M220P – Mongo DB for Python Developers

**URI;** used to define connection between applications and mongo db instances.

Host name, port, username and password. Additional options allowed too.

For this course, we will be using an srv uri string. <host> is the host name of the srv record. It does not actually point to a database server. It only hosts the srv records. Host name is the address of the file called service record that defines its own dns with a list of host names that we want to resolve to. So even though we are connecting to a cluster of servers. We don’t need to know where each server in the cluster is because the SRV record keeps track of it for us.

Changes in the clusters/servers are recorded in the srv. This means that changes do not have to made on the client side.

mongodb+srv://<username>:<password>@<host>/<database>

Copy requirements.txt file from folder – useful for installing required packages for a given program!

Application Architecture

* Tree function to look at architecture
* README.rst file contains detailed setup instructions.
* API layer is implemented by movies.py and user.py in the mflix/api folder
* Methods to interact with database are all in db.py, mostly use this.
* Tests directory. Tickets are labs that contain a user story and you are to solve it. Once solved, run the respective test and go to the localhost/status and get the code.

MongoClient

* object is a part of the pymongo library
* Can pass uri to the MongoClient object
* Client.stats
* SSL stands for Secure Sockets Layer and, in short, it's the standard technology for keeping an internet connection secure and safeguarding any sensitive data that is being sent between two systems, preventing criminals from reading and modifying any information transferred, including potential personal details. Get for free through MongoClient
* Client.list\_names()
* Client.db or client[‘mflix’]
* In mongo client connection we can specify certain options like connectTimeoutMS which is how long the driver in milliseconds will allow attempt to connect before erroring and also retryWrites = True which is self explanatory – will be shown when you call client.stats
* Collection handles are referenced from the database object. Collection specific operations like querying or updating documents are performed on the collection object.

Aggregations

* There is an aggregation builder in the Compass Desktop application. These can be exported in your desired language of Python, Java, C# or Node.js

Write Concerns

* writeConcern:{w:1} – only requests an acknowledgement that one node applied the write. This is the default write concern. W:1 refers to the number of nodes in the set
* w: majority – client doesn’t get acknowledgment back from the driver until the write has been applied to the majority of nodes in the set. Takes longer but is more durable than w:1. Very useful for ensuring vital writes are majority-committed.
* w:0 – does not request acknowledgement. “fire-and-forget”. Fastest but least durable. E.g. IoT device reporting status every couple of seconds to MongoDB

Update Operations

* update\_one, update\_many
* faker is a package in python to create fake data so that you begin testing your applications
* Update operation return UpdateResult with acknowledged, matched\_count, modified\_count and upserted\_id
* Modified\_count and matched\_count will be 0 in the case of an upsert

Basic Joins

* Use new expressive $lookup
* Build your aggregation in Compass, and then export to language