AI-Powered Sensitive Data Masking for Documents: Java implementation with Google Cloud DLP & Document AI) - ANIL LALAM

Abstract:

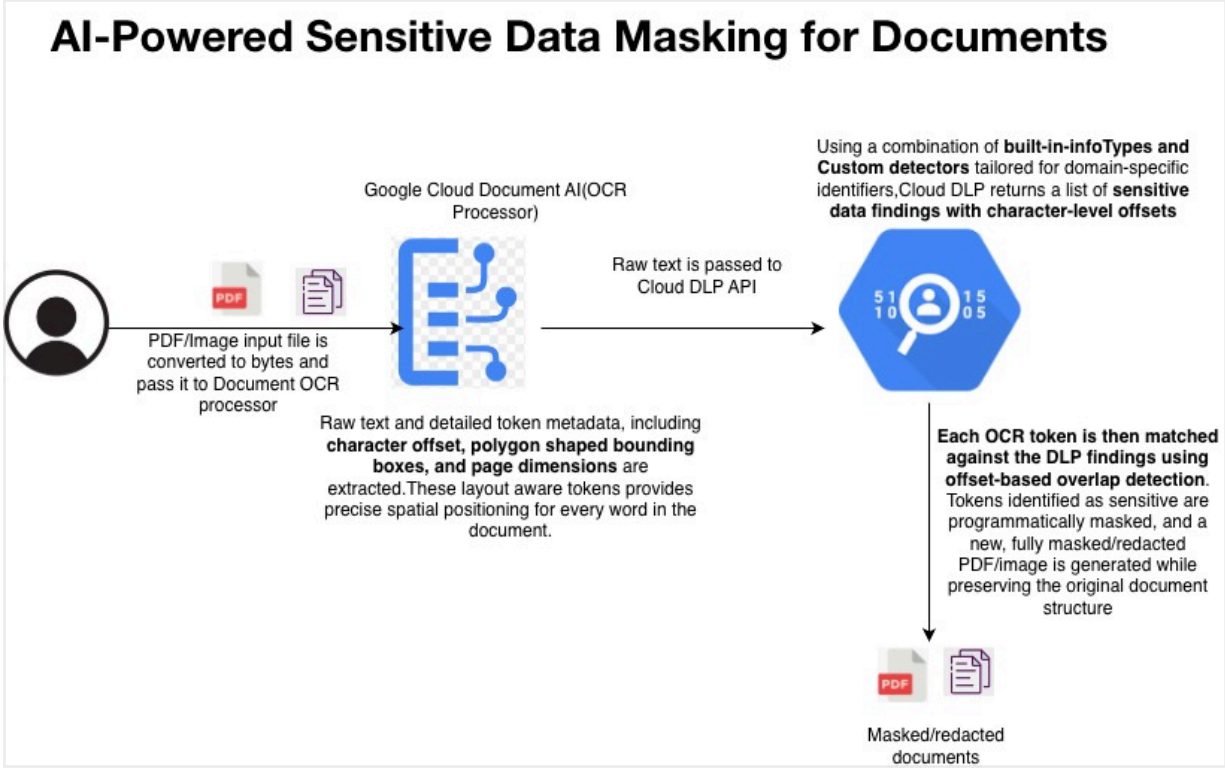
Sensitive data such as personal identifiers, financial numbers, and government-issued IDs are frequently embedded within documents, making automated detection and redaction an essential requirement for modern enterprise workflows. This article presents a complete, production-ready Java implementation for **AI-driven sensitive data masking**, leveraging **Google Cloud Document AI** for text extraction and layout understanding, and **Google Cloud DLP** for robust detection of PII and custom entity types.

The system begins by accepting user-provided PDF or image files, which are converted into byte arrays and processed using the Document AI OCR model. This step extracts both the **raw text** and detailed **token metadata**, including character offsets, polygon-shaped bounding boxes, and page dimensions. These layout-aware tokens provide precise spatial positioning for every word in the document.

In the second stage, the raw text from Document AI is submitted to the Cloud DLP API. Using a combination of built-in infoTypes and custom detectors tailored for domain-specific identifiers, Cloud DLP returns a list of sensitive-data findings with character-level offsets. Each OCR token is then matched against the DLP findings using offset-based overlap detection. Tokens identified as sensitive are programmatically masked, and a new, fully masked/redacted PDF/image is generated while preserving the original document structure.

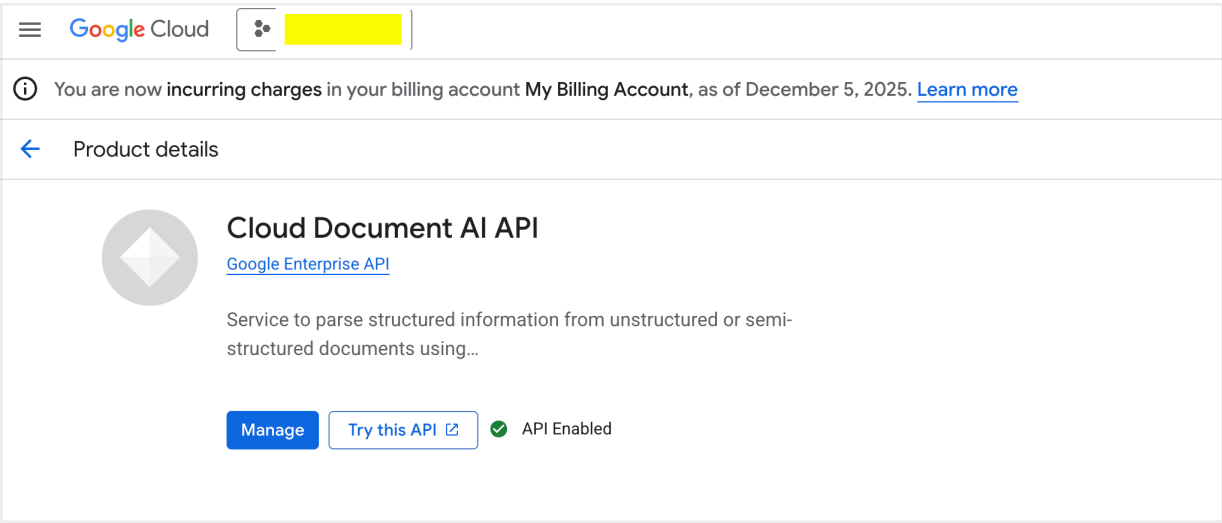
This end-to-end pipeline demonstrates how AI-powered text extraction, PII detection, and document spatial analysis can be combined to build a highly accurate, scalable, and automated solution for secure document handling. The implementation is designed for real-world use cases such as compliance workflows, data privacy automation, and enterprise document processing.

Architecture Diagram :

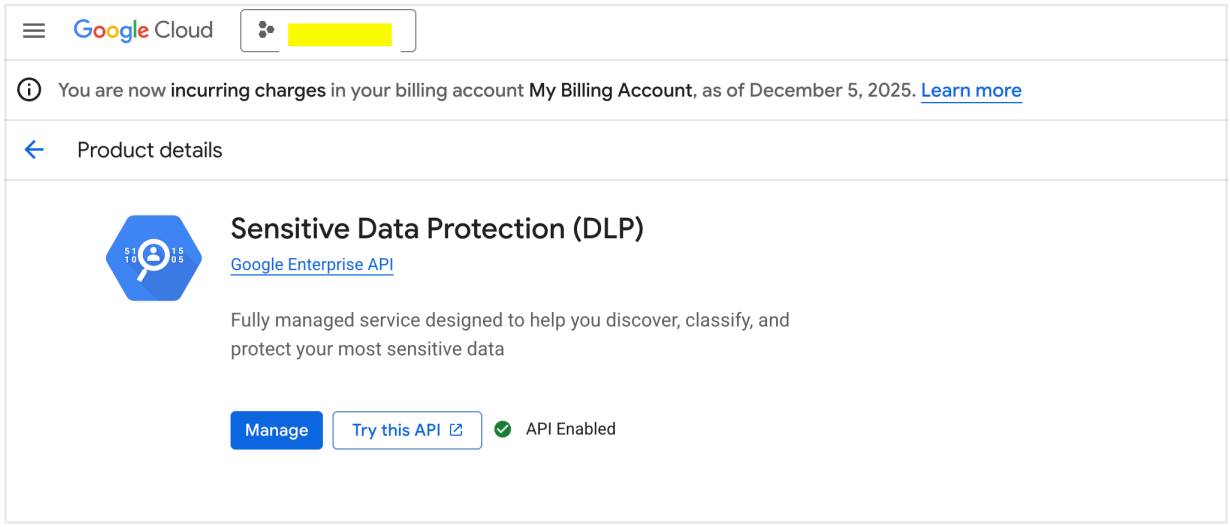


Pre-requisites :

1. Enable Cloud Document AI API



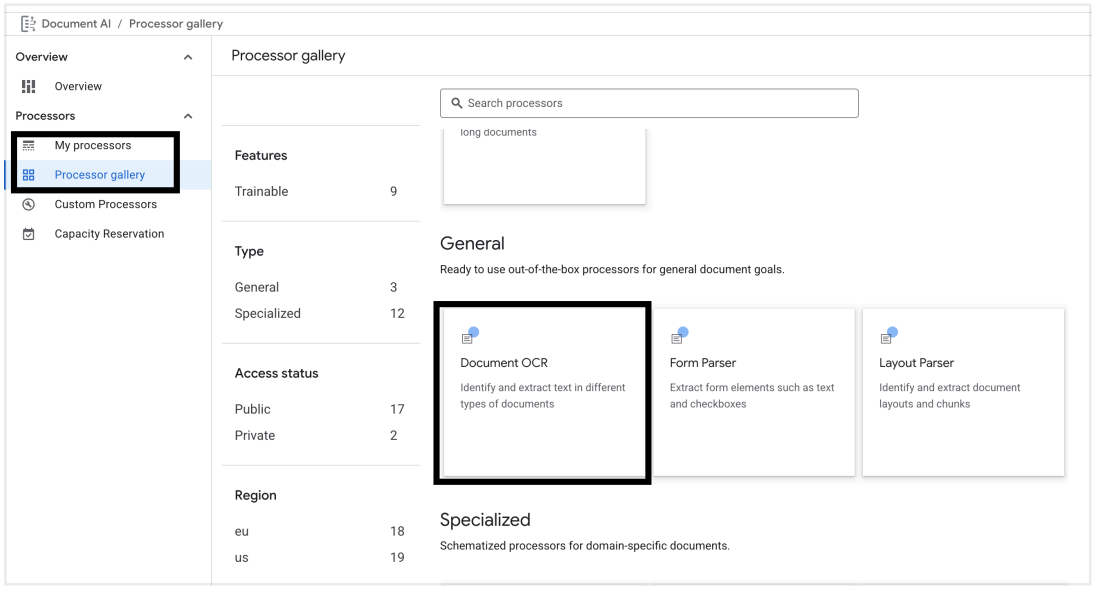
2. Enable Sensitive Protection (DLP) API

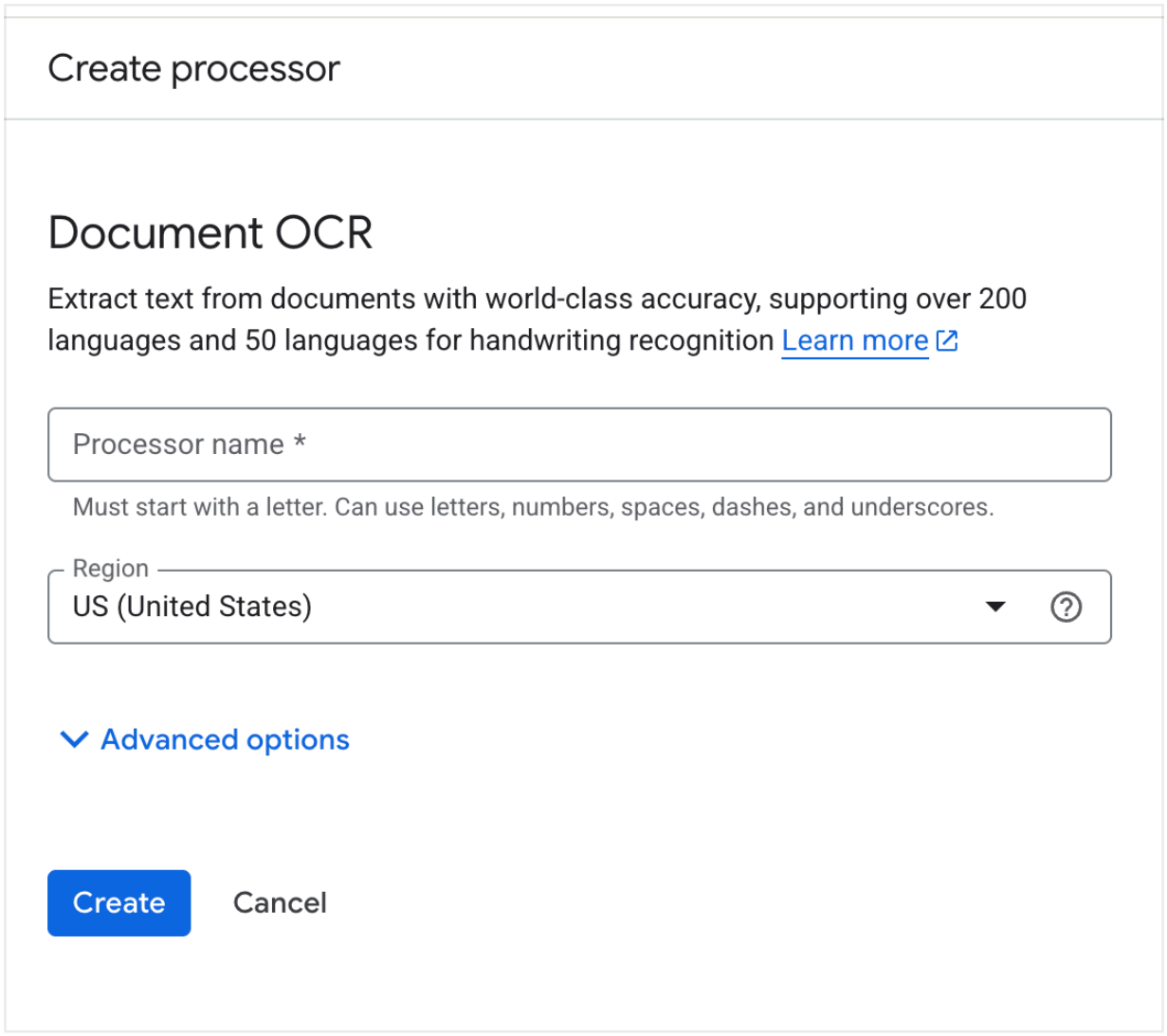


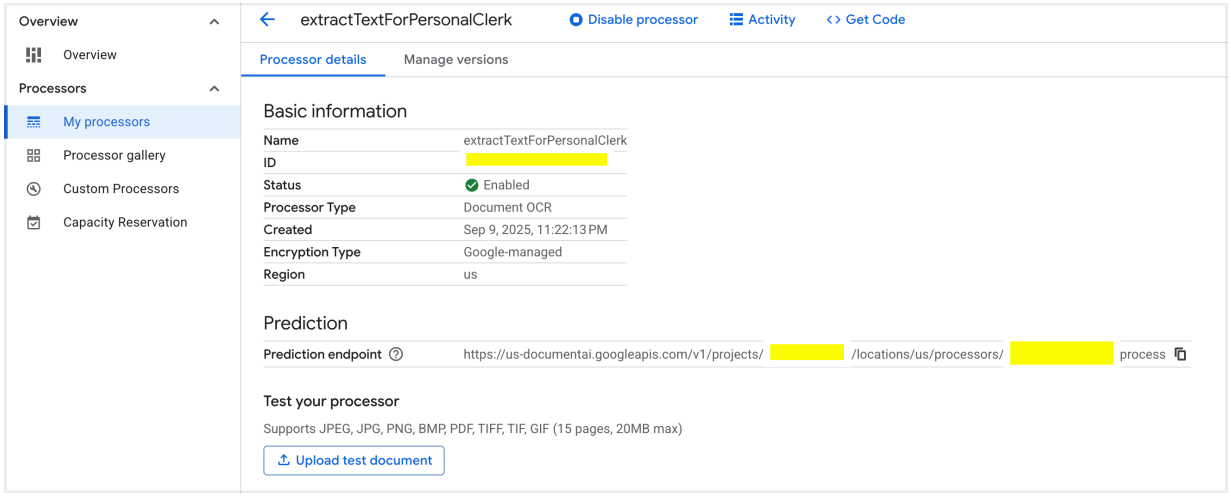
3. Google Cloud command to check if Cloud Document AI & DLP API is enabled

```
lalamanil@Anils-MacBook-Pro PassPort&Visa % gcloud services list
NAME                                    TITLE
aiplatform.googleapis.com               Vertex AI API
analyticshub.googleapis.com             Analytics Hub API
artifactregistry.googleapis.com         Artifact Registry API
bigquery.googleapis.com                 BigQuery API
bigqueryconnection.googleapis.com       BigQuery Connection API
bigquerydatapolicy.googleapis.com       BigQuery Data Policy API
bigquerymigration.googleapis.com        BigQuery Migration API
bigqueryreservation.googleapis.com      BigQuery Reservation API
bigquerystorage.googleapis.com          BigQuery Storage API
cloudbuild.googleapis.com               Cloud Build API
cloudtrace.googleapis.com               Cloud Trace API
compute.googleapis.com                  Compute Engine API
containerregistry.googleapis.com        Container Registry API
dataform.googleapis.com                 Dataform API
dataplex.googleapis.com                 Cloud Dataplex API
datastore.googleapis.com                Cloud Datastore API
dlp.googleapis.com                      Sensitive Data Protection (DLP)
documentai.googleapis.com               Cloud Document AI API
eventarc.googleapis.com                 Eventarc API
firebaserules.googleapis.com            Firebase Rules API
firestore.googleapis.com                Cloud Firestore API
iam.googleapis.com                      Identity and Access Management (IAM) API
iamcredentials.googleapis.com           IAM Service Account Credentials API
logging.googleapis.com                  Cloud Logging API
monitoring.googleapis.com               Cloud Monitoring API
oslogin.googleapis.com                  Cloud OS Login API
pubsub.googleapis.com                   Cloud Pub/Sub API
run.googleapis.com                      Cloud Run Admin API
servicemanagement.googleapis.com        Service Management API
serviceusage.googleapis.com             Service Usage API
sql-component.googleapis.com            Cloud SQL
storage-api.googleapis.com              Google Cloud Storage JSON API
storage-component.googleapis.com        Cloud Storage
storage.googleapis.com                  Cloud Storage API
texttospeech.googleapis.com             Cloud Text-to-Speech API
lalamanil@Anils-MacBook-Pro PassPort&Visa %
```

4. Create a Document OCR Processor to Identify and extract text in different types of documents







5. Create a service account and provide access to Cloud Document AI and DLP API. Below is Google cloud command

```
lalamanil@Anils-MacBook-Pro PassPort&Visa % gcloud projects get-iam-policy videoanalyzer-455321 --flatten="bindings[].members" --format='table(binding
s.members, bindings.role)' --filter="bindings.members:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com"
MEMBERS                                                                                                ROLE
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/aiplatform.user
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/bigquery.admin
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/cloudfunctions.developer
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/cloudfunctions.invoker
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/datastore.owner
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/dlp.admin
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/documentai.apiUser
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/pubsub.admin
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/run.developer
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/run.invoker
serviceAccount:ai-projects-service-account@videoanalyzer-455321.iam.gserviceaccount.com  roles/storage.admin
lalamanil@Anils-MacBook-Pro PassPort&Visa %
```

Technical Walk through:

1. Maven dependency required

```xml
<dependencies>
    <dependency>
        <groupId>com.google.cloud</groupId>
        <artifactId>google-cloud-document-ai</artifactId>
    </dependency>

    <dependency>
        <groupId>com.google.cloud</groupId>
        <artifactId>google-cloud-dlp</artifactId>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.19.2</version>
    </dependency>

    <dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>itext7-core</artifactId>
        <version>9.2.0</version>
        <type>pom</type>
    </dependency>
</dependencies>
```

2. **Extracting Raw Text and Detail Token Meta Data:**

When a user uploads a PDF or image, the document is first converted into a byte array and sent to the Document AI OCR processor along with the appropriate

MIME type. Document OCR returns the complete raw text content of the file, along with detailed information for every page—specifically, each **token's text, text offset within the raw text, and its normalized bounding polygon.**

For each token, the normalized bounding-poly **coordinates (ranging from 0 to 1) are scaled using the page's actual width and height to obtain real pixel-level coordinates.** These resolved values—page number, page dimensions, token text, text offset positions, and bounding-box coordinates—are then encapsulated into a custom TokenModel object. All tokens across all pages are stored in a page-wise structure (Map<Integer, List<TokenModel>>), forming a complete OCR extraction response (OCRResponseModel). This structured representation later enables precise masking of sensitive text on the original document.

```
String name = String.format("projects/%s/locations/%s/processors/%s", ApplicationConstants.PROJECT_ID,
        ApplicationConstants.DOCUMENT_AI_PROCESSORS_LOCATION, ApplicationConstants.PROCESSOR_ID);
ByteString content = ByteString.copyFrom(filedata);
RawDocument rawDocument = RawDocument.newBuilder().setContent(content).setMimeType(mimeType).build();
ProcessRequest processRequest = ProcessRequest.newBuilder().setName(name).setRawDocument(rawDocument)
        .build();
ProcessResponse processResponse = documentProcessorServiceClient.processDocument(processRequest);
Document document = processResponse.getDocument();
String rawText = document.getText();
ocrResponseModel.setRawText(rawText);
System.out.println("rawText:" + ocrResponseModel.getRawText());
List<Page> pageList = document.getPagesList();
if (null == pageList || pageList.isEmpty()) {
    LOGGER.info("pagelist received is null or empty for a document");
} else {
    for (int p = 0; p < pageList.size(); p++) {

        Page page = pageList.get(p);

        List<Token> tokenList = page.getTokensList();

        if (null == tokenList || tokenList.isEmpty())
            continue;

        List<TokenModel> pageTokenlist = new ArrayList<TokenModel>();

        float pageWidth = page.getDimension().getWidth();
        float pageHeight = page.getDimension().getHeight();

        for (Token token : tokenList) {
            Layout layout = token.getLayout();
            if (null == layout)
                continue;
            String tokenText = getText(layout.getTextAnchor(), rawText);
            int startindex = 0, endIndex = 0;
```

```
                        if (layout.getTextAnchor().getTextSegmentsCount() > 0) {
                            TextSegment seg = layout.getTextAnchor().getTextSegments(0);
                            startindex = (int) seg.getStartIndex();
                            endIndex = (int) seg.getEndIndex();
                        }
                        // Normalized vertices
                        List<NormalizedVertex> normalizedVertexs = layout.getBoundingPoly()
                                .getNormalizedVerticesList();
                        if (null == normalizedVertexs || normalizedVertexs.isEmpty())
                            continue;
                        float minX = Float.MAX_VALUE, minY = Float.MAX_VALUE, maxX = 0, maxY = 0;
                        for (NormalizedVertex nv : normalizedVertexs) {
                            float x = nv.getX() * pageWidth;
                            float y = nv.getY() * pageHeight;
                            minX = Math.min(minX, x);
                            minY = Math.min(minY, y);
                            maxX = Math.max(maxX, x);
                            maxY = Math.max(maxY, y);
                        }

                        pageTokenlist.add(new TokenModel(p + 1, tokenText, startindex, endIndex, minX, minY, maxX,
                                maxY, pageWidth, pageHeight));
                    }
                    if (!pageTokenlist.isEmpty()) {
                        tokenListMap.put(p + 1, pageTokenlist);
                    }
                }
                System.out.println(ocrResponseModel.getPageTokenMap().size());
            }
        }
    }
    return ocrResponseModel;
}
```

3. **Identifying Sensitive data from Raw Text using the Google Cloud DLP API**

Once the raw text is extracted from Document OCR, it is passed to the Google Cloud DLP API for sensitive data detection. The DLP inspection step uses a combination of **Google's built-in infoTypes** and **custom-defined patterns** to locate sensitive entities within the text.

To start, the system loads a predefined list of global infoTypes (such as PERSON_NAME, PHONE_NUMBER, CREDIT_CARD_NUMBER, PASSPORT, GOVERNMENT_ID, etc.). These are added to the DLP inspection configuration so the API can detect a wide spectrum of personal, financial, and document identifiers.

In addition to built-in detectors, custom infoTypes are defined for domain-specific patterns that are not natively supported. For example, a custom regex is used to detect **USCIS receipt numbers**, and another pattern identifies masked or formatted **account numbers**. These custom detectors are added alongside the global infoTypes, enabling the system to capture both common and specialized sensitive patterns.

The raw text is then wrapped in a ContentItem and submitted to the DLP API's inspectContent() method. The API processes the content and returns a list of **Findings**, each containing:

- Detected sensitive value (quote
- Its infoType category

- ✅ Offset positions in the raw text
- ✅ Confidence score

These findings form the basis for the next stage, where each sensitive region is mapped back to the corresponding OCR tokens and masked on the original document with exact spatial accuracy.

```
public static List<Finding> inspectWithDlp(String rawString) {
    List<Finding> findings = new ArrayList();
    if (null == dlpServiceClient) {
        LOGGER.info("dlpServiceClient is null or empty. Please check application logs");
    } else {

        List<InfoType> infoTypes = new ArrayList<InfoType>();

        ApplicationConstants.globalInfoTypes.forEach(infotype -> {
            infoTypes.add(InfoType.newBuilder().setName(infotype).build());
        });

        CustomInfoType uscisReceiptType = CustomInfoType.newBuilder()
                .setInfoType(InfoType.newBuilder().setName("USCIS_RECEIPT_NUMBER"))
                .setRegex(Regex.newBuilder().setPattern("[A-Z]{3}[0-9]{10}")).build();

        CustomInfoType accountNumber = CustomInfoType.newBuilder()
                .setInfoType(InfoType.newBuilder().setName("Account Number"))
                .setRegex(Regex.newBuilder()
                        .setPattern("((Account #|Account number:)\\s*)((\\d{4}\\s*){2}\\d{4}|\\d+)")
                        .addGroupIndexes(3))
                .build();

        InspectConfig inspectConfig = InspectConfig.newBuilder().addAllInfoTypes(infoTypes)
                .addCustomInfoTypes(uscisReceiptType).addCustomInfoTypes(accountNumber).setIncludeQuote(true)
                .build();

        ContentItem contentItem = ContentItem.newBuilder().setValue(rawString).build();

        String parent = String.format("projects/%s/locations/global", ApplicationConstants.PROJECT_ID);

        InspectContentRequest request = InspectContentRequest.newBuilder().setParent(parent)
                .setInspectConfig(inspectConfig).setItem(contentItem).build();

        try {
            InspectContentResponse response = dlpServiceClient.inspectContent(request);
            if (null != response && null != response.getResult()) {
                findings = response.getResult().getFindingsList();
            }
        } catch (ApiException e) {
            // TODO: handle exception
            e.printStackTrace();
        }

    }

    return findings;

}
```

Below are the few build in info types for location global to identify sensitive data

```java
public static final List<String> globalInfoTypes = Arrays.asList(
        // FINANCIAL & BANKING (22)
        "FINANCIAL_ACCOUNT_NUMBER", "CREDIT_CARD_NUMBER", "IBAN_CODE", "SWIFT_CODE", "US_BANK_ROUTING_MICR",
        "CANADA_BANK_ACCOUNT", "JAPAN_BANK_ACCOUNT", "PORTUGAL_NIB_NUMBER", "US_EMPLOYER_IDENTIFICATION_NUMBER",
        "US_INDIVIDUAL_TAXPAYER_IDENTIFICATION_NUMBER", "AUSTRALIA_TAX_FILE_NUMBER", "UK_TAXPAYER_REFERENCE",
        "VAT_NUMBER", "FINANCIAL_ID", "CREDIT_CARD_EXPIRATION_DATE", "CREDIT_CARD_TRACK_NUMBER", "CVV_NUMBER",
        "AMERICAN_BANKERS_CUSIP_ID", "SPAIN_CIF_NUMBER", "US_ADOPTION_TAXPAYER_IDENTIFICATION_NUMBER",
        "PORTUGAL_NIB_NUMBER", "CANADA_OHIP",

        // PII CORE & LOCATION (10)
        "PERSON_NAME", "FIRST_NAME", "LAST_NAME", "EMAIL_ADDRESS", "PHONE_NUMBER", "DATE_OF_BIRTH",
        "STREET_ADDRESS", "LOCATION", "LOCATION_COORDINATES", "AGE",

        // GOVERNMENT IDs & PASSPORTS (30)
        "GOVERNMENT_ID", "PASSPORT", "US_PASSPORT", "US_SOCIAL_SECURITY_NUMBER", "US_DRIVERS_LICENSE_NUMBER",
        "CANADA_SOCIAL_INSURANCE_NUMBER", "CANADA_DRIVERS_LICENSE_NUMBER", "UK_NATIONAL_INSURANCE_NUMBER",
        "UK_DRIVERS_LICENSE_NUMBER", "IRELAND_PPSN", "CHINA_RESIDENT_ID_NUMBER", "INDIA_AADHAAR_INDIVIDUAL",
        "INDIA_PAN_INDIVIDUAL", "BRAZIL_CPF_NUMBER", "GERMANY_IDENTITY_CARD_NUMBER", "MEXICO_CURP_NUMBER",
        "NETHERLANDS_BSN_NUMBER", "FRANCE_CNI", "FRANCE_NIR", "SPAIN_DNI_NUMBER", "JAPAN_INDIVIDUAL_NUMBER",
        "AUSTRALIA_DRIVERS_LICENSE_NUMBER", "AUSTRALIA_MEDICARE_NUMBER", "POLAND_PESEL_NUMBER",
        "THAILAND_NATIONAL_ID_NUMBER", "UK_NATIONAL_HEALTH_SERVICE_NUMBER", "IRELAND_EIRCODE",
        "UK_ELECTORAL_ROLL_NUMBER", "GENDER", "DRIVERS_LICENSE_NUMBER",

        // SECURITY & TECHNICAL (13)
        "GCP_API_KEY", "GCP_CREDENTIALS", "AWS_CREDENTIALS", "AUTH_TOKEN", "OAUTH_CLIENT_SECRET", "JSON_WEB_TOKEN",
        "PASSWORD", "ENCRYPTION_KEY", "IP_ADDRESS", "MAC_ADDRESS", "BASIC_AUTH_HEADER", "XSRF_TOKEN", "URL"

);
```

4. **Mapping Sensitive Data Findings from Google Cloud DLP API to OCR Tokens.**

At this stage, we have two key datasets:

1. **OCR Tokens** extracted by Document AI, each containing:
    - token text
    - character offset within the raw text
    - pixel-level bounding box
    - page width/height

2. **Sensitive Data Findings** returned by Cloud DLP, each containing:
    - infoType (e.g., EMAIL_ADDRESS, PASSPORT, ACCOUNT_NUMBER)
    - text segment value
    - start and end byte offsets

Both systems refer to regions inside the *same raw text*, which enables accurate mapping.
To determine which document regions must be masked, we compare the offset range of each DLP finding with the offset range of each OCR token. The byte offsets returned by DLP are first converted to character offsets to align with the token model. For each finding, every OCR token is checked for overlap using the condition:

token_end > finding_start AND token_start < finding_end

If a token's offset intersects with a DLP finding's offset range, that token is considered sensitive and added to the mask list.

This mapping step produces a final list of TokenModel objects representing the exact regions on the document—across all pages—that must be redacted. These token-level coordinates are then used to draw masking rectangles on top of the original PDF, enabling high-precision, layout-preserving redaction.

```
    private static List<TokenModel> getTokenModelOfFindings(byte[] filebytes, String mimeType) {
        List<TokenModel> tokenToMask = new ArrayList<TokenModel>();
        OCRResponseModel ocrResponseModel = processDocument(filebytes, mimeType);
        String rawtext = ocrResponseModel.getRawText();
        if (null == rawtext) {
            LOGGER.info("rawtext from document OCR is null");
        } else {
            List<Finding> findings = CloudDLPUtility.inspectWithDlp(rawtext);
            if (null != findings && !findings.isEmpty()) {
                for (Finding f : findings) {
                    if (!f.hasLocation() || !f.getLocation().hasByteRange())
                        continue;
                    long start = f.getLocation().getByteRange().getStart();
                    long end = f.getLocation().getByteRange().getEnd();
                    int charStart = getCharacterIndex((int) start, rawtext);
                    int charEnd = getCharacterIndex((int) end, rawtext);
                    System.out.println(f.getInfoType().getName() + ":" + charStart + ":" + charEnd);
                    System.out.println(rawtext.substring((int) charStart, (int) charEnd));
                    for (List<TokenModel> pageTokens : ocrResponseModel.getPageTokenMap().values()) {
                        for (TokenModel token : pageTokens) {
                            int tStart = token.getStartIndex();
                            int tend = token.getEndIndex();

                            if (tend > charStart && tStart < charEnd) {
                                tokenToMask.add(token);
                            }

                        }
                    }

                }

            } else {
                LOGGER.info("findings is null or empty");
            }
        }
        return tokenToMask;
    }
```

5. Masking Sensitive Data on the Original PDF/Image

Once the sensitive tokens are identified, the next step is to mask those regions on the original document.
This redaction process involves several careful transformations because:

- Document AI uses normalized coordinates
- PDF uses a bottom-left coordinate system
- Images use a top-left coordinate system
- Original documents may be rotated
- Each page may have different dimensions from the OCR-inferred dimensions

To ensure pixel-perfect masking, the system performs the following steps:

🔷 Step 1 — Load the Original Document

For PDFs, iText7 is used to read/write document pages.
For images, the file is loaded into a **BufferedImage**.
Each page's actual width, height, and rotation angle are extracted from the PDF/Image metadata.

🔷 Step 2 — Handle Page Rotation

Document AI automatically normalizes text orientation while extracting tokens.
However, the original PDF may have been rotated (90°, 180°, 270°, or none).
To align bounding boxes correctly:

- The actual PDF rotation is checked.
- If rotation information is missing, a custom inference method compares:
  - PDF aspect ratio
  - Document AI page aspect ratio

This logic determines whether the Document AI bounding boxes need to be re-rotated before redaction.

🔷 Step 3 — Coordinate Scaling

Document AI provides normalized bounding boxes (0–1 range).
These are first converted to absolute coordinates using Document AI's page width and height.

Since the original PDF or image may have different dimensions:
scaledX = docAI_X * (actualPdfWidth / docAI_PageWidth)
scaledY = docAI_Y * (actualPdfHeight / docAI_PageHeight)

Separate scaling factors are calculated for:

- **X axis:** actualPageWidth / docAIWidth
- **Y axis:** actualPageHeight / docAIHeight

🔷 Step 4 — Convert Coordinate Systems

Document AI:

- Uses **top-left origin**, with **Y increasing downward**

PDF:

- Uses **bottom-left origin**, with **Y increasing upward**

Images:

- Use **top-left origin** (Java's default)

Therefore:

⭐ Converting Document AI → PDF Coordinates

pdfY = actualPdfHeight - docAI_Y

⭐ Converting Document AI → Image Coordinates

No Y inversion needed, but rotation compensation is applied.

🔷 Step 5 — Drawing Mask Rectangles

For each token:

- The transformed bounding rectangle is calculated.
- A small padding is applied for visual consistency.
- A gray (204,204,204) box is drawn using:

PDF

iText7 **PdfCanvas**

Images

Java **Graphics2D**

Each rectangle precisely covers the sensitive region without altering the rest of the page layout.

```
public static byte[] maskPdfTokenks(byte[] inputPdfBytes, Map<Integer, List<TokenModel>> pageTokenModelMap) {

    if (null == inputPdfBytes || null == pageTokenModelMap || pageTokenModelMap.isEmpty()) {
        return null;
    }

    // PdfReader reads from the input byte array stream.
    ByteArrayInputStream inputStream = new ByteArrayInputStream(inputPdfBytes);

    // PdfWriter writes to this output byte array stream.
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    PdfReader reader = null;
    PdfWriter writer = null;
    PdfDocument pdfDocument = null;
    try {
        reader = new PdfReader(inputStream);
        writer = new PdfWriter(outputStream);
        pdfDocument = new PdfDocument(reader, writer);
        Set<Map.Entry<Integer, List<TokenModel>>> entrySet = pageTokenModelMap.entrySet();
        DeviceRgb maskColor = new DeviceRgb(204, 204, 204);
        float padding = 1.0f;
        for (Map.Entry<Integer, List<TokenModel>> entry : entrySet) {
            int pageNumber = entry.getKey();
            List<TokenModel> tokensTomask = entry.getValue();
            // get the canvas for drawing content over the existing content
            try {

                PdfPage page = pdfDocument.getPage(pageNumber);

                float actualPageHeight = page.getMediaBox().getHeight();
                float actualPageWidth = page.getMediaBox().getWidth();

                System.out.println("itext PageNumber:" + pageNumber + " height:" + actualPageHeight);
                System.out.println("itext Pagenumber:" + pageNumber + " width:" + actualPageWidth);

                PdfCanvas canvas = new PdfCanvas(page);
                canvas.setFillColor(maskColor);
```

```
int rotationtouse = getinferrenceRotation(actualPageHeight, actualPageWidth,
        tokensTomask.get(0).getPageHeight(), tokensTomask.get(0).getPageWidth(),
        page.getRotation());

System.out.println("page Number:" + pageNumber + " Reported rotation:" + page.getRotation()
        + " inferred rotation:" + rotationtouse);

// Draw the mask for each token
for (TokenModel token : tokensTomask) {
    // --- prepare Rectangle Coordinated (using your corrected PDF bottom-up Y)

    float docAIHeight = token.getPageHeight();
    float docAIWidth = token.getPageWidth();

    float xScaleFactor;
    float yScalingFactor;
    float scaledMinX;
    float scaleMaxX;
    float scaledDocAITopY;
    float scaledDocAIBottomY;
    float lly;
    float ury;
    float llx;
    float width;
    float height;

    if (rotationtouse == 90) {

        float rotatedX_from_MinY = docAIHeight - token.getMaxY();
        float rotatedX_from_MaxY = docAIHeight - token.getMinY();

        float rotatedMinY = token.getMinX();
        float rotatedMaxY = token.getMaxX();

        float finalRotatedMinX = Math.min(rotatedX_from_MinY, rotatedX_from_MaxY);
        float finalRotatedMaxX = Math.max(rotatedX_from_MinY, rotatedX_from_MaxY);

        xScaleFactor = actualPageWidth / docAIHeight;
        yScalingFactor = actualPageHeight / docAIWidth;
```

```
        scaledMinX = finalRotatedMinX * xScaleFactor;
        scaleMaxX = finalRotatedMaxX * xScaleFactor;

        scaledDocAITopY = rotatedMinY * yScalingFactor;
        scaledDocAIBottomY = rotatedMaxY * yScalingFactor;

    } else {

        // if rotationtouse is 0
        xScaleFactor = actualPageWidth / docAIWidth;
        yScalingFactor = actualPageHeight / docAIHeight;

        scaledMinX = token.getMinX() * xScaleFactor;
        scaleMaxX = token.getMaxX() * xScaleFactor;

        scaledDocAITopY = token.getMinY() * yScalingFactor;
        scaledDocAIBottomY = token.getMaxY() * yScalingFactor;

    }

    // new y-axis inversion (using iText's true height)
    lly = actualPageHeight - scaledDocAIBottomY;
    ury = actualPageHeight - scaledDocAITopY;
    llx = scaledMinX;
    width = scaleMaxX - llx;
    height = ury - lly;
    // apply padding
    llx = llx - (padding / 2);
    lly = lly - (padding / 2);
    width = width + padding;
    height = height + padding;

    Rectangle rect = new Rectangle(llx, lly, width, height);
    canvas.rectangle(rect).fill();
}

canvas.release();
```

```
                } catch (PdfException e) {
                    // TODO: handle exception
                    e.printStackTrace();
                }

            }

        } catch (IOException e) {
            // TODO: handle exception
            e.printStackTrace();
        } finally {
            if (null != pdfDocument) {
                pdfDocument.close();
            }
        }

        return outputStream.toByteArray();
    }
```

Output:

Some sample bank statement pdf document is provide to the system. Sensitive information is masked with grey solid boxes.

```
package com.document.sensitive.mask.app;

import com.document.sensitive.mask.app.utility.DocumentOCR;

/**
 * @author lalamanil
 */
public class App {
    public static void main(String[] args) {

        String filePath = "/Users/lalamanil/Documents/Anil LALAM/canadavisaphase2/BankStatements/eStmt_2023-11-06.pdf";

        DocumentOCR.performMasking(filePath);

    }
}
```

BANK OF AMERICA

P.O. Box 15284
[redacted], DE 19850

Customer service information

Customer service: [redacted]

En Español: [redacted]

[redacted]

Bank of America, N.A.
P.O. [redacted]

[redacted] LALAM
[redacted]

Your Bank of America Advantage Savings

for October 11, 2023 to November 8, 2023

Account number: [redacted]

[redacted] LALAM