# JLOD Library

## About

JLOD stands for JSON Local Document Database, it is a serverless in-process library that implements a self-contained document-oriented database use for storing small application data within an application without an internet connection. The database is like MongoDB, it uses Collection, Documents, and JSON objects. The JLOD is a version of SQLite in document format, the JLOD collections can be exported to Remote MongoDB collection as well as remote MongoDB collection can also be imported to the JLOD database.

JLOD is an embedded document-oriented database library. Unlike MongoDB. JLOD does not have a separate server process. JLOD reads and writes the data to ordinary disk files. The complete JLOD database along with the collections and the documents are contained in a disk file. The folder is a database in JLOD while the file is a collection, and each line in the file is a document.

## Audience

This library will be quite helpful for all those developers who want to develop serverless web applications using Python. It is meant for programmers with a stronghold on MongoDB concepts.

## Prerequisite

The library assumes that the readers have adequate exposure to MongoDB or any document-oriented database concepts. If you have worked on MongoDB, then it will help you further to grasp the concepts of the library quickly.

## Copyright & Disclaimer

1. Any user is permitted to reuse, distribute this library
2. Any user can contribute to this library
3. No user that is permitted to retain, copy, or republish this Library.

# Table of Contents



# Self Contained

JLOD database stored data in a folder within an application,  once the application is hosted to any server or build by any compiler, the database will remain along with the application. And data can be stored and retrieved without an internet connection.

**Folder** is a database

**File** is a Collection

**Object** is a Document

# Performance

Since the JLOD database stored data in a local folder within an application, the performance is extremely awesome. The application will not need to initiate an internet connection to access the stored data.

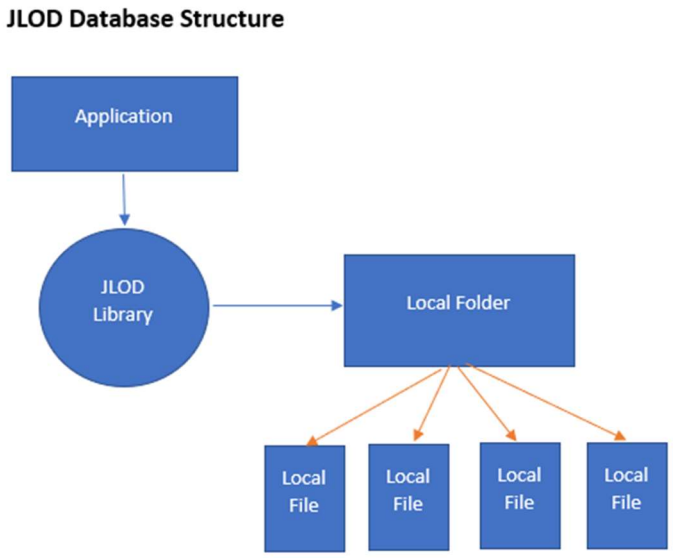
# Serverless

Since the JLOD database stored data in a local folder within an application, there is no need for a server. The database stored where ever the application is hosted.

# Structure

JLOD library creates a folder as a Database and creates a file as a collection then adds a JSON array of objects to the files as a group of documents.

### JLOD Database Structure




# Installation

Install JLOD for python.

Open the command line and type in [ **pip install jlod - - save** ]

The above command will install the JLOD library to your project, once the installation is done, you can import it to your project and begin to use it.

To confirm JLOD installed perfectly, run this command [ **jlod --version** ] if it is installed the version of the JLOD installed will appear like **jlod Version 0.0.1**.

# Create Database

```
from jlod import database

dbi = database.connect('example').instance

print(dbi)
```

Database name: **example**

Base on the above JLOD script, [**example**] is the database name, therefore folder will be created with the same name or the reference will be returned if the folder already exists. The instance of the database will now be returned and the stored in **dbi** variable.

The reference of the database created will look like below in the **dbi** variable.

**<jlod.Database object at 0x00CC01A8>**

With the **dbi** variable, you can manipulate the database.

# Create Collection

```
from jlod import database

dbi = database.connect('example').instance
client = dbi.collection('users')
print(client)
```

Collection name: **users**

Base on the above JLOD script, [**users**] is the collection name, therefore a file will be created with the same name or referenced of the file will be returned if the file already exists. The instance of the collection will now be returned and stored in the **client** variable.

The reference of the collection created will look like below in the **client** variable.

**<jlod.Database object at 0x038B01C0>**

With the **client** variable, you can manipulate the collection.

# Add Documents

The JLOD support two different ways to add a document to a collection

1. Add a single document
2. Add multiple documents

- Add a single document

```
 added = client.add({
   "name": "Rabie",
   "gender": "Male",
   "age": 24,
   "city": "Los Angeles",
   "country": "America"
})
if added:
   print("Document Added")
else:
   print("Failed to add Document")
```

Output: **Document Added**

- Add a multiple documents

```
added = client.addMany([
   {"name": "Jhon", "gender": "Male", "age": 24, "city": "New Delhi", "country": "India"},
   {"name": "Zee", "gender": "Female", "age": 20, "city": "Lagos", "country": "Nigeria"},
   {"name": "Frank", "gender": "Male", "age": 27, "city": "Toronto", "country": "Canada"}
])
if added:
   print("Documents Added")
else:
   print("Failed to add Documents")
```

Output: **Documents Added**

# Drop Collection

```
res = client.drop
print(res)
```

Output: **True**

Above JLOD script will drop [**users**] collection and return True as a response. If the collection doesn't exist will also return True.

# Collection Size

```
size = client.size
print(size)
```

Output: **0**

Above will return the size of [**users**] collection

# Get Documents

JLOD support 3 different ways to display all documents in the collection

See examples below:

```
result = client.documents

result = client.get('*')

result = client.find()

print(result)
```

Output:


[
{'name': 'Rabie', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'T7ra8IIRqp1qsDic4nQvoRJjAx4Pbq'},
{'name': 'Jhon', 'gender': 'Male', 'age': 24, 'city': 'New Delhi', 'country': 'India', 'id': 'd6PraKHTRdljH13EQLNUDRQLvj556W'},
{'name': 'Zee', 'gender': 'Female', 'age': 20, 'city': 'Lagos', 'country': 'Nigeria', 'id': 'lEtmtjZwZLZHUbFZFjbIAAkwzEXlNj'},
{'name': 'Frank', 'gender': 'Male', 'age': 27, 'city': 'Toronto', 'country': 'Canada', 'id': 'O7MRcbdlZGzfy6VD8ghbJoTpy2qXPi'}
]


```
result = client.get(['name'])

print(result)
```

## Output:

```
[
  {'name': 'Rabie'},
  {'name': 'Jhon'},
  {'name': 'Zee'},
  {'name': 'Frank'}
]
```

Above will get only name  from the  all documents in the collection

```
result = client.get(['name', 'age'])

print(result)
```

## Output:

```
[
  {'name': 'Rabie', 'age': 24},
  {'name': 'Jhon', 'age': 24},
  {'name': 'Zee', 'age': 20},
  {'name': 'Frank', 'age': 27}
]
```

Above will get name and age from the  all documents in the collection

```
result = client.get(['name', 'age'], {
        "name": "Jhon",
        "age": 24
})

print(result)
```

## Output:

```
[
 {'name': 'Rabie', 'age': 24},
 {'name': 'Jhon', 'age': 24}
]
```

Above will get name and age from any document that contains name with value “**Jhon**” or age with value **24** in the collection

```
result = client.getOne(['name', 'age'], {
          "name": "Jhon",
          "age": 24
})

print(result)
```

Output:

```
[
  {'name': 'Rabie', 'age': 24}
]
```

Above will get name and age from any document that contains a name with value "**Jhon**" and age with value **24** in the collection and return one document of the result set.

# Query Documents

There are two types of logical Operators in JLOAD:

1. AND
2. OR

AND => returns document that matches all given conditions

OR=> returns documents that match any of the given conditions

```
result = client.find({
   "$and": {"name": {"$eq": "Rabie"}}
})

print(result)
```

Output:

```
[
  {'name': 'Rabie', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'y3wRXCDwx0cm2UHcB7LX2b1P8XymsC'}
]
```

Using the AND operator. Above will find any document that has key "**name**" and value equal to "**Rabie**"

```
result = client.find({
  "$or": [{"name": {"$eq": "Rabie"}}, {"age": {"$eq": 20}}]
 })

print(result)
```

Output:

[
{'name': 'Rabie', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'y3wRXCDwx0cm2UHcB7LX2b1P8XymsC'},
{'name': 'Zee', 'gender': 'Female', 'age': 20, 'city': 'Lagos', 'country': 'Nigeria', 'id': 'fXjqWydHS6GEibY3JF84L0l9oEvNF0'}
]

Using the OR operator. Above will get any document that has a key name as “Rabie” or age as “**23**” in the collection.

```
result = client.findOne()

print(result)
```

Output:

[
{'name': 'Rabie', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'T7ra8IIRqp1qsDic4nQvoRJjAx4Pbq'}
]

Above will return first one document  from the entire collection.

```
result = client.findOne({
  "$and": {"age": {"$gt": 24}}
})

print(result)
```

Output:

[
{'name': 'Frank', 'gender': 'Male', 'age': 27, 'city': 'Toronto', 'country': 'Canada', 'id': 'opSgQ8bMLirWfuPsFrNFtuQPJXUjbs'}
]

Above will return the first one document that has key “age” with value greater than 24.

## Sorting and Distinct

1. Sorting is used in sorting data in certain format **Accending** or **Descending**.

see examples below

```
result = client.sort({"age": -1})

print(result)
```

## Output:


[
{'name': 'Mary', 'gender': 'Female', 'age': 20, 'city': 'Lagos', 'country': 'Nigeria', 'id': 'fXjqWydHS6GEibY3JF84L0I9oEvNF0'},
{'name': 'Mary', 'gender': 'Male', 'age': **20**, 'city': 'Toronto', 'country': 'Canada', 'id': 'opSgQ8bMLirWfuPsFrNFtuQPJXUjbs'},
{'name': 'Zee', 'gender': 'Female', 'age': 20, 'city': 'Lagos', 'country': 'Nigeria', 'id': 'PrmKnIWEbUtvyudWunPjSOfINmjX98'},
{'name': 'Jhon', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'y3wRXCDwx0cm2UHcB7LX2b1P8XymsC'},
{'name': 'Jhon', 'gender': 'Male', 'age': 24, 'city': 'New Delhi', 'country': 'India', 'id': 'GV5fUn2SN8eX2eGn3nti6VfH6wVEEJ'},
{'name': 'Frank', 'gender': 'Male', 'age': 27, 'city': 'Toronto', 'country': 'Canada', 'id': 'Xx3M1ub4F4lGwOWpQi1GicSweElwgm'}
]


Above will sort all documents base on key "**age**" Deceinding order.

```
result = client.sort({"name": 1},{
  "$and": {"age": {"$eq": 20}}
})

print(result)
```

## Output:


[
{'name': 'Zee', 'gender': 'Female', 'age': 20, 'city': 'Lagos', 'country': 'Nigeria', 'id': 'PrmKnIWEbUtvyudWunPjSOfINmjX98'},
{'name': 'Mary', 'gender': 'Female', 'age': 20, 'city': 'Lagos', 'country': 'Nigeria', 'id': 'fXjqWydHS6GEibY3JF84L0I9oEvNF0'},
{'name': 'Mary', 'gender': 'Male', 'age': 20, 'city': 'Toronto', 'country': 'Canada', 'id': 'opSgQ8bMLirWfuPsFrNFtuQPJXUjbs'}
]


Above will sort all documents base on key "**name**" Accending order for all documents that has key "**age**" with value equal to 20.

1. Distinct is used to get distinct data from a collection with no duplicate.

```
result = client.distinct()

print(result)
```

Output:

```
[
  {'name': 'Rabie', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'T7ra8IIRqp1qsDic4nQvoRJjAx4Pbq'},
  {'name': 'Jhon', 'gender': 'Male', 'age': 24, 'city': 'New Delhi', 'country': 'India', 'id': 'd6PraKHTRdljH13EQLNUDRQLvj556W'},
  {'name': 'Zee', 'gender': 'Female', 'age': 20, 'city': 'Lagos', 'country': 'Nigeria', 'id': 'lEtmtjZwZLZHUbFZFjbIAAkwzEXlNj'},
  {'name': 'Frank', 'gender': 'Male', 'age': 27, 'city': 'Toronto', 'country': 'Canada', 'id': 'O7MRcbdlZGzfy6VD8ghbJoTpy2qXPi'}
]
```

Above will return the distinct documents

```
result = client.distinct({
 "$and": {"city": {"$eq": "Los Angeles"}}
})

print(result)
```

Output:

```
[
  {'name': 'Rabie', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'T7ra8IIRqp1qsDic4nQvoRJjAx4Pbq'},
]
```

Above will return the distinct documents for any document that has key “city” with value “Los Angels”

# Remove Document

In JLOD remove is used to remove the entire or specific document from a collection,

see examples below.

```
client.remove()
```

```
client.removeAll()
```

Output: **True**

Anyone of the above will remove all documents in the collection

```
result = client.remove({
  "$and": {"name": {"$eq": "Jhon"}}
})

print(result)
```

Output: **True**

Above will remove any document that has key "**name**" with value "**Jhon**" from the collection

```
result = client.remove({
   "$or": {"age": {"$gt": 27}}
})

print(result)
```

Output: **True**

Above will remove any document that has key "**name**" with a value greater than "**27**"

# Update Document

JLOD update is used in updating documents of a specific collection.
see examples below.

```
client.update({
     "name": "Mary",
     "age": 20
})
```

Output: **True**

Above will update the key "**name**" to "**Mary**" and age to "**20**" for all documents in the collection. And if the key age doesn't exist in the collection, it will be created

```
result = client.update( {"name": "Jhon", "age": 24},{
        "$and": {"city": {"$eq": "Los Angeles"}}
})

print(result)
```

Output: **True**

Above will update the name to "**Jhon**" and age to "**24**" on any document that has a city equal to "**Los Angeles**".

# Truncate Document

```
result = client.truncate

print(result)
```

Output: **True**

Above will remove all documents in the collection

# Export Collection

Export is used to export the entire or specific document in a JLOD collection to remote MongoDB see examples below.

```
from jlod import database
import pymongo

dbi = database.connect('example').instance
client = dbi.collection('users')
```

```
host = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = host["example"]
mongoClient = mydb["users"]

result = client.exportTo(mongoClient)

print(result)
```

Output: **True**

The above will export all the documents in JLOD users collection to MongoDB users collections.

```
result = client.exportTo(mongoClient, {
  "$and": {"name": {"$eq": "Zee"}}
})

print(result)
```

Output: **True**

Above will export any document that contains a name with value "**Zee**" from the JLOD users collection to remote MongoDB users collections.

# Import Collection

Import is used for importing entire or specific documents from the remote MongoDB collection to JLOD local database collection. See the examples below.

```
from jlod import database
import pymongo

dbi = database.connect('example').instance
client = dbi.collection('users')

host = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = host["example"]
mongoClient = mydb["users"]
```

```
result = client.importFrom(mongoClient)

print(result)
```

Output:


[
{'name': 'Rabie', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'T7ra8IIRqp1qsDic4nQvoRJjAx4Pbq'},
{'name': 'Jhon', 'gender': 'Male', 'age': 24, 'city': 'New Delhi', 'country': 'India', 'id': 'd6PraKHTRdljH13EQLNUDRQLvj556W'},
{'name': 'Zee', 'gender': 'Female', 'age': 20, 'city': 'Lagos', 'country': 'Nigeria', 'id': 'lEtmtjZwZLZHUbFZFjbIAAkwzEXlNj'},
{'name': 'Frank', 'gender': 'Male', 'age': 27, 'city': 'Toronto', 'country': 'Canada', 'id': 'O7MRcbdlZGzfy6VD8ghbJoTpy2qXPi'}
]


Above will import all the documents from remote MongoDB users collections to JLOD users collection

```
result = client.importFrom(mongoClient, {
   "$and": {"name": {"$eq": "Zee"}}
})

print(result)
```

Output:


[
{'name': 'Rabie', 'gender': 'Male', 'age': 24, 'city': 'Los Angeles', 'country': 'America', 'id': 'T7ra8IIRqp1qsDic4nQvoRJjAx4Pbq'},
]


Above will import any document that contains key "**name**" with value "**Zee**" from the remote MongoDB users collections to JLOD users collection

# End of the library documentation

If you want to contribute or discover any errors or suggestions on the code or logic, please notify us at contact@jload.org.