus
43. COMPLETE-WEEK-BY-WEEK-TASKS.docx
44. https://microsoft.github.io/presidio/getting_started/
45. https://github.com/medspacy/sectionizer
46. https://cryptography.io/en/latest/fernet/
47. https://pypi.org/project/clinical-sectionizer/1.0.0.0/
48. https://microsoft.github.io/presidio/api/analyzer_python/
49. https://microsoft.github.io/presidio/getting_started/
50. https://spacy.io/usage
51. https://microsoft.github.io/presidio/tutorial/05_languages/
52. https://github.com/microsoft/presidio/issues/813
53. https://stackoverflow.com/questions/68660137/why-does-presidio-stanzanlpengine-throw-nameerror-name-stanzalanguage-is-not
54. https://spacy.io/usage/models
55. https://github.com/explosion/spaCy/issues/3188
56. https://spacy.io/api/language
57. https://www.nature.com/articles/s41598-025-04971-9
58. https://github.com/microsoft/presidio/discussions/726
59. https://par.nsf.gov/servlets/purl/10504238
60. https://hackage.haskell.org/package/unscramble-1.0/src/lists/enable.txt
61. https://docs.llamaindex.ai/en/v0.10.23/CHANGELOG/
62. https://cs.slu.edu/~goldwasser/class/slu/csci150/2010_Spring/programs/prog10/boglong.txt

63. https://epub.jku.at/obvulihs/download/pdf/8620803
64. https://nlp.biu.ac.il/~ravfogs/resources/embeddings-alignment/glove_vocab.250k.txt
65. https://stackoverflow.com/questions/79549787/why-does-presidio-with-spacy-nlp-engine-not-recognize-organizations-and-pesel-wh
66. https://huggingface.co/datasets/maxbarshay/langchain-docs
67. https://stackoverflow.com/questions/53676542/importerror-e048-cant-import-language-en-from-spacy-lang
68. https://github.com/explosion/spaCy/discussions/9376
69. https://spacy.io/models/en
70. https://github.com/explosion/spaCy/issues/13690
71. https://spacy.io/usage/models
72. https://pypi.org/project/spacy/3.7.5/
73. https://stackoverflow.com/questions/77230706/using-only-pip-for-installing-spacy-model-en-core-web-sm
74. https://github.com/explosion/spaCy/discussions/13162
75. https://stackoverflow.com/questions/64715848/getting-error-while-loading-spacy-en-core-web-sm-library
76. https://github.com/explosion/spacy-models
77. https://microsoft.github.io/presidio/getting_started/
78. https://microsoft.github.io/presidio/api/analyzer_python/
79. https://spacy.io/usage
80. https://github.com/explosion/spaCy/issues/3000
81. https://deepnote.com/blog/ultimate-guide-to-the-spacy-library-in-python
82. https://pypi.org/project/spacy/
83. https://github.com/explosion/spaCy/issues/8574
84. https://discuss.streamlit.io/t/no-module-named-en-core-web-sm/7003
85. https://github.com/explosion/spaCy/discussions/10895
86. https://colab.research.google.com/drive/1Dk4LjDGih2RrkNi6jBC9d4FH4Z2OKP19?usp=sharing
87. https://pypi.org/project/spacy/2.2.4/
88. COMPLETE-WEEK-BY-WEEK-TASKS.docx