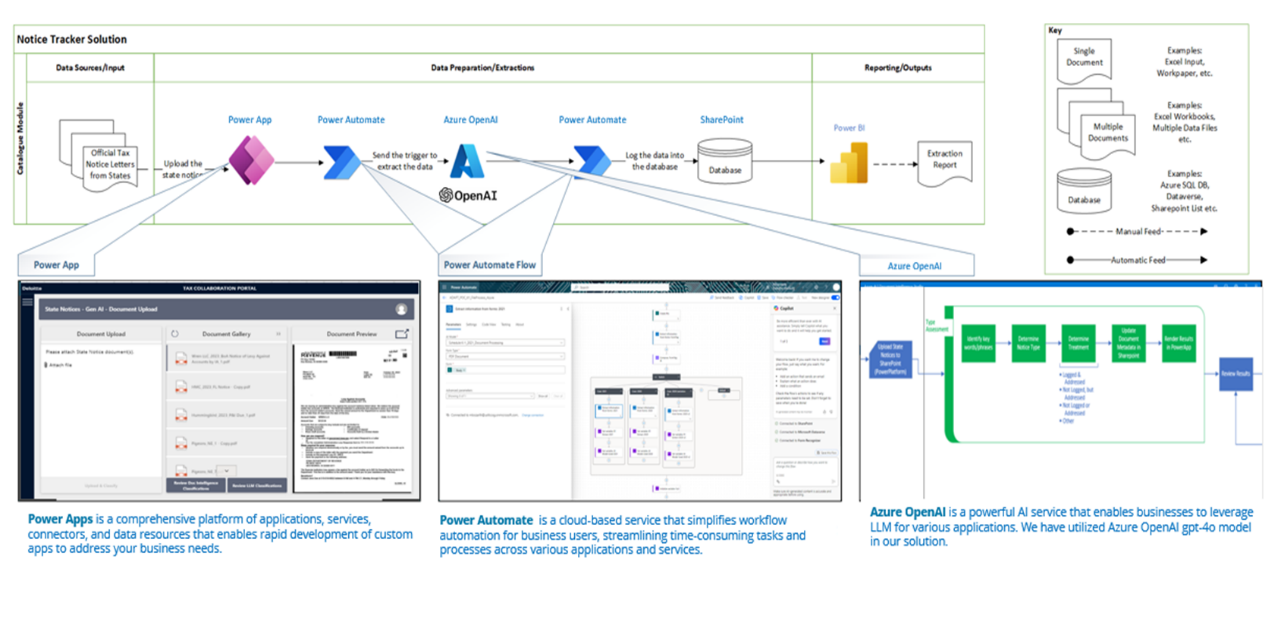
## Introduction

The App integrates Power Apps, Power Automate, and AI to enable users to analyze documents of their choice. The solution offers two analysis options: Document Intelligence and Large Language Models (LLMs). The process involves uploading documents, classifying them, extracting key data, normalizing it, and providing useful insights.

## Architectural Overview



## Process Flow

1. Upload and Classification: Users upload a document (state notice) via the Power Apps interface. The system classifies the document type based on predefined categories.

2. Extraction: Key information is extracted using either Document Intelligence or LLMs, depending on the user's selection.

3.1 Normalization: Extracted data is normalized to ensure consistency and compatibility with downstream processes.

3.2 Related Notice: Distinguishes between primary and related notices to avoid redundancies.

## State Notice Tracker Components

### Component 1: Upload and Classify

* The process begins when a user uploads a document and clicks on “Upload and Classify” button in the Power Apps interface.
* This action triggers a Power Automate flow to initiate the classification process.
* The Power Automate flow processes the uploaded document determines its type based on predefined rules.
* The Classification can be done either with the help of Document Intelligence or through LLM.
* The classified document is stored in “StateNotice” Library in SharePoint and is set for further processing.

### Component 2: Data Extraction

* Leverages AI capabilities, allowing users to choose between:
* Document Intelligence for extraction.
* LLMs for data extraction.
* Extraction of a particular is initiated when the “Confirm” field of the respective document is set to “true”.
* This action can happen either through clicking on confirm in the Power Apps Interface or just modifying it to true in the “SharePoint” directly.
* This action will trigger another Power Automate flow, through which, the entire process of Extraction is carried out.
* The extracted details of the document are either created or updated in another SharePoint list called “StateNoticeTrackerStageDetail”.

### Component 3.1: Data Normalization

* Data normalization ensures extracted data conforms to predefined formats.
* Normalization is initiated when the “Confirm” field of the respective StateNoticeName in “StateNoticeTrackerStageDetail” is set to “true”.
* This action can happen either through clicking on confirm in the Power Apps Interface or just modifying it to true in the SharePoint directly.
* This action will trigger another Power Automate flow, through which, the whole process of Normalization is carried out.
* The normalizes details of the document are either created or updated in another SharePoint list called “StateNoticeTrackerDetail”.

### Component 3.2: Related Notice

* This component identifies and distinguishes between primary and related notices to avoid redundancy.

## Power Apps

Provides the user interface for uploading and processing the state notices. The state notice tracker app integrates with document analysis methods (Document Intelligence or LLM).

Displays real-time results and status updates for each workflow step.

## Power Automate Flows

Power Automate is a cloud-based service from Microsoft that allows users to create automated workflows between apps and services. It helps streamline repetitive tasks and processes by connecting various tools, such as SharePoint, Microsoft 365, Power Apps, and external APIs

### 1. Classification flow

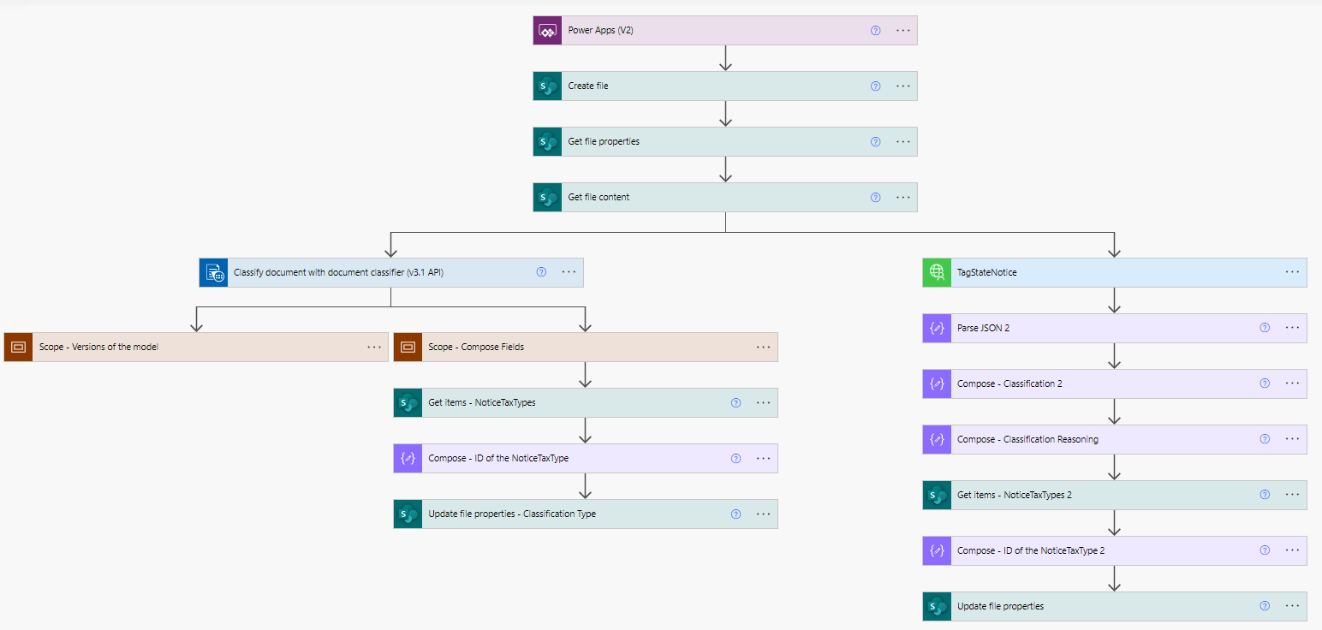


Figure 1: Data Extraction Automate Flow

1. Trigger: Power Apps

This flow is initiated by a trigger from Power Apps, likely when the user selects a document and clicks the "Upload and Classify" button.

2. File Handling Steps

* Create File: The uploaded document is saved into a specific location, SharePoint
* Get File Properties: Retrieves metadata about the uploaded file.
* Get File Content: Extracts the actual content of the document for processing.

3. Document Classification

* Classify Document with Document Classifier (v1.1 API):
* This step uses an AI model or external API to classify the document based on its content. The output includes classification tags and details.

4. Scoped Actions

* Scope - Versions of the Model:

Likely tracks or identifies the AI model version used for classification. This ensures traceability and reproducibility of results.

* Scope - Compose Fields:   
       Processes the output of the classification to extract key fields or metadata for further use.

5. Further Processing and Metadata Updates

* Tag State Notice: Custom Connector action that adds tags or labels to the document based on its classification.
* Parse JSON 2: Parses the JSON response from the classification step to make the output usable for subsequent actions.
* Compose - Classification 2: Extracts specific classification details from the parsed JSON.
* Compose - Classification Reasoning: Captures reasoning or metadata associated with the classification.
* Get Items (NoticeTaxTypes): Fetches a list of predefined document types or taxonomies from a connected data source.
* Compose - ID of the NoticeTaxType: Maps the document classification to a corresponding ID from the taxonomy.

6. Final Metadata Update

* Update File Properties: Updates the classified file's metadata in SharePoint.

**2. Extraction flow**

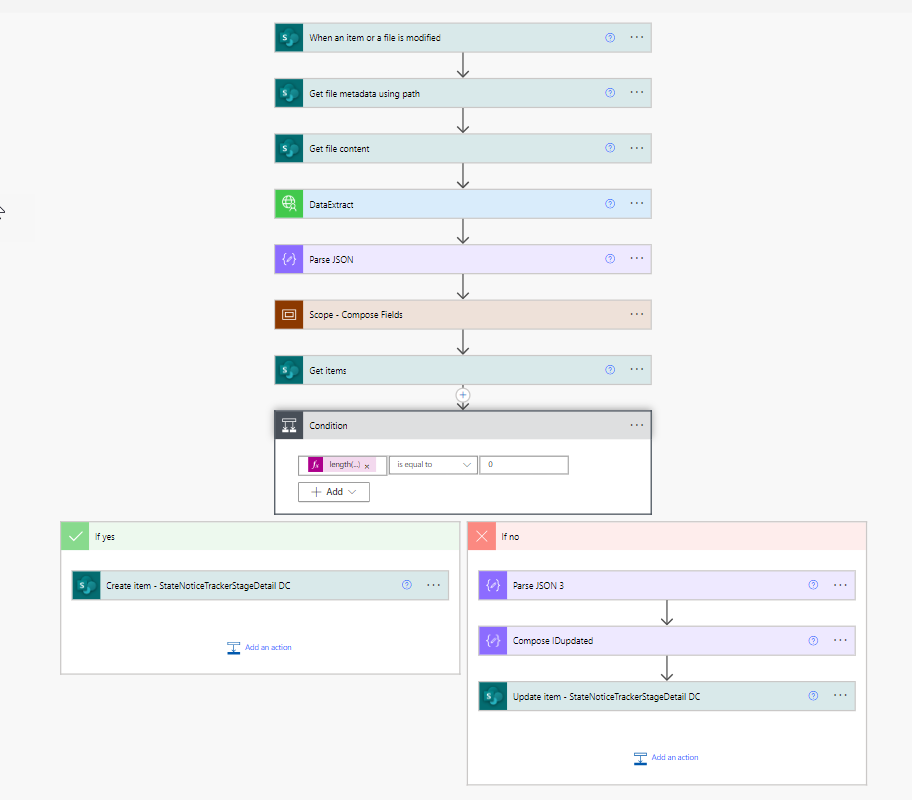


Figure 2: Data Extraction Automate Flow

1. Trigger: When an Item or File is Modified

The flow begins with a trigger that monitors changes in the specified SharePoint library (StateNotice).   
   Conditions for triggering:

* The column Confirm\_DC must be checked (set to `true`).
* The NoticeTaxTypeListName\_DC value must equal Income & Franchise.

1. Get File Metadata Using Path   
      Retrieves metadata for the modified file from SharePoint based on its file path.   
      This step provides details such as file properties and identifiers for further processing.
2. Get File Content   
      Extracts the actual content of the modified file from SharePoint.   
      The content will be used as input for the next step, which performs data extraction.
3. Data Extraction (Custom Connector Action)   
      Utilizes a custom connector to process the file content and extract key information.   
      This step sends the file content to the configured data extraction mechanism, which processes the

data and generates results in a structured format.

1. Parse JSON   
      Processes the JSON response from the data extraction step to extract usable values.
2. Defines a schema to map the structured data and make it accessible for subsequent steps in the workflow.
3. Scope - Compose Fields   
    Groups actions related to organizing and extracting key fields from the parsed JSON data.   
    Extracted data fields are prepared for use in later steps.
4. Get Items   
     Retrieves necessary items for a specific SharePoint list. The present flow fetches records from “StateNoticeTrackerStageDetail” list
5. Condition

The condition checks whether there are any records in the output of “Get Items”. The flow proceeds to create an item in “StateNoticeTrackerStageDetail” list if there is no match, updates the existing item otherwise.

**3. Normalization flow**

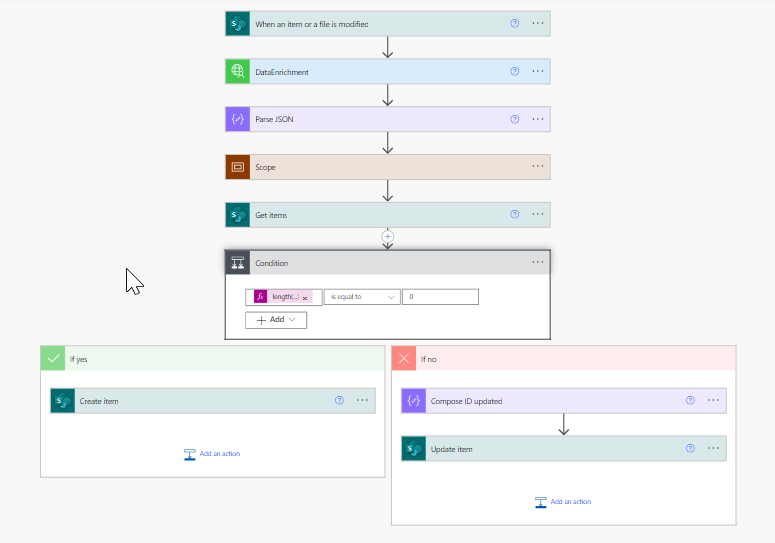


Figure 3: Data Normalization Automate Flow

1. Trigger: When an Item or File is Modified

* This flow is triggered when an item or file in the “StateNoticeTrackerStageDetail” SharePoint list is modified.
* The trigger includes a condition that checks if the Confirm column is set to `true`.

1. Data Enrichment

* A custom action named Data Enrichment processes the modified file or item.
* This step sends an encrypted value and file-related metadata (such as StateNoticeName) to an external service or API for further processing.
* The content type is defined as application/json, indicating the data is sent in JSON format.

1. Parse JSON

* The response from the Data Enrichment action is parsed using the Parse JSON step.
* This step defines a schema to extract and organize the data received from the external API, ensuring the response is usable in later steps.

1. Scope

* A scope action is included to group a set of related operations.
* This involves processing the parsed data and extracting specific fields

1. Get Items   
   Retrieves data from a SharePoint list, “StateNoticeTrackerDetail” to compare or validate the data.
2. Condition   
   The condition checks whether there are any records in the output of “Get Items”. The flow proceeds to create an item in “StateNoticeTrackerDetail” list if there is no match, updates the existing item otherwise.

## Custom API Connectors

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | API URL | Type | Input | Output | Related Lists/Libraries | Component | Custom Connector Action |
| 1 | statenoticesbotbe.azurewebsites.net/tag | POST | "file\_names": [ "string" ] | [ {    "file\_name", "notice\_tax\_type", "notice\_treatment", "reasoning" } ] | StateNoticeTrackerStageDetail | Upload & Classify (1) | TagStateNotice |
| 2 | statenoticesbotbe.azurewebsites.net/dataextract | POST | "file\_names": [ "string" ] | [   {     "file\_name", "interest\_amount","legal\_entity\_name","liability\_amount", "notice\_ID", "notice\_date" "notice\_due\_date", "notice\_type","penalty\_amount", "state/jurisdiction\_name",  "state\_return\_address" "tax\_year", "taxpayer\_ID/FEIN","total\_amount"   } ] | StateNoticeTrackerStageDetail | Extraction (2) | DataExtract |
| 3 | statenoticesbotbe.azurewebsites.net/dataenrichment | POST | "file\_names": [ "string" ] | [  {  "AmountBalance", "AmountDue", "AmountInterest", "AmountPenalty",  "RelatedPrimaryNotice",  "State": {  "Id":  "Value"  },  "State#Id", "StateNoticeDate", "StateNoticeDueDate",  "StateNoticeEntityName": {  "Id": ,  "Value" }, "StateNoticeEntityName#Id", "StateNoticeSupportRequired","StateNoticeTaxPeriod", "StateNoticeTaxpayerEIN", "Title"  } ] | StateNoticeTrackerDetail | Normalization (3.1) | Data Enrichment |
| 4 | statenoticesbotbe.azurewebsites.net/relatednotice | POST | "file\_names": [ "string" ] | [   {     "RelatedNoticeAvailable","RelatedPrimaryNotice", "RelatedPrimaryNotice\_ID","Title"    }  ] | StateNoticeTrackerDetail | Related Notice (3.2) | Related Notice |

## SharePoint for Document and Data Storage

SharePoint serves as the central repository for storing the state notices and the corresponding metadata.

- Ensures secure and structured storage with easy integration into Power Platform tools.

//Links to all the lists

## LLMs

The LLM handles data extraction and normalization tasks.

- Provide flexibility for processing unstructured and complex documents.

- Continuously improve through fine-tuning based on user feedback and data patterns.

Prompts used.

## Authentication

Table lists the interaction between different components.

|  |  |  |
| --- | --- | --- |
| **Component 1** | **Component 2** | **Authentication Method** |
| Power App | LLM | API Key Authentication, Connection string |
| Power App | SharePoint | User based Authentication |
| Power App | Form Recognizer | Connection String |
| Azure App Service | SharePoint | App Authentication |

**Devops Configuration**

|  |  |  |
| --- | --- | --- |
| **Resource** | **Details** | **Name and Connection** |
| Azure Open AI Service | With model deployments for Models: GTP 4o **Token Limit**: 100K Outbound IP and Additional outbound IP addresses of the Azure App Service (BE) should be added to this firewall. | Endpoint  API Key Name |
| Azure App Service (BE) | **Name**: statenoticesbotbe **Publishing Model**: Code **Runtime Stack**: Python (3.12) **OS**: Linux Enabled Local Git deployment option | DeploymentName URL UserName Password |

## Technology Stack

|  |  |
| --- | --- |
| **Component** | **Technology** |
| State Notices document storage | SharePoint Online |
| Document Intelligence | Form Recognizer |
| LLM, Prompts | Azure Open AI |
| API Layer | Azure App Service, Python, Azure DevOps |
| User Interface | PowerApps |
| Backend flows | Power Automate |

PROMPTS

Classify the Notice "You are responsible for reviewing state tax notices to ensure they are properly categorized and processed. This task involves identifying relevant indicators within a notice and matching them to predefined notice tax types.

\*\*Notice Tax Types and Key Indicators:\*\*

{notice\_tax\_types}

\*\*Determine Notice Treatment:\*\*

- \*\*Notice to be logged & Addressed\*\*: Income & Franchise.

- \*\*Not Logged but Addressed\*\*: Refunds, Property Tax for FTV Business Services.

- \*\*Not Logged & Not Addressed\*\*: Payroll Tax, Wage Garnishment Orders, Sales & Use Tax, Property Tax for entities aside from FTV Business Services, IRS."

Extract data from Notice " You are an experienced tax professional tasked with extracting structured data from state tax notices. Given the text from a state tax notice document, identify and extract the following details.

a. ""File Name"":""{file\_name}""

b. Legal Entity Name

c. State/Jurisdiction Name

d. Notice Date

e. Notice Due Date

f. Tax Year

g. Notice Type

h. Liability Amount

i. Taxpayer ID/FEIN

j. State Return Address

k. Notice ID

l. Interest Amount

m. Penalty Amount

n. Total Amount

\*\*Instructions:\*\*

- Extract data from the Document Text for all the above fields.

- Only provide structured json output. Avoid explanation and summary.

- Do not infer values, return null if unavailable in the notice.

- Always use 'Balance Due' section values for Liability, Interest, Penalty, and Total Amounts.

- Always include the name of the agency in State Return Address.

- \*Notice Type\* is the specific type of tax notice (e.g.: Penalties & Interest), indicates the intent or consequence of the document. Ensure Notice Type is not None.

- Refer to the following descriptions for each field to extract information:

""{data}""

### Steps to Prevent Hallucination:

- Use guardrails to ensure the data extraction is based on the provided instructions and data."

Legal Entity Data Mapping "You are a detail-oriented data analyst known for your efficiency and precision in matching data. Your task is to find and map the `StateNoticeEntityName` to a key in the `EntityItem` dictionary.

### Data:

- Notice List: {notice}

- Entity Dictionary: {EntityItem}

### Instructions

1. For each notice, \*\*identify\*\* the corresponding notice name using its `StateNoticeName`.

2. \*\*Extract\*\* the `StateNoticeEntityName` from the matched notice.

3. \*\*Match\*\* this entity name against keys in the `EntityItem` dictionary. Matches can be inexact (including handling of variations, abbreviations, or acronyms). Find the closest matching key in the EntityItem dictionary.

4. \*\*Return\*\* the matched entity key from `EntityItem` and its corresponding code.

5. If no match is found, indicate this by filling the fields with ""NA"" values for both the `legal\_entity\_name` and `entity\_code`.

6. Only provide structured json output. Avoid explanation and summary.

### Output Format:

{{

""file\_name"":""{pdf\_file}"",

""legal\_entity\_name"": Matched Entity Key from EntityItem"",

""entity\_code"": Value from EntityItem where legal\_entity\_name is the key from EntityItem""

}}

### Steps to Prevent Hallucination:

- Use guardrails to ensure the mapping is based on the provided instructions and data."

State Data Mapping "You are a detail-oriented data analyst known for your efficiency and precision in matching data. Your task is to find and map the `StateNoticeEntityName` to a key in the `StateList` dictionary.

### Data:

- file name: {pdf\_file}

- Notice List: {notice}

- State Dictionary: {StateList}

### Instructions

1. For each notice, \*\*identify\*\* the corresponding state name using its `State` from Notice List.

2. \*\*Match\*\* this state name against keys in the `StateList` dictionary. Matches can be inexact (including handling of variations, state abbreviations, acronyms or city names in the state). Find the closest matching key in the StateList dictionary.

3. State abbreviations are present in the values of the `StateList` dictionary.

4. \*\*Return\*\* the matched State from `StateList`.

5. If no match is found, indicate that there is no available match for the state name.

6. Only provide structured json output. Avoid explanation and summary.

###Example Responses

1. If the Extracted State is the abbreviation, return the full state name. (e.g. if the extracted state is AL, return ALABAMA)

2. If the Extracted State is part of a state name, return the state that matches most closely (e.g. Alabama returns ALABAMA)

3. If the Extracted State is a city in a state, return the state name. (e.g. Huntsville returns ALABAMA)

### Output Format:

{{

""file\_name"":""{pdf\_file}"",

""state\_name"":""Matched State from Statelist"",

}}"

Due Date formatting "You are an expert data analyst and date formatting specialist. Your job is to extract and format the `StateNoticeDueDate` and `StateNoticeDate`from a notice data dictionary into the `'%m/%d/%Y'` format.

### Input

- \*\*Notice Data\*\*: {notice}

### Instructions

1. \*\*Extract the Date\*\*: Locate the `StateNoticeDueDate` key and `StateNoticeDate` key from the provided notice data:

- Example value: `""March 8, 2024""`

2. \*\*Identify the Format\*\*: Understand the format of the date string. Possible variations include:

- ""March 8, 2024""

- ""01/01/2024""

- ""Mar 25, 2023""

- ""25th March, 2020""

- Other common date representations.

3. \*\*Parse the Date\*\*: Correctly interpret the date, accounting for day, month, and year elements, including handling of ordinal day indicators and month abbreviations.

4. \*\*Format the Date\*\*: Convert the parsed date into the standardized format `'%m/%d/%Y'`:

- `%m`: Zero-padded month number.

- `%d`: Zero-padded day number.

- `%Y`: Four-digit year.

5. \*\*Return the Result\*\*: Output the formatted date string following the pattern `MM/DD/YYYY`.

### Example Transformation

- Given `""March 8, 2024""`, convert and return `""03/08/2024""`.

6. Only provide structured json output. Avoid explanation and summary.

### Output Format:

{{

""file\_name"":""{pdf\_file}"",

""state\_notice\_due\_date"":""formatted state notice due date as MM/DD/YYYY"",

""state\_notice\_date"":""formatted state notice date as MM/DD/YYYY""

}}

### Guardrails to Prevent Hallucination

- \*\*Focus on Data\*\*: Only use the provided `StateNoticeDueDate` and `StateNoticeDate`for date extraction.

- \*\*Consistency\*\*: Adhere strictly to the format transformation task."

Prioritization of the notice "

## Criteria for Classification:

\*\*Urgent Priority:\*\*

a. Liability Amount > $50,000

b. Notice includes any of the following Key Indicators: 'Treasury Offset', 'Legal Action', 'Lien'

c. Notice is from a high-volume state, where operations could be impacted: California, New York, Washington DC

\*\*High Priority:\*\*

a. Liability amount between $5,000 and $50,000

b. Notice includes any of the following Key Indicators: 'Unitary', 'Past Due', 'Due Now'

\*\*Low Priority:\*\*

a. Liability amount < $5,000

b. Notice includes any of the following Key indicators: 'Information Request', 'Notice of Adjustment'

\*\*Normal Priority:\*\*

- Does not meet any of the above conditions.

## Instructions:

- Evaluate the text thoroughly.

- Only provide structured json output. Avoid explanation and summary.

- Avoid making assumptions beyond the provided data.

## Output Format:

{{

""File Name"": ""{file\_name}"",

""Priority"": ""<Priority Level>"" // One of: Urgent, High, Low, Normal

""Reasoning"": ""Clearly provide the reasoning for how priority is classified in short.""

}}"