



ECCAM

BRUSSELS ENGINEERING SCHOOL

MONGODB

OCTOBER 22, 2021

YILMAZ MUSTAFA
ROQUERO PEDRO

Contents

1. Introduction:.....	3
2. Installation:.....	3
a) Python and Kivy:.....	3
b) Mongo DB:.....	4
3. Features of our application:	8
a) Add Customer:.....	8
b) Look for Client and update client:	9
4. Links:.....	12

1. Introduction:

Mongo DB is a document-oriented database that uses the Bson format in order to stock the data in the database. Being document-oriented and not needing to use a specific schema makes this database extremely useful when you want to stock different types of data or when you want to allow freedom to the users in what concerns the information they want to introduce. For this particular reason, we build as Proof of Concept a small application that will allow a company to add new customers into their database or search and update the data for a particular costumer.

To do so, we started building a small form which will ask for some informations about the customer such as the name and first name, the date of birth or the phone number. As MongoDB gives us the possibility to stock different information, the customer does not need to complete all the fields in order to be added into the database. This can be really useful for companies, because as for example, if the client does not have a Belgian National Number and the form is asking for one, with a regular SQL database this would generate some issues comparing to Mongo DB where you will just leave the field empty and add the client to the database.

2. Installation:

a) Python and Kivy:

To build our application we used python and the graphics interface Kivy. First of all you will need to install python from their official website([Download Python | Python.org](https://www.python.org/)). Once Python is installed, we can now proceed to the Kivy installation which is a bit more complex. These are the steps you will need to follow in order to install Kivy using python 3.7.4 (these steps can be different with another python version):

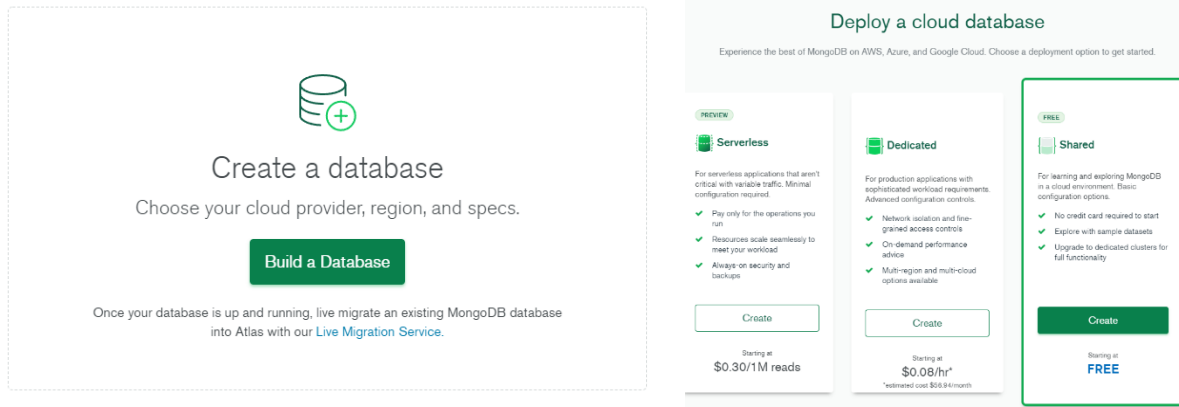
- 1) Open the Command Prompt (CMD)
- 2) Once it is opened, type the following lines in order:
 - `py -m pip install --upgrade pip wheel setuptools virtualenv`
 - `py -m virtualenv kivy_venv`
 - `py -m pip install docutils pygments pypiwin32 kivy_deps.sdl2==0.1.22 kivy_deps.glew==0.1.12`
 - `py -m pip install kivy_deps.gstreamer==0.1.17`
 - `py -m pip install kivy_deps.angle==0.1.9`
 - `py -m pip install kivy==1.11.1`
- 3) Once the Kivy installation completed, you will only need to import Kivy in your python file and you will be ready to start building your app.

Now that we have everything, we need to build our app lets talk about the installation of Mongo DB to use it with our app.

First, we will need to install the python library "pymongo", to do so, we open the Command Prompt and type "`py -m pip install pymongo[srv]`". When installed you will need to import this library on your python file ("`import pymongo`").

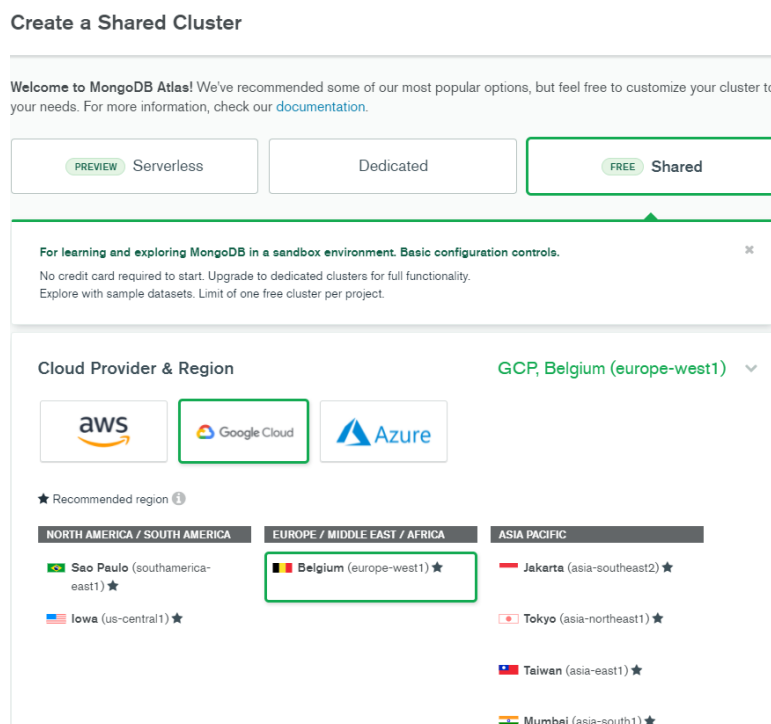
b) Mongo DB:

Then, we will need to go to our Mongo DB website (Mongo DB Atlas) and create the database. This is going to be an example if you are setting up your database from scratch. Once you have created your account you will arrive to this page:



Pressing on the “Build a Database button will open the window shown on the image on the right, from which we are going to select the shared option.

Then, we will have to create a cluster, where we can choose from different types of providers such as Amazon, Google Cloud or Azure, and also different regions:



We chose Google Cloud because it was the one that had as Belgium as a region option but any of them could have worked with our application.

When the cluster has been created, you will arrive to the main page:

The screenshot shows the MongoDB Atlas main page for 'Project 0'. The left sidebar has a 'DEPLOYMENT' section with 'Databases' selected. The main content area is titled 'Database Deployments' and shows a search bar and a table of deployments. The first deployment is 'Cluster0', which is in a 'Ready' state. It has 0 connections and 0.0 B/s of data in/out. The 'Cluster0' row is highlighted with a green background.

From here, you will then need to configure the database access, by adding a new user:

The screenshot shows the 'Database Access' page in the MongoDB Atlas interface. The left sidebar has a 'SECURITY' section with 'Database Access' selected. The main content area is titled 'Database Access' and shows a 'Create a Database User' button. Below the button, there is a description: 'Set up database users, permissions, and authentication credentials in order to connect to your clusters.' and a 'Learn more' link.

Add New Database User

Create a database user to grant an application or user, access to databases and collections in your clusters in this Atlas project. Granular access control can be configured with default privileges or custom roles. You can grant access to an Atlas project or organization using the corresponding [Access Manager](#).

Authentication Method

Password	Certificate	AWS IAM (MongoDB 4.4 and up)
-----------------	--------------------	--

MongoDB uses [SCRAM](#) as its default authentication method.

Password Authentication

Pedro	
pedro	
Autogenerate Secure Password	Copy

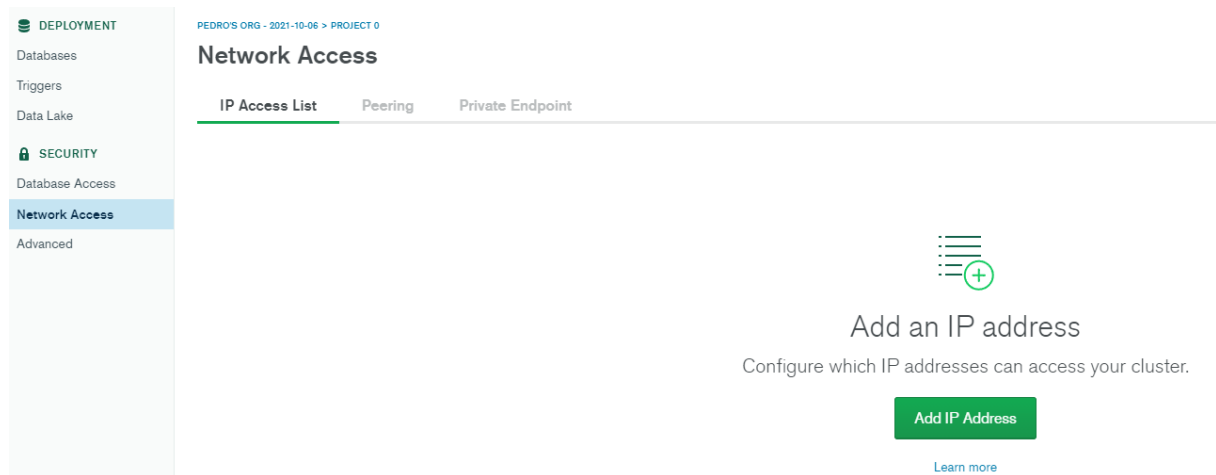
Database User Privileges

Select a [built-in role](#) or [privileges](#) for this user.

Read and write to any database

We have chosen to use password authentication and for the privileges we allow this user to read and write on the database.

Once the user is created, we will now have to allow the IP address from which we will have access to the database. To do so we need to go to network access:



Add IP Access List Entry

Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. [Learn more](#).

ADD CURRENT IP ADDRESS

ALLOW ACCESS FROM ANYWHERE

Access List Entry:

109.136.185.140

Comment:

Optional comment describing this entry

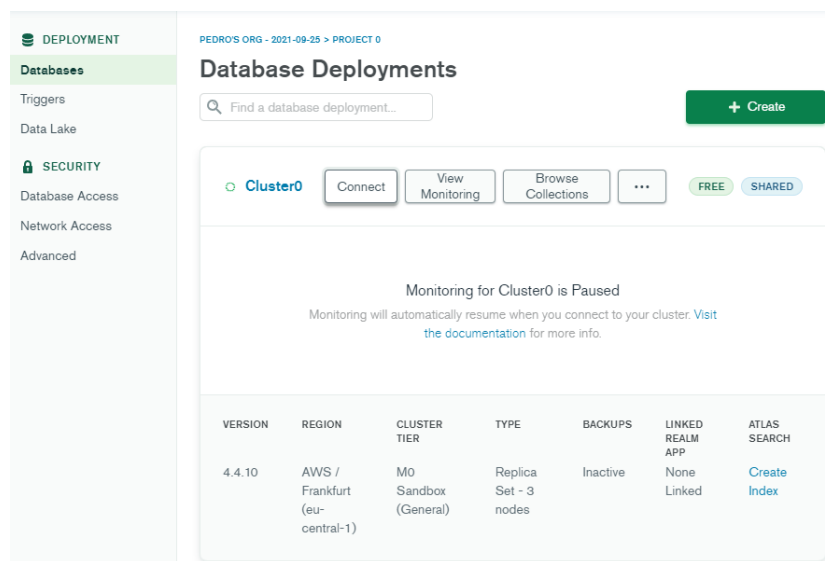
☐ This entry is temporary and will be deleted in 6 hours

Cancel

Confirm

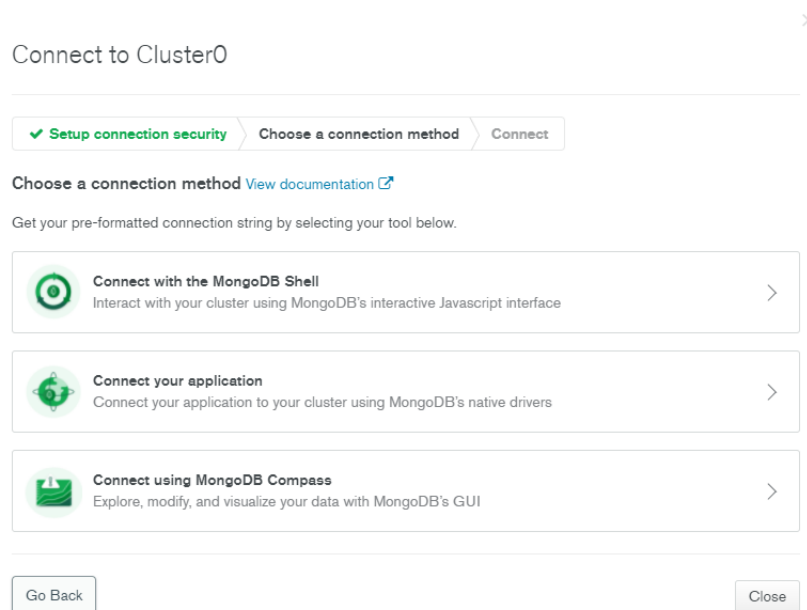
By pressing add current IP address, this will automatically complete the field with your IP address. You can also allow the connections to the database from anywhere, but this could create some security problems, because it will mean that anyone could modify your data if they have your credentials.

Once the IP address has been added we can now focus on getting our application and the database to work together. To do so, from the main page, press on the Connect button:

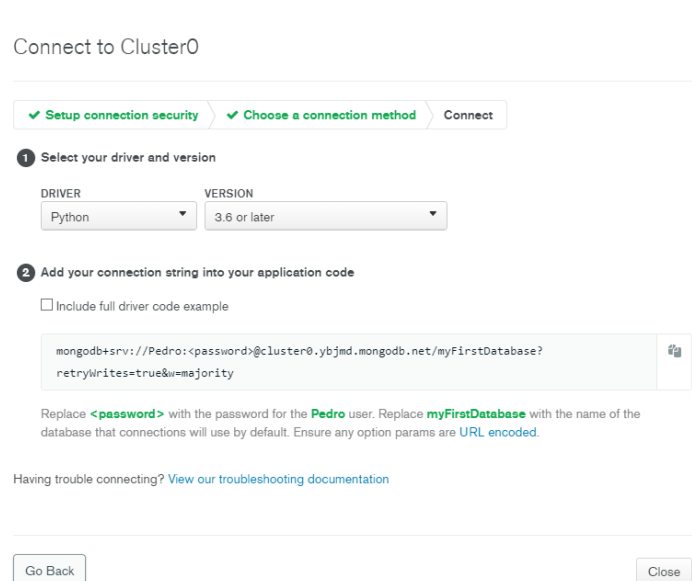


VERSION	REGION	CLUSTER TIER	TYPE	BACKUPS	LINKED REALM APP	ATLAS SEARCH
4.4.10	AWS / Frankfurt (eu-central-1)	M0 Sandbox (General)	Replica Set - 3 nodes	Inactive	None Linked	Create Index

This will open the following window:



From which you will need to press the “Connect your application” button, this will open the following window:



Once in here, you will need to choose the driver you want to use based on the language you are coding your application. In our case, this is going to be Python and the version 3.6 or later as we are using Python 3.7.4. After that, you will have to copy the line that has been automatically generated into your python file and replace “<password>” with the password you used when you created your user, in this example the code to copy on the python file will be:

“mongodb+srv://Pedro:pedro@cluster0.dkwr8.mongodb.net/myFirstDatabase?retryWrites=true&w=majority”

Everything has been installed and you will now be able to start developing your application.

3. Features of our application:

We decided to build an application that will allow a company to write, read and update the database.

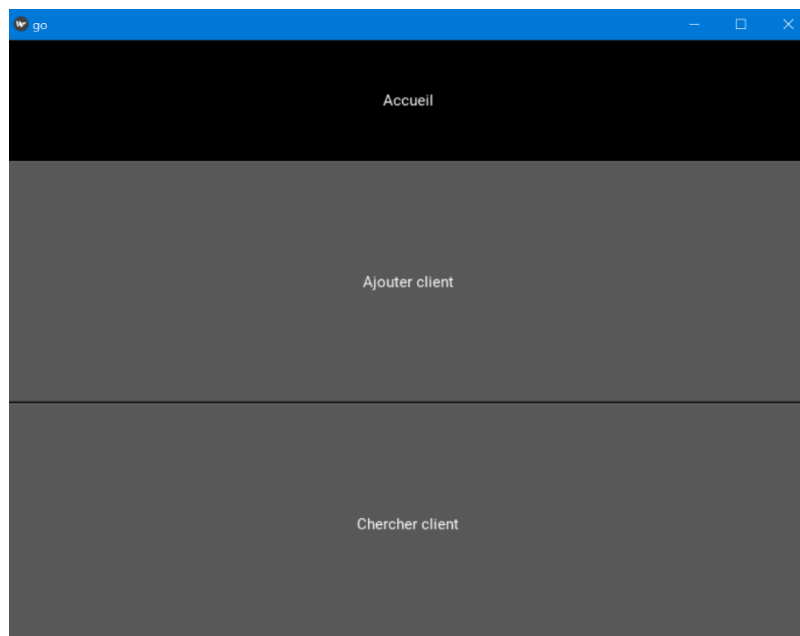
To do so, first we had to connect to the database as explained before. Here is what we have in our code:

```
def dbConn():
    client = pymongo.MongoClient("mongodb+srv://dbUser:Password@mustafaloucluster.euggs.mongodb.net/MustafalouCluster?retryWrites=true&w=majority")
    db = client["Mustafalou"]

    try: db.command("ServerStatus")
    except Exception as e: print(e)
    else: print("you are connected")
    return db, client
```

We added a small try-except clause to our function in order to be able to know if we managed to connect the application to the database.

Then we have created the main menu page using Kivy, this page will only display two buttons, from which we will select if we want to add a new costumer or search for a costumer that is already in the database.



```
def menu(self):
    self.add_widget(self.label)
    self.btn = Button(text = "Ajouter client", on_press = self.ajt)
    self.btn2 = Button(text = "Chercher client", on_press = self.chercher)
    self.add_widget(self.btn)
    self.add_widget(self.btn2)
    self.run = True
    self.setupForms()
```

a) Add Customer:

Then, when we press add a client button, this will open the form we have created and where we are going to introduce the data that we want to stock. To do so we have created two lists, one where we are going to be saving all the labels we want to display and the other one where we are going to add all the inputs that have been inserted.

```
def setupForms(self):
    self.liste_lab = []
    self.liste_input = []
    self.labelNom = Label(text = "Nom")
    self.liste_lab.append(self.labelNom)
    self.inputNom = TextInput()
    self.liste_input.append(self.inputNom)
    self.labelPrenom = Label(text = "Prenom")
    self.liste_lab.append(self.labelPrenom)
    self.inputPrenom = TextInput()
    self.liste_input.append(self.inputPrenom)
```


Once you have completed the fields you wanted to complete you will only need to press the button “Ajouter”, this will automatically redirect you to the main menu page and add to the database the data that has been introduced in a new document.

```
def Confirm(self, btn):
    self.db, self.client = dbConn()
    collection = self.db.Clients
    post = {}
    for elem in range(len(self.liste_lab)):
        if self.liste_input[elem].text != "":
            post[self.liste_lab[elem].text] = self.liste_input[elem].text
    collection.insert_one(post)
    self.client.close()
    print("Fait")
    self.clear_widgets()
    self.menu()
```

This *Confirm* function is the one that sends the data to the database when the user has pressed the “Ajouter” button. What this function does is, first of all establishing a connection by using *dbConn()* function, explained previously. Then we set that the collection where we want to add the data is *Clients* and we create an empty dictionary *post*.

Once this is done, we are going to search in the list of labels, and for each element where we have introduced an input, we will add to the dictionary *post*, the label as Key and the input as Value. When we have run through all the elements of the labels list and added the ones that had to be added, we can send the data to the database. To do so, we only need to use the Mongo DB function *insert_one()* and we put inside the brackets what we want to add to the database, in this case, the dictionary *post*. We also need to make sure we are adding the data to the correct collection, so we put “*collection.*” before the *insert_one()* function to do it.

b) Look for Client and update client:

When we press “Chercher client” button, this will open a new form in which we should put the name we want to look for. Once, name completed we should press the button “Chercher”.

Thanks to MongoDB, we must not complete all the name. If we write the beginning of the name, it will look for all client with the same beginning of the name.

Here is the code we should use to use the regular expression:

```
def Chercher(self, btn):
    self.db, self.client = dbConn()
    self.collection = self.db.Clients
    self.listeclients = self.collection.find({"Nom": {'$regex': self.inputNom.text}})
```

Here is the result for research with the name "R". If there are more than one client with a name beginning with 'R' or 'r', we will see a list of clients. Then we should confirm the one we want to see or update.

Now if I press the button confirm we will have the same form than the one for the add operation with a button update.

Nom
Roquero
Prenom
Pedro
Date de naissance
29/11/1998
Sexe
M
Status social(*)
C
Numéro de registre national(*)
Pays d'origine(*)
GSM
0492567897
Fixe(*)
Adresse
Avenue Saint Antoine 24
Email
update

If we change some Input and press in update button, it will update the database directly

Here is the code to update the database:

```
def update(self, btn):
    post= {}
    for elem in range(len(self.liste_lab)):
        if self.liste_input[elem].text != "":
            post[self.liste_lab[elem].text] = self.liste_input[elem].text
    self.collection.update_one(self.clientdb, {"$set" : post})
    self.client.close()
    self.clear_widgets()
    self.menu()
```

Now, you know how to use the app.

4. Links:

[Présentation Nosql.pptx](#)

<https://github.com/Mustafalou/ProjectNosql.git>

MongoDB:

[MongoDB Atlas: Cloud Document Database | MongoDB](#)

Python:

[Welcome to Python.org](#)

Docs of pymongo:

[Tutorial — PyMongo 3.12.1 documentation](#)