OmniIndex Python Library

Release 0.1.12

James Stanbridge

Apr 27, 2023

PYTHON API CONTENTS:

1 omniindex **1**
- 1.1 omniindex package 1

2 Introduction **7**
- 2.1 Getting Started 7

3 Sample code **9**
- 3.1 Set up the OmniIndex API client 9
- 3.2 get_block_schematic 10
- 3.3 get_folders 10
- 3.4 get_searchchain 11
- 3.5 get_files 12
- 3.6 run_analytic_query 12
- 3.7 post_minedata 13
- 3.8 Datasets, dataframes and pandas 14

4 Tests **15**
- 4.1 API endpoint tests 15
- 4.2 Redaction test 16
- 4.3 Count of chains in the blockchain test 16

5 Creating and Maintaining encryption keys **17**
- 5.1 Creating a key 17

6 Indices and tables **19**

Python Module Index **21**

Index **23**

CHAPTER

ONE

OMNIINDEX

1.1 omniindex package

1.1.1 Submodules

1.1.2 omniindex.api module

class omniindex.api.**OmniIndexClient**(*server*, *api_key*, *unit_name*, *block_type*, *user*)

Bases: `object`

Instantiate an Omniindex client object to interact with the Omniindex API.

Parameters

- **server** (*str*) – The server to connect to. This is the node server for your blockchain, refer to the Omniindex documentation for more information.
- **api_key** (*str*) – The API key to use for authentication.
- **unit_name** (*str*) – The name of the unit to use for the transaction.
- **user** (*str*) – The user to use for the transaction.
- **block_type** (*str*) – The type of block to use for the transaction.

For more information on the client object and elements refer to the OmniIndex Documentation.

get_block_schematic()

Omniindex API call to get the schematic of a block. Before you can do any work with Omniindex you must first get the schematic of the block you want to use. This will return a JSON string containing the schematic of the block. You can then use this schematic to create a block object. This POST method will bring back the schematic of a block that the user has access to

Parameters

- **method** (*str*) – (hard coded) HTTP request method (POST)
- **url** (*str*) – (hard coded) URL to the Omniindex API endpoint
- **payload** (*str*) – JSON string containing the unit name, server, block type, user and API key.
- **headers** (*dict*) – (hard coded) Content-Type and Accept headers.
- **response** (*str*) – Response from the API call.

Returns

JSON string containing the block schematic.

Reference to `omniindex.api.OmniIndexClient.get_block_schematic()`.

`get_folders`(*show_protected*)

Omniindex API call to get the folders in a block. This POST method will bring back the folders in a block that the user has access to. OmniIndex allows you to save 'file structure' to the blockchain. This capability allows you to securely encrypt your critical data as a file system from a variety of tools. We built the Dropblock Add-on for Google Workspace and BigQuery as a perfect implementation of this capability. And it is further enhanced by the ability to redact (partially encrypt) data from within a document into a separate block. (We'll see more of that later in the documentation. But you can do this from any tool you wish via the API.) **Notice the extra key in this JSON object: 'showProtected'. You must declare 'true', or the default value false will be used.**

Note: When set 'true', the full folder name will be returned in the result set. When set 'false', the folder name is redacted. However, you can see that one exists. [case sensitive, default is false]

This API allows an authorized user to view the folder structure of a block they have the required permissions to inspect:

Parameters

- **method** (*str*) – (hard coded) HTTP request method (POST)
- **url** (*str*) – (hard coded) URL to the Omniindex API endpoint
- **showProtected** (*str*) – str value to show protected folders.
- **payload** (*str*) – JSON string containing the unit name, server, block type, user and API key.
- **headers** (*dict*) – (hard coded) Content-Type and Accept headers.
- **response** (*str*) – Response from the API call.

Returns

JSON string containing the folders in the block.

Reference to `omniindex.api.OmniIndexClient.get_folders()`.

`get_searchchain`(*show_protected*, *search_phrase*, *search_type*)

Omniindex API call to search the entire unitname in the blockchain. This POST method will bring back documents in a block that the user has access to. Using showProtected, you can search the entire blockchain for a specific phrase. This is a powerful capability that allows you to search for documents that contain a specific phrase including redacted data. Using searchType, you can search for a specific type of search, either of just the document contents or of the titles, filenames and contents. With this API Call, we can return datasets using familiar, natural language search techniques of the encrypted data on the blockchain.

Note: This is an API call that demonstrates the power of OmniIndex. In this result set, you can see that the document is redacted. However, the search phrase is still visible. This is a powerful capability that allows you to search for documents that contain a specific phrase including redacted data without ever needing to decrypt the data. As well as that you can see the sentiment analysis of the document as well as the context (as analysed against the machine learning (narrow AI) ontology)

Parameters

- **show_protected** (*str*) – "true" or "false" (default is "false") sets if the search will include redacted content.

- **search_phrase** (*str*) – a string to search for.
- **search_type** (*str*) – "fulltext" or "files" (default is "fulltext") FullText will search the file names, folder names, content, and dates, while files will only search within the content of the files:
- **method** (*str*) – (hard coded) HTTP request method (POST)
- **url** (*str*) – (hard coded) URL to the Omniindex API endpoint
- **payload** (*str*) – JSON string containing the unit name, server, block type, user and API key.
- **headers** (*dict*) – (hard coded) Content-Type and Accept headers.
- **response** (*str*) – Response from the API call.

Returns

JSON string containing the search results.

reference to `omniindex.api.OmniIndexClient.get_searchchain()`.

`getfiles`(*show_protected*, *folder_name*)

Omniindex API call to get the files in a given *folder_name* in a block. This POST method will bring back the files in a block that the user has access to. OmniIndex allows you to save 'file structure' to the blockchain. This capability allows you to securely encrypt your critical data as a file system from a variety of tools. We built the Dropblock Add-on for Google Workspace and BigQuery as a perfect implementation of this capability. And it is further enhanced by the ability to redact (partially encrypt) data from within a document into a separate block. (We'll see more of that later in the documentation. But you can do this from any tool you wish via the API.)

Notice the extra key in this JSON object: 'showProtected'. You must declare 'true', or the default value false will be used.

Note: When set 'true', the full file name will be returned in the result set. When set 'false', the file name is redacted. However, you can see that one exists. [case sensitive, default is false]

This API allows an authorized user to view the files within a folder structure of a block they have the required permissions to inspect:

Parameters

- **show_protected** – "true" or "false" (default is "false") sets if the search will include redacted content.
- **folder_name** (*str*) – a string to search for.
- **method** (*str*) – (hard coded) HTTP request method (POST)
- **url** (*str*) – (hard coded) URL to the Omniindex API endpoint
- **payload** (*str*) – JSON string containing the unit name, server, block type, user and API key.
- **headers** (*dict*) – (hard coded) Content-Type and Accept headers.
- **response** (*str*) – Response from the API call.

Returns

JSON string containing the files in the block.

Reference to `omniindex.api.OmniIndexClient.getfiles()`.

post_minedata(*key*, *data*)

This POST method will add a block to the chain. This is a very dynamic call, that requires a json object with the data to be sent to the server. As of version 0.1.11, the JSON parser will not allow anything other than a string to be sent to the server. We will be adding INT and FLOAT support in the future.

This object MUST follow the follwing rules: Any object that needs to be encrypted, the key must have the word 'Encrypt' added to it. EG: fileContentsEncrypt.

This will ensure that the SDK encrypts the value in all methods available prior to it being sent to a node.

All things OmniIndex, tend to make incredibly complex things, for which we hold multiple patents, very simple indeed. *minedata* is no exception, indeed it is the poster child for simplicity. To create an OmniIndex Blockchain you simply need to create a master encryption key, declare the unit_name of your choice, a user and their api key / passphrase / password (whatever you want).

Parameters

- **data** (*str*) – a string of JSON which is merged with the credentials payload to form the new block.
- **method** (*str*) – (hard coded) HTTP request method (POST)
- **url** (*str*) – (hard coded) URL to the Omniindex API endpoint
- **payload** (*str*) – JSON string containing the unit name, server, block type, user and API key.
- **headers** (*dict*) – (hard coded) Content-Type and Accept headers.
- **response** (*str*) – Response from the API call.
- **key** (*str*) – the base encryption key to use for the new block.

Returns

JSON string containing new block.

Reference to *omniindex.api.OmniIndexClient.post_minedata()*.

run_analytic_query(*show_protected*, *query*)

This POST method will run a query on the Blockchain. To use it you are required to know the definition of the blocks that you are querying. If your *where* syntax includes data that has been encrypted for searching you need to use curly braces around your search string. EG: SELECT X FROM Y where thissearchable-owners LIKE '%{what am i searching for}%'. The API will then convert this into a searchable ciphered stream.

Unlike standard SQL, there is no need to include the name of the datastore because that is defined by the unitName that we are working with. Similarly, there are no joins in 'runanalyticquery', but you can 'SELECT', 'ORDER', 'LIMIT' and set parameters including 'LIKE' to return the data that you want to query

Note: when returning 'data objects' as opposed to 'file objects', these will be base64 encoded and you will need to handle decoding in your own scripts. This is standard practice for all major data store providers

Parameters

- **show_protected** – "true" or "false" (default is "false") sets if the search will include redacted content.
- **query** (*str*) – a string to search for.
- **method** (*str*) – (hard coded) HTTP request method (POST)

- **url** (*str*) – (hard coded) URL to the Omniindex API endpoint
- **payload** (*str*) – JSON string containing the unit name, server, block type, user and API key.
- **headers** (*dict*) – (hard coded) Content-Type and Accept headers.
- **response** (*str*) – Response from the API call.

Returns

JSON string containing the matches for the query.

Reference to *omniindex.api.OmniIndexClient.run_analytic_query()*.

1.1.3 Module contents

CHAPTER

TWO

INTRODUCTION

Welcome to the *OmniIndex Python API* library documentation! This library provides a simple and easy-to-use Python interface for interacting with the OmniIndex API. With *omniindex*, you can quickly retrieve block schematics and perform various tasks related to the OmniIndex blockchain. With the OmniIndex suite of tools, data is protected with our patented 360° encryption, stored in hybrid-blockchains, and then made usable in your favorite collaboration, productivity, and analytics tools with no risk of exposure. Our patented FHE (Fully Homomorphic Encryption) means search and analytics can happen on the data while it is encrypted, and there is tiered access using two encryption keys with configurable access rights. In other words: **Your data is secure and private at all times**

2.1 Getting Started

To use the library, start by importing the *OmniIndexClient* and *os* class: **assumes you have already set an environmental variable for your API key using your os.** If you have not, you can set it using the following command:

For Linux and Mac OS

```
export OMNIINDEX_API_KEY=your_api_key
```

For Windows

```
setx OMNIINDEX_API_KEY your_api_key
```

Usage

```
import os
from omniindex import OmniIndexClient
```

Next, create an instance of the *OmniIndexClient* class with your API credentials and desired parameters: We strongly recommend you *never* use typed strings for your API credentials. Instead, use environment variables to store your API credentials and retrieve them using the *os* class. Refer to your OmniIndex API credentials for the following parameters which are required to create an instance of the *OmniIndexClient* class:

```
your_api_key = os.environ.get("OMNIINDEX_API_KEY")

client = OmniIndexClient(
    server="https://[your node server address]", # the OmniIndex API blockchain node
    api_key="your_api_key", # your OmniIndex API key
    unit_name="your_unit_name", # your unit name
    block_type="your_block_type", # your block type
    user="your_user" # your user
)
```

For more information on the available classes, methods, and their usage, refer For more information on the client object and elements refer to the OmniIndex Documentation..

CHAPTER

THREE

SAMPLE CODE

all these examples use the demonstration API key, which is limited to 100 requests per day. If you want to use the API for a project, please contact us at developer help to get a production API key.

the demonstration uses the Enron email dataset, which is available at https://www.cs.cmu.edu/~enron/ which has been preprocessed and indexed by OmniIndex. The dataset is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

3.1 Set up the OmniIndex API client

- before we dive into the code, firstly, set up your python virtual environment and install the *omniindex* package:

```
python3 -m venv venv
source venv/bin/activate
pip install --upgrade omniindex
```

- to your environment variables, add the api_key you received from OmniIndex.

```
export OMNIINDEX_API_KEY=your_api_key
```

- now you can start coding!

```
import omniindex as oi
import os
import json

api_key = os.environ['OMNIINDEX_API_KEY']
client = oi.OmniIndexClient(server="https://node1.omniindex.xyz/node",unit_name=
↪"enronemail",api_key=api_key, block_type="Owner", user="enronemail")
```

We can test this by fetching the block schema:

3.2 get_block_schematic

Now you can use the *get_block_schematic* method to fetch block schematics from the OmniIndex API:

```
block_schematic = client.get_block_schematic()
data = json.loads(block_schematic)
print(data)
```

You will now have a JSON output of the block schematic:

```
{'0': {'column_name': 'bodyowners', 'data_type': 'text'}, '1': {'column_name':
↪'bodysearchableowners', 'data_type': 'text'}, '2': {'column_name':
↪'contentsearchableowners', 'data_type': 'text'}, '3': {'column_name': 'context', 'data_
↪type': 'text'}, '4': {'column_name': 'context2', 'data_type': 'text'}, '5': {'column_
↪name': 'folder', 'data_type': 'text'}, '6': {'column_name': 'fromowners', 'data_type':
↪'text'}, '7': {'column_name': 'fromsearchableowners', 'data_type': 'text'}, '8': {
↪'column_name': 'hash', 'data_type': 'character varying'}, '9': {'column_name':
↪'message_id', 'data_type': 'text'}, '10': {'column_name': 'oidxid', 'data_type':
↪'integer'}, '11': {'column_name': 'prevhash', 'data_type': 'character varying'}, '12':
↪{'column_name': 'priorhash', 'data_type': 'text'}, '13': {'column_name': 'recieveddate
↪', 'data_type': 'timestamp without time zone'}, '14': {'column_name': 'sentiment',
↪'data_type': 'text'}, '15': {'column_name': 'sentiment2', 'data_type': 'text'}, '16': {
↪'column_name': 'subject', 'data_type': 'text'}, '17': {'column_name': 'toowners',
↪'data_type': 'text'}, '18': {'column_name': 'tosearchableowners', 'data_type': 'text'}}
```

3.3 get_folders

The *get_folders* method returns a list of folders in the dataset. You can use the *showRedacted* parameter to return the redacted data or the full data.

First, let's get a list of folders with the data redacted: (The use case for this is check that folders are present in the dataset, without revealing the folder names)

```
folders = client.get_folders("false")
data = json.loads(folders)
print(data)
```

You will now have a JSON output of the folders:

```
{'results': [{'directory': 'Data has been redacted.'}, {'directory': 'Data has been␣
↪redacted.'}, {'directory': 'Data has been redacted.'}, {'directory': 'Data has been␣
↪redacted.'}, {'directory': 'Data has been redacted.'}]}
```

Now let's get a list of folders with the full data:

```
folders = client.get_folders("true")
data = json.loads(folders)
print(data)
```

You will now have a JSON output of the folders:

```
{'results': [{'directory': '/data/user/0/com.example.dropblock/cache'}, {'directory':
↪'Dropblock/2023-03-07'}, {'directory': 'Dropblock/2023-03-02'}, {'directory':
↪'Dropblock/2023-03-01'}, {'directory': 'Dropblock'}]}Tests
```

3.4 get_searchchain

The *get_searchchain* method returns a JSON block for a search phrase within for a given block. You can use the *showRedacted* parameter to return the redacted data or the full data. There is one other parameter you can pass with this method, *block_id*, which is the block id you want to search within. If you don't pass this parameter, the method will search within the latest block. which is *fulltext* or *files* - this means that if you only want to search the encrypted file content, you can do that with the *files* parameter.

Note:

The get_searchchain method is a perfect demonstration of how the OmniIndex Fully Homomorphic Encryption works. The data content is never decrpted, and the search is performed on the encrypted data. The search results are returned in encrypted form, and the data is never decrypted. Also note the two machine learning (Narrow AI) fields that are derived from the encrypted data:

- sentiment
- context

Let's get stuck into a query on the blockchain:

```
searchresult = client.get_searchchain("true", "working with google workspace", "fulltext
↪")
data = json.loads(searchresult)
print(data)
```

You will now have a JSON output of the search results, which in this case will be every document that has the search phrase 'working with google workspace' in the document content, or title, showing the context and sentiment of that document:

We could go on to pull a specific content block, or version of that content using the *get_files* method, and will explore that next.

```
{ "results" : [{"author" : "matthew@omniindex.io ,sibain@omniindex.io ,james@omniindex.
↪io ","context" : "investment","datetime" : "2022-12-02 19:49:00","directory" : "/
↪OmniIndex/demonstration","filecreateddate" : "2022-12-02 11:27:00","fileextension" :
↪"application/vnd.google-apps.document","filemodifieddate" : "2022-12-02 19:49:00",
↪"filename" : "OmniIndex Distributed Data Platform with Google Workspace","file" :
↪"JVBERi0xLjQKJdPr6eEKMSAwIG9iago8PC9UaXRsZSAoT21uaUluZGV4IERpc3RyaWJ1dGVkIERhdGEgUGxhdGZvcm0gd2l0aCBH
↪/H57d679+5dvevaQBGFLW5xXV0riHTQdfda1oKC0tKxixXFRk9mQu/SSZkSQugQWiDUhA5JICEJgZDe+2Tm8z2/
↪OUHvXZIJTWBSXs/
↪H+WMX4sznzJxzjOd9Pp+vpgEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪","filesize" : "749017","fullpath" : "/OmniIndex/demonstration/OmniIndex Distributed␣
↪Data Platform with Google Workspace","hash" :
↪"FB25A50F51EF1B009ED4D2BBB2466A4427A1C3A3F3F09358677BE797126D2CC3","oidxid" : "7",
↪"priorhash" : "AAEB383EC54C3AE02A982DEBC03934869BAC737863369BE836F4F7EAFE48D495",
↪"sentiment" : "Happy"}]}
```

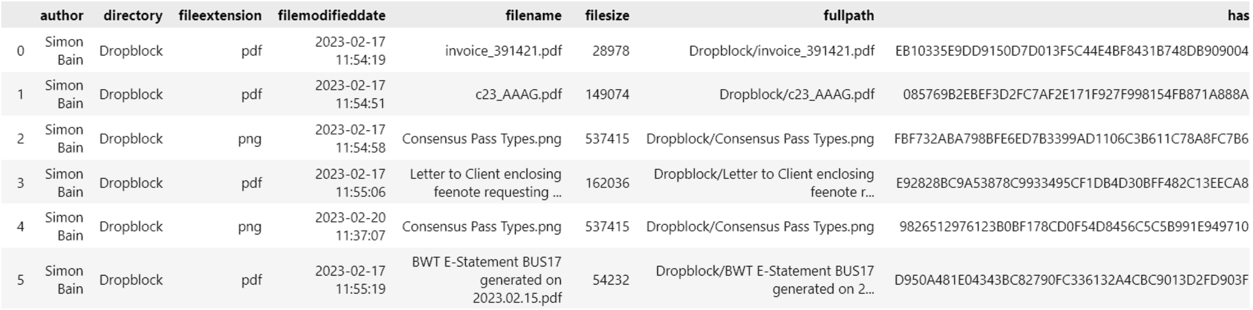
(For convenience, we have excluded the majority of the Encryption hash)

3.5 get_files

In order to list files in a folder construct on the block chanin, we use the following (note that in this example we are using the pandas dataframe library)

```
fileresult = client.getfiles("true", "dropblock")
data = json.loads(fileresult)
data_df = pd.DataFrame(data['results'])
print(data_df)
```

This gives you a list of files in the folder, and the encrypted content of the file:



	author	directory	fileextension	filemodifieddate	filename	filesize	fullpath	has
0	Simon Bain	Dropblock	pdf	2023-02-17 11:54:19	invoice_391421.pdf	28978	Dropblock/invoice_391421.pdf	EB10335E9DD9150D7D013F5C44E4BF8431B748DB909004
1	Simon Bain	Dropblock	pdf	2023-02-17 11:54:51	c23_AAAG.pdf	149074	Dropblock/c23_AAAG.pdf	085769B2EBEF3D2FC7AF2E171F927F998154FB871A888A
2	Simon Bain	Dropblock	png	2023-02-17 11:54:58	Consensus Pass Types.png	537415	Dropblock/Consensus Pass Types.png	FBF732ABA798BFE6ED7B3399AD1106C3B611C78A8FC7B6
3	Simon Bain	Dropblock	pdf	2023-02-17 11:55:06	Letter to Client enclosing feenote requesting ...	162036	Dropblock/Letter to Client enclosing feenote r...	E92828BC9A53878C9933495CF1DB4D30BFF482C13EECA8
4	Simon Bain	Dropblock	png	2023-02-20 11:37:07	Consensus Pass Types.png	537415	Dropblock/Consensus Pass Types.png	9826512976123B0BF178CD0F54D8456C5C5B991E949710
5	Simon Bain	Dropblock	pdf	2023-02-17 11:55:19	BWT E-Statement BUS17 generated on 2023.02.15.pdf	54232	Dropblock/BWT E-Statement BUS17 generated on 2...	D950A481E04343BC82790FC336132A4CBC9013D2FD903F

You can see from the structure of the returned dataset that there is a 'context' field included, which is automatically calculated by the OmniIndex engine against a narrow machine learning model or ontology. This is a great way to quickly identify the context of a document, and can be used to filter search results.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 9 columns):
#   Column            Non-Null Count  Dtype
---  ------            --------------  -----
0   author            20 non-null     object
1   directory         20 non-null     object
2   fileextension     20 non-null     object
3   filemodifieddate  20 non-null     object
4   filename          20 non-null     object
5   filesize          20 non-null     object
6   fullpath          20 non-null     object
7   hash              20 non-null     object
8   context           2 non-null      object
dtypes: object(9)
```

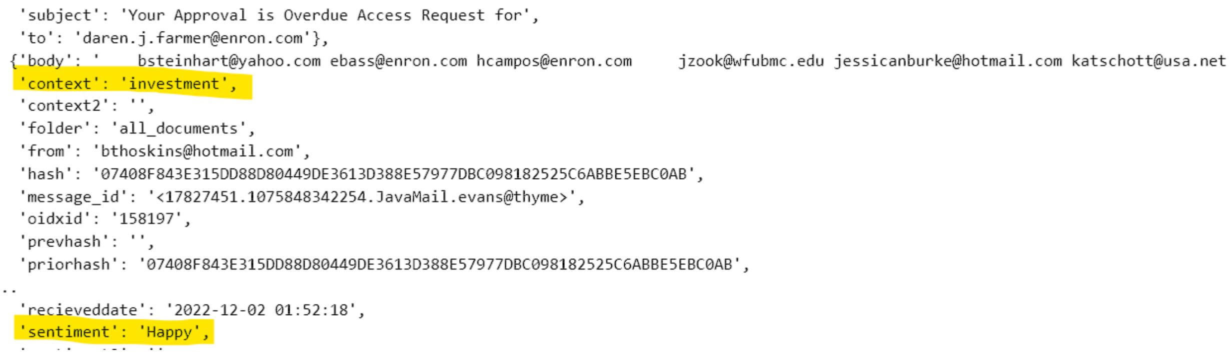
3.6 run_analytic_query

This POST method will run a query on the Blockchain. To use it you are required to know the definition of the blocks that you are querying. If your where syntax includes data that has been encrypted for searching you need to use curly braces around your search string. EG:

```
SELECT X FROM Y where thissearchableowners LIKE '%{what am i searching for}%'.
```

The API will then convert this into a searchable ciphered stream. Running this query is akin to a SQL or OData query on any dataset, except this one is protected by OmniIndex's patented FHE. The only thing to watch out for is that unlike standard SQL, there is no need to include the name of the datastore because that is defined by the unitName that we are working with. Similarly, there are no joins in ‘runanalyticquery’, but you can ‘SELECT’, ‘ORDER’, ‘LIMIT’ and set parameters including ‘LIKE’ to return the data that you want to query. Note that when returning ‘data objects’ as opposed to ‘file objects’, these will be base64 encoded and you will need to handle decoding in your own scripts. This is standard practice for all major data store providers. Notice that the select statement is ‘SELECT * FROM WHERE […]’

```
query = "SELECT * FROM WHERE contentsearchableowners LIKE '%{enron}%' LIMIT 10 "
queryresult = client.run_analytic_query("true", query)
data = json.loads(queryresult)
data_df = pd.DataFrame(data['results'])
print(data_df)
```



```
  'subject': 'Your Approval is Overdue Access Request for',
  'to': 'daren.j.farmer@enron.com'},
 {'body': '    bsteinhart@yahoo.com ebass@enron.com hcampos@enron.com     jzook@wfubmc.edu jessicanburke@hotmail.com katschott@usa.net
  'context': 'investment',
  'context2': '',
  'folder': 'all_documents',
  'from': 'bthoskins@hotmail.com',
  'hash': '07408F843E315DD88D80449DE3613D388E57977DBC098182525C6ABBE5EBC0AB',
  'message_id': '<17827451.1075848342254.JavaMail.evans@thyme>',
  'oidxid': '158197',
  'prevhash': '',
  'priorhash': '07408F843E315DD88D80449DE3613D388E57977DBC098182525C6ABBE5EBC0AB',
..
  'recieveddate': '2022-12-02 01:52:18',
  'sentiment': 'Happy',
```

3.7 post_minedata

This POST method will add a new block on the Blockchain. To use it you are required to know the definition of the blocks that you are adding to. Blockchain by definition are immutable, so you cannot update a block, but you can add a new block with updated data. It is also necessary to pass the master encryption key when creating a new block as always, we would recommend that you never type this in your code, but use environment variables or a config file to store this information. Having followed these strict rules, you need only pass 2 parameters to the API which are the master encryption key and the data that you want to add to the blockchain. The data is passed as a JSON string, and the API will return only status code 200. It will do this even if you are unsuccesful in adding to the blockchain, we are very careful not to return any information that could be used to identify the data that is on your blockchain, even if the appaling event of your encryption key being known to a bad actor.

Note:

> 1. Create a master encryption key. This is a key that only you will ever know and will be used to encrypt all data in the OmniIndex Blockchain. This key is the heart of the OmniIndex Blockchain and what makes us so unique. So long as this key is safe, your data can never be compromised. For this reason, you must keep this key safe and never share it with anyone. If you lose this key, you will lose all of your data. There are many enterprise grade encryption tools available to help you create and store your master key. 2. Set up your omniindex client with the unit_name of your choice (This should map to the business unit or use case for this blockchain. It is the name that will be used to identify your Blockchain. It is also the name that will be used to identify your Blockchain when you are querying it or running data analytics against it), user/password pairings. 3. Create a JSON object with the data you want to store in the OmniIndex Blockchain. This object must follow the rules outlined above.

Think hard about the data you want to store in the OmniIndex Blockchain. You can store anything you want, unstructured blobs of ‘stuff’ or structured filesystems. The choice is yours. The only thing you need to remember is that there is no going back. The schema of the blockchain is set at the time of creation and cannot be changed. (If it could, it would not be the immutable ledger or system of record that is a key feature of OmniIndex).

Warning: If you are using the OmniIndex API to add data to the blockchain, you are responsible for ensuring that the data you are adding is compliant with the GDPR and other data protection laws. You are also responsible for ensuring that you have the right to add the data to the blockchain.

Note: as of version 0.1.11, the JSON parser will only accept strings, so you will need to convert any numbers to strings before adding to the blockchain.

```
import os
NODE = os.environ.get('OMNIINDEX_NODE')
USER_KEY = os.environ.get('OMNIINDEX_USER_KEY')
UNIT_NAME = os.environ.get('OMNIINDEX_UNIT_NAME')
USER = os.environ.get('OMNIINDEX_USER')

client = OmniIndexClient(NODE, USER_KEY, UNIT_NAME, 'Owner', USER)
# even though the data is JSON, it needs to be passed as a string, see the unix␣
↪timestamp and filesize examples below
data = '{"blahEncrypt": "blah1", "contentsearchable": "Some fabulous content", "dateAdded
↪": "2021-01-01", "dateModified": "190266420000", "fileExtension": "txt", "fileSize":
↪"100", "filename": "test.txt"}'
result = client.post_minedata(MASTER_KEY, data)
print(result)
```

3.8 Datasets, dataframes and pandas

If you want to use the popular Pandas dataframe library, the OmniIndex API returns JSON string, which needs to be loaded as JSON using the JSON library.

```
import pandas as pd
import json

searchresult = client.get_searchchain("true", "working with google workspace", "fulltext
↪")
data = json.loads(searchresult)
df = pd.DataFrame(data['results'])
print(df)
```

CHAPTER

FOUR

TESTS

There is a set of tests that can be run against the API endpoints. These are not exhaustive, but they do cover the main functionality of the API and are included in the file tools/omni_api_test.py and utilise the *pytest* package which should be installed in your virtual environment.

4.1 API endpoint tests

- to run the tests, first install the *pytest* package:

```
pip install pytest
```

Note:

you will need to have set the following environment variables:

- 'OI_API_TEST_NODE' the node you want to test
- 'OI_API_TEST_USER_KEY' the user key you want to test
- 'OI_API_TEST_DEMO_KEY' another user key you want to test
- 'OI_API_TEST_USER_DEMO' the user name of the user key you want to test
- 'OI_API_TEST_UNIT_DEMO' the unit name of the user key you want to test

- then run the tests:

```
def test_ssl_api_endpoint():
 # import the requests library
 import requests

 endpoint = "https://api.omniindex.xyz/api_v1" # set the api endpoint
 response = requests.get(endpoint, verify=True) # make a request to the endpoint

 assert response.status_code == 200 # assert that the response is successful
```

4.2 Redaction test

When you run a call with showRedacted=True, the API will return the redacted data. To make sure that the redaction is working correctly, we have a test that checks the redaction has happened when set to ‘false’

```
USER_DEMO_KEY = os.getenv('OI_API_TEST_DEMO_KEY')
UNIT_DEMO = os.getenv('OI_API_TEST_UNIT_DEMO')
USER_DEMO = os.getenv('OI_API_TEST_USER_DEMO')

def test_get_folders_false_returns_json_string():
 """Test that the get_block_schematic() method returns a valid JSON string when␣
↪showProtected is set to false"""
 client = OmniIndexClient(NODE, USER_DEMO_KEY, UNIT_DEMO, 'Owner', USER_DEMO)    # user␣
↪your own api key etc here
 json_string = client.get_folders("false")
 assert type(json_string) == str
 assert json.loads(json_string) is not None
 assert json.loads(json_string) != {}
 json_data = json.loads(json_string)
 assert "Data has been redacted" in json.dumps(json_data) # check that the data has been␣
↪redacted
```

4.3 Count of chains in the blockchain test

This test is super useful to check how many chains there are in the blockchain, most often used when you want to know the number of chains with a particular data set within (although this test just returns the total, you can create your own SQL command for more complex queries)

```
def test_get_blockchain_count():
 """Test that the get_blockchain_count() method returns a valid JSON string"""
 client = OmniIndexClient(NODE, USER_DEMO_KEY, UNIT_DEMO, 'Owner', USER_DEMO)    # user␣
↪your own api key etc here
queryresult = client.run_analytic_query("false", "SELECT COUNT (*) FROM ")
data = json.loads(queryresult)
assert data['results'][0]['count'] >= 1
```

CHAPTER

FIVE

CREATING AND MAINTAINING ENCRYPTION KEYS

This is by no means an exhaustive review of creating and maintaining encryption keys. It is a quick overview of the process. For more information, please see the following resources:

- Google cloud https://cloud.google.com/storage/docs/encryption/customer-managed-keys
- AWS https://docs.aws.amazon.com/mgn/latest/ug/ebs-encryption-kms.html
- Azure https://learn.microsoft.com/en-us/azure/storage/common/customer-managed-keys-configure-existing-account?tabs=azure-portal
- Oracle https://docs.oracle.com/en-us/iaas/Content/KeyManagement/Concepts/keyoverview.htm
- National Cyber Security Centre https://www.ncsc.gov.uk/collection/top-tips-for-staying-secure-online/password-managers

5.1 Creating a key

This sample would use the gcloud cli:

```
gcloud kms keyrings create my-keyring --location global
gcloud kms keys create my-key --location global --keyring my-keyring --purpose encryption
```

This sample would create a key using openssl (Linux, MacOS etc)

```
openssl genrsa -out private.key 2048
```

This is the code to use powershell on Windows:

```
$key = New-Object System.Security.Cryptography.RSACryptoServiceProvider 2048
$key.ExportParameters($true) | Export-Clixml -Path private.key
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

o

omniindex, 5

omniindex.api, 1

INDEX

G

get_block_schematic() (*omniindex.api.OmniIndexClient method*), 1

get_folders() (*omniindex.api.OmniIndexClient method*), 2

get_searchchain() (*omniindex.api.OmniIndexClient method*), 2

getfiles() (*omniindex.api.OmniIndexClient method*), 3

M

module
- omniindex, 5
- omniindex.api, 1

O

omniindex
- module, 5

omniindex.api
- module, 1

OmniIndexClient (*class in omniindex.api*), 1

P

post_minedata() (*omniindex.api.OmniIndexClient method*), 3

R

run_analytic_query() (*omniindex.api.OmniIndexClient method*), 4