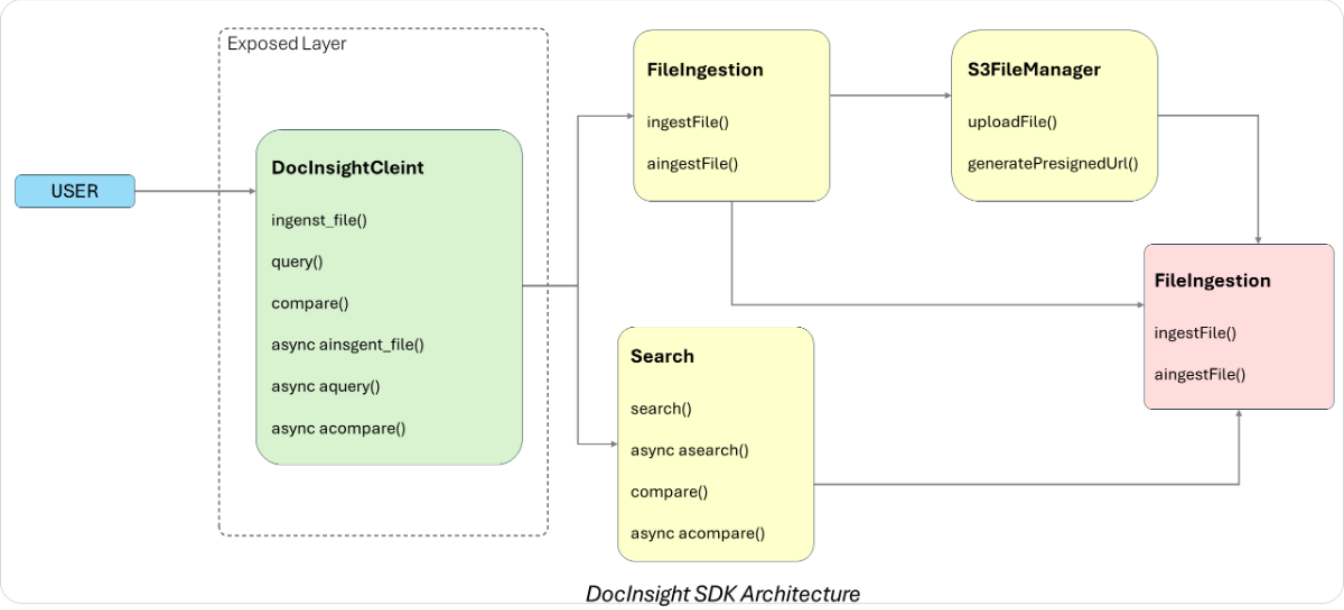
DocInsights SDK

Introduction
The DocInsights SDK is a Python library designed to interact with the DocInsights API, providing a simplified interface for developers to perform various operations such as file ingestion, document search, comparison, and summarization. The SDK abstracts the complexity of API interactions, authentication, and file handling, making it easier for developers to integrate DocInsights functionality into their applications.

Overview
The SDK is structured around the following key components:

1. DocInsightsClient: The main class that exposes the SDK's functionality to the user.
2. FileIngestion: Handles file ingestion operations into the RAG (Retrieval- Augmented Generation) system.
3. Search: Manages search, comparison, and summarization operations.
4. Authentication: Handles authentication using client credentials.
5. S3FileManager: Manages file uploads to S3 and generates presigned URLs for file access.



DocInsight SDK Architecture

Usage

Initilization

To start using the SDK, initialize the DocInsights Client with your client credentials and other optional parameters:

from docinsights import DocInsightsClient

```
client = DocInsightsClient(
client_id="your_client_id",
client_secret="your_client_secret",
base_url="https://api.docinsights.com",
s3_bucket_name="your_s3_bucket",
s3_path_within_bucket="your_s3_path"
)
```

File Ingestion

The SDK allows you to ingest files into the system for further processing. The files can be uploaded from a local path or provided as an S3 presigned URL.

Ingesting a File

```
# Ingest a file from a local path
request_id = await client.aingest_file("path/to/local/file.pdf", metadata=
{"description": "Sample document"}, **kwargs)
# Ingest a file synchronously
request_id = client.ingest_file("path/to/local/file.pdf", metadata={"description":
"Sample document"}, **kwargs)
```

Checking Ingestion Status

```
# Asynchronously check ingestion status
status = await client.ainjestion_status(request_id, **kwargs)
# Synchronously check ingestion status
status = client.injestion_status(request_id, **kwargs)
```

Document Search

The SDK provides functionality to search ingested documents based on a query.

Performing a Search

```
# Asynchronously search documents
search_results = await client.aquery(query="search query", context=["additional
context"], metadata={"filter": "value"}, **kwargs)
# Synchronously search documents
search_results = client.query(query="search query", context=["additional context"],
metadata={"filter": "value"}, **kwargs)
```

Document Comparison

You can compare two or more documents and get a summary of the differences.

Comparing Documents

```
# Asynchronously compare documents
comparison_results = await client.acompare(metadata={"document": ["doc1.pdf",
"doc2.pdf"]}, **kwargs)
# Synchronously compare documents
comparison_results = client.compare(metadata={"document": ["doc1.pdf", "doc2.pdf"]},
**kwargs)
```

Document Summarization

The SDK allows you to generate summaries for ingested documents.

Generating a Summary

```
# Asynchronously generate a summary
summary = await client.aquery(metadata={"document": "doc1.pdf"}, summary=True,
**kwargs)
# Synchronously generate a summary
summary = client.query(metadata={"document": "doc1.pdf"}, summary=True, **kwargs)
```

Internal Classes

FileIngestion

The FileIngestion class handles the ingestion of files into the RAG system. It supports both asynchronous and synchronous operations.

Key Methods:

- **aingest_file(file_path: str, **kwargs)**: Asynchronously ingests a file.
- **ingest_file(file_path: str, **kwargs**): Synchronously ingests a file.
- **aingestion_status(request_id: str, **kwargs)**: Asynchronously checks the status of an ingestion request.
- **ingestion_status(request_id: str, **kwargs)**: Synchronously checks the status ofan ingestion request.

Search

The Search class manages search, comparison, and summarization operations.

Key Methods:

- **asearch(query: str, context: list, metadata: dict, **kwargs)summarize(metadata: dict, **kwargs)**: Synchronously generates a document summary.
- **search(query: str, context: list, metadata: dict, **kwargs)**: Synchronously searches ingested documents.
- **acompare(metadata: dict, **kwargs)**: Asynchronously compares documents.
- **compare(metadata: dict, **kwargs)**: Synchronously compares documents.
- **asummarize(metadata: dict, **kwargs)**: Asynchronously generates a document summary.
- **summarize(metadata: dict, **kwargs)**: Synchronously generates a document summary.

Authentication

The Authentication class handles the authentication process using client credentials.

Key Methods:

- authenticate(): Asynchronously authenticates and retrieves an access token.
- authenticate_sync(): Synchronously authenticates and retrieves an access token.
- get_headers(): Asynchronously returns authentication headers.
- get_headers_sync(): Synchronously returns authentication headers.

S3FileManager

The S3FileManager class manages file uploads to S3 and generates presigned URLs.

Key Methods:

- upload_file_to_s3(file_path: str, object_name: str, **kwargs): Uploads a file to S3.
- generate_presigned_url(object_name: str, expiration: int, **kwargs): Generates a presigned URL for an S3 object.
- handle_file_path(url: str, **kwargs): Handles file paths, uploading local files to S3 and returning presigned URLs.

Examples

Ingesting and Searching a Document

```
# Initialize the client
client = DocInsightsClient(client_id="your_client_id",
client_secret="your_client_secret")
# Ingest a document
request_id = client.ingest_file("path/to/document.pdf", metadata={"title": "Sample
Document"}, **kwargs)
# Check ingestion status
status = client.injestion_status(request_id, **kwargs)
print(f"Ingestion Status: {status}")
# Search the ingested document
results = client.query(query="search query", metadata={"title": "Sample Document"},
**kwargs)
print(f"Search Results: {results}")
```

Comparing Documents

```
# Compare two documents
comparison_results = client.compare(metadata={"document": ["doc1.pdf", "doc2.pdf"]},
**kwargs)
print(f"Comparison Results: {comparison_results}")
```

Generating a Summary

```
# Generate a summary for a document
summary = client.query(metadata={"document": "doc1.pdf"}, summary=True, **kwargs)
print(f"Document Summary: {summary}")
```

Asynchronous Functionality

The DocInsights SDK provides full support for asynchronous operations, allowing for non-blocking API calls. This is particularly useful for improving the performance of applications that require multiple concurrent operations, such as ingesting multiple files or performing multiple searches simultaneously.

Example of Asynchronous Usage

```
import asyncio
async def main():
client = DocInsightsClient(client_id="your_client_id",
client_secret="your_client_secret")
# Ingest a file asynchronously
request_id = await client.aingest_file("path/to/document.pdf", metadata={"title":
"Sample Document"}, **kwargs)
# Check ingestion status asynchronously
```

```
status = await client.ainjestion_status(request_id, **kwargs)
print(f"Ingestion Status: {status}")
# Search the ingested document asynchronously
results = await client.aquery(query="search query", metadata={"title": "Sample
Document"}, **kwargs)
print(f"Search Results: {results}")
# Run the async function
asyncio.run(main())
```

Conclusion

The DocInsights SDK simplifies the integration of DocInsights API functionalities into your applications. By abstracting complex operations such as authentication, file handling, and API interactions, the SDK allows developers to focus on building powerful document processing features. The SDK supports both synchronous and asynchronous operations, providing flexibility for different use cases. For further assistance, refer to the source code or contact the development team.