NoSQL (Distributed / Big Data) Databases

FastAPI can also be integrated with any NoSQL.

Here we'll see an example using **Couchbase**, a document based NoSQL database.

You can adapt it to any other NoSQL database like:

- **MongoDB**
- **Cassandra**
- **CouchDB**
- **ArangoDB**
- **ElasticSearch**, etc.

!!! tip There is an official project generator with **FastAPI** and **Couchbase**, all based on **Docker**, including a frontend and more tools: https://github.com/tiangolo/full-stack-fastapi-couchbase

Import Couchbase components

For now, don't pay attention to the rest, only the imports:

```
{!../../../docs_src/nosql_databases/tutorial001.py!}
```

Define a constant to use as a "document type"

We will use it later as a fixed field `type` in our documents.

This is not required by Couchbase, but is a good practice that will help you afterwards.

```
{!../../../docs_src/nosql_databases/tutorial001.py!}
```

Add a function to get a `Bucket`

In **Couchbase**, a bucket is a set of documents, that can be of different types.

They are generally all related to the same application.

The analogy in the relational database world would be a "database" (a specific database, not the database server).

The analogy in **MongoDB** would be a "collection".

In the code, a `Bucket` represents the main entrypoint of communication with the database.

This utility function will:

- Connect to a **Couchbase** cluster (that might be a single machine).
  - Set defaults for timeouts.
- Authenticate in the cluster.
- Get a `Bucket` instance.
  - Set defaults for timeouts.
- Return it.

```
{!../../../docs_src/nosql_databases/tutorial001.py!}
```

Create Pydantic models

As **Couchbase** "documents" are actually just "JSON objects", we can model them with Pydantic.

`User` model

First, let's create a `User` model:

```
{!../../../docs_src/nosql_databases/tutorial001.py!}
```

We will use this model in our *path operation function*, so, we don't include in it the `hashed_password`.

`UserInDB` model

Now, let's create a `UserInDB` model.

This will have the data that is actually stored in the database.

We don't create it as a subclass of Pydantic's `BaseModel` but as a subclass of our own `User`, because it will have all the attributes in `User` plus a couple more:

```
{!../../../docs_src/nosql_databases/tutorial001.py!}
```

!!! note Notice that we have a `hashed_password` and a `type` field that will be stored in the database.

```
But it is not part of the general `User` model (the one we will return in the *path
operation*).
```

Get the user

Now create a function that will:

- Take a username.
- Generate a document ID from it.
- Get the document with that ID.
- Put the contents of the document in a `UserInDB` model.

By creating a function that is only dedicated to getting your user from a `username` (or any other parameter) independent of your *path operation function*, you can more easily re-use it in multiple parts and also add unit tests for it:

```
{!../../../docs_src/nosql_databases/tutorial001.py!}
```

f-strings

If you are not familiar with the `f"userprofile::{username}"`, it is a Python "f-string".

Any variable that is put inside of `{}` in an f-string will be expanded / injected in the string.

`dict` unpacking

If you are not familiar with the `UserInDB(**result.value)`, it is using `dict` "unpacking".

It will take the `dict` at `result.value`, and take each of its keys and values and pass them as key-values to `UserInDB` as keyword arguments.

So, if the `dict` contains:

```
{
    "username": "johndoe",
    "hashed_password": "some_hash",
}
```

It will be passed to `UserInDB` as:

```
UserInDB(username="johndoe", hashed_password="some_hash")
```

Create your FastAPI code

Create the `FastAPI` app

```
{!../../../docs_src/nosql_databases/tutorial001.py!}
```

Create the *path operation function*

As our code is calling Couchbase and we are not using the experimental Python `await` support, we should declare our function with normal `def` instead of `async def`.

Also, Couchbase recommends not using a single `Bucket` object in multiple "threads", so, we can just get the bucket directly and pass it to our utility functions:

```
{!../../../docs_src/nosql_databases/tutorial001.py!}
```

Recap

You can integrate any third party NoSQL database, just using their standard packages.

The same applies to any other external tool, system or API.