Simple Python Flask Program with MongoDB

Sarathlal Saseendran, 6 Aug 2018

★★★★★ 5.00 (2 votes) Rate this:

This is a sample Python Flask program uses mongodb as database.

Introduction

In this article, we are going to create a simple Python Flask application with MongoDB as database.

Background

We use **Flask** framework to build REST APIs and also use **Pymongo** to connect flask with **MongoDB**.

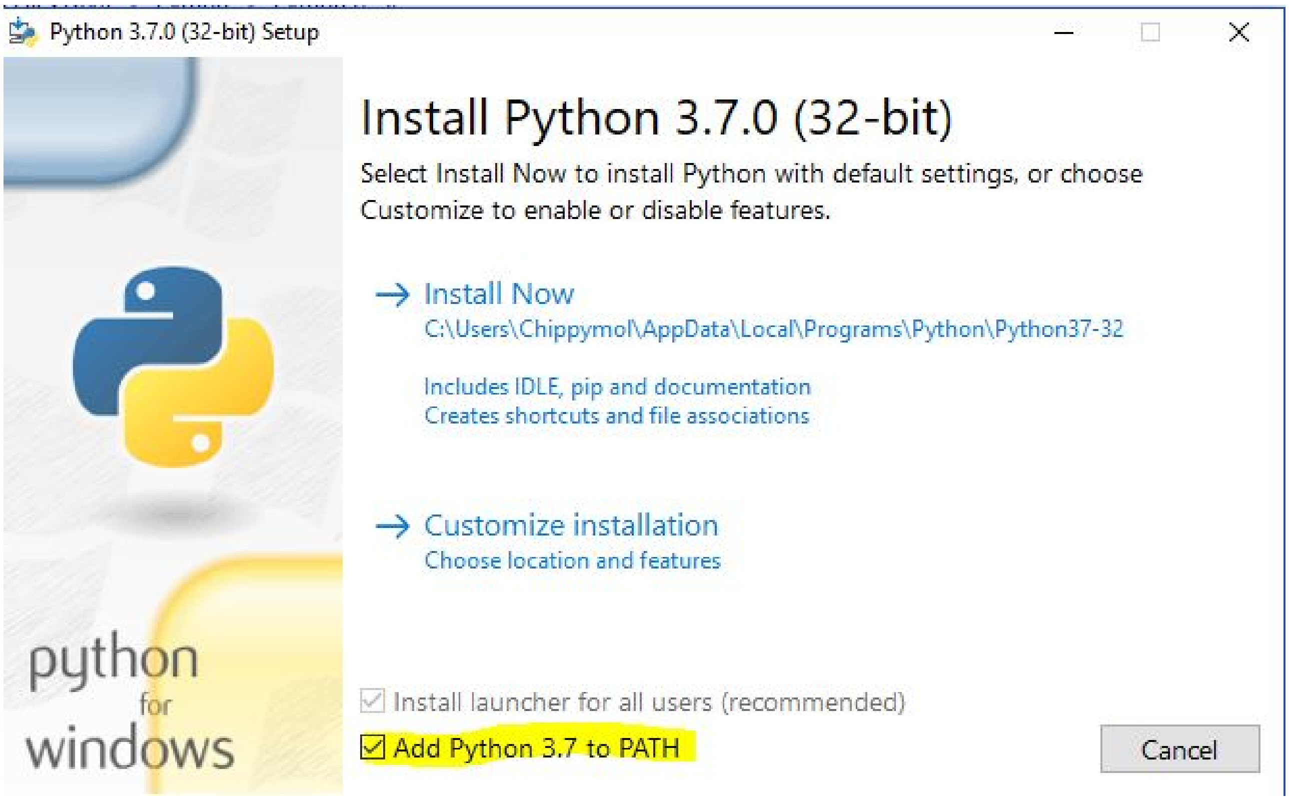
Implementation

We are going to create a sample python web application using Flask framework and MongoDB. It is very easy to work with flask as well as mongodb.

Download python from https://www.python.org/downloads/.

You can select the Customize installation for adding python environment variables.

If you choose default installation, please set these variables manually.


Python 3.7.0 (32-bit) Setup

Install Python 3.7.0 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

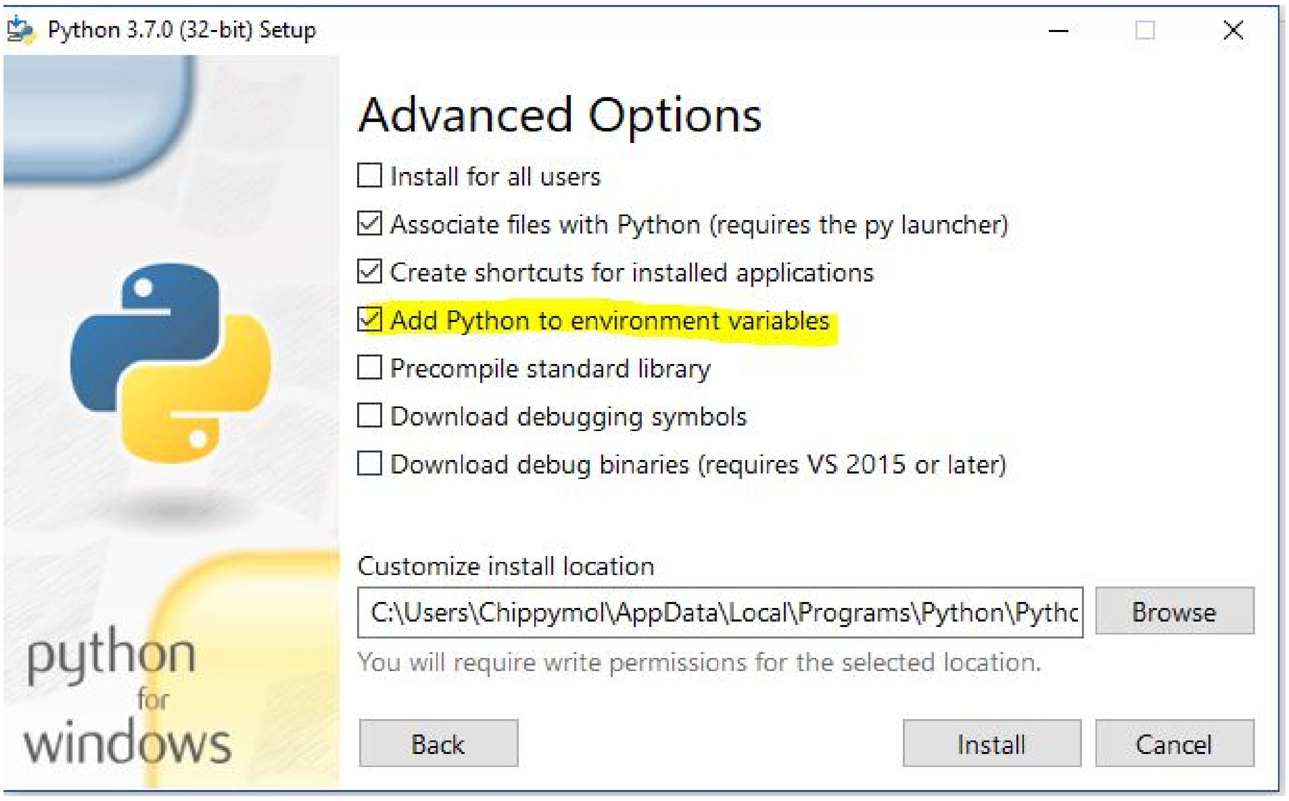
→ Install Now
C:\Users\Chippymol\AppData\Local\Programs\Python\Python37-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

Please click Add Python 3.7 to **PATH** option. It will add one entry in environment variable.


Python 3.7.0 (32-bit) Setup

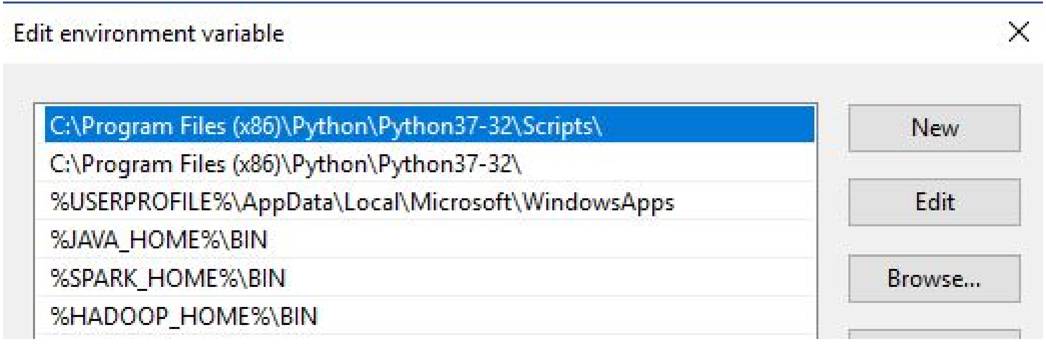
Advanced Options

☐ Install for all users
☑ Associate files with Python (requires the py launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☐ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2015 or later)

Customize install location
C:\Users\Chippymol\AppData\Local\Programs\Python\Pythc
You will require write permissions for the selected location.

If needed, you can modify the installation path. Please select the **Python to environment variables**.

After successful installation, if you check the system environment variable, you can see the below two entries are added in the system.



C:\Program Files (x86)\Python\Python37-32\Scripts\
C:\Program Files (x86)\Python\Python37-32\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
%JAVA_HOME%\BIN
%SPARK_HOME%\BIN
%HADOOP_HOME%\BIN

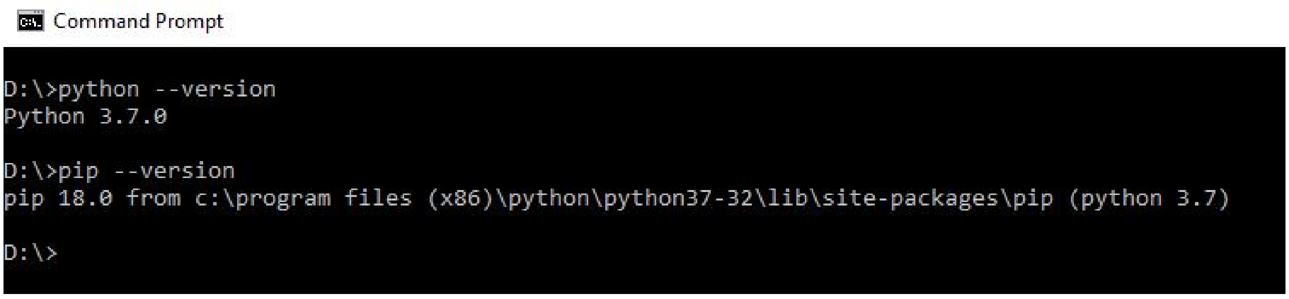
We installed the python 3.7 version, so it added as **Python 37-32**.

Also note that *Python37-32\Scripts* folder is used for handling additional packages needed for python.

Now you can check the python and PIP version in command prompt. PIP is the abbreviation for python package index which is used for installing, upgrading or removing the additional python libraries from our system.

Hide Copy Code

```
> python --version and pip -- version
```



```
D:\>python --version
Python 3.7.0

D:\>pip --version
pip 18.0 from c:\program files (x86)\python\python37-32\lib\site-packages\pip (python 3.7)

D:\>
```

By default, pip version may not be 18. This is the latest version as of today. If your pip version is older than it, please upgrade it.

Hide Copy Code

```
>python -m pip install -U pip
```

Now we are going to install Flask. Flask is a web framework for python. Flask provides you with tools, libraries and technologies that allow you to build a web application in python.

Its dependencies are:

- **`Werkzeug`** a WSGI utility library
- `jinja2` which is its template engine

WSGI is basically a protocol defined so that Python application can communicate with a web-server and thus be used as web-application outside of CGI.

`Jinja2` is used for creating views in flask application. It holds all the static files like HTML, CSS and JavaScript files.

First, install Flask using pip command in command prompt.

Hide Copy Code

```
>pip install Flask
```

It will install the latest version of `Flask` library from its global repository and in Windows machines, it saved the below file location:

Hide Copy Code

```
C:\Program Files (x86)\Python\Python37-32\Lib\site-packages
```

Our sample application uses `MongoDB` as database. So, we are going to install `MongoDB` from the below URL.

- https://www.mongodb.com/download-center#community

It is a free community version. After successful installation, it is all set to run `mongodb` instance.

Before running the `mongodb` instance, we must create a data folder and run the below command in command prompt.

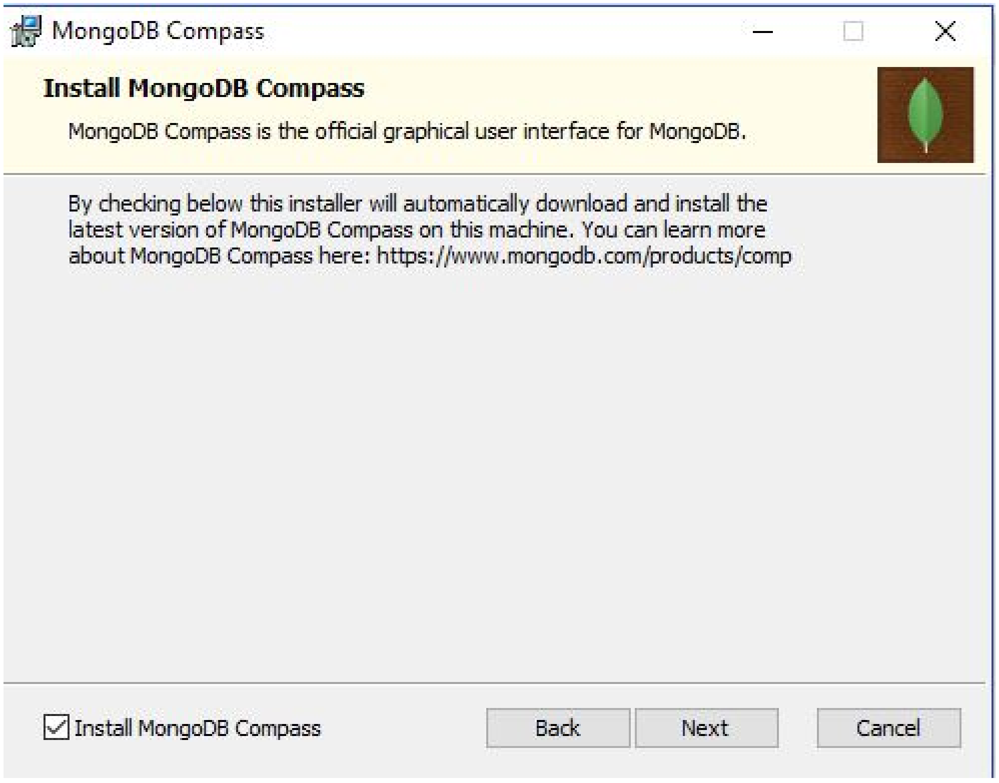
Hide Copy Code

```
"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="C:\mongo-data"
```

Here, *C:\mongo-data* folder is used for saving `mongodb` files.

While installing the `mongodb` in Windows system, you can choose the compass community edition also.

This is a free edition and will help to handle our `mongodb` databases, collections (tables in `mongodb`) and documents (records) graphically.


MongoDB Compass

Install MongoDB Compass

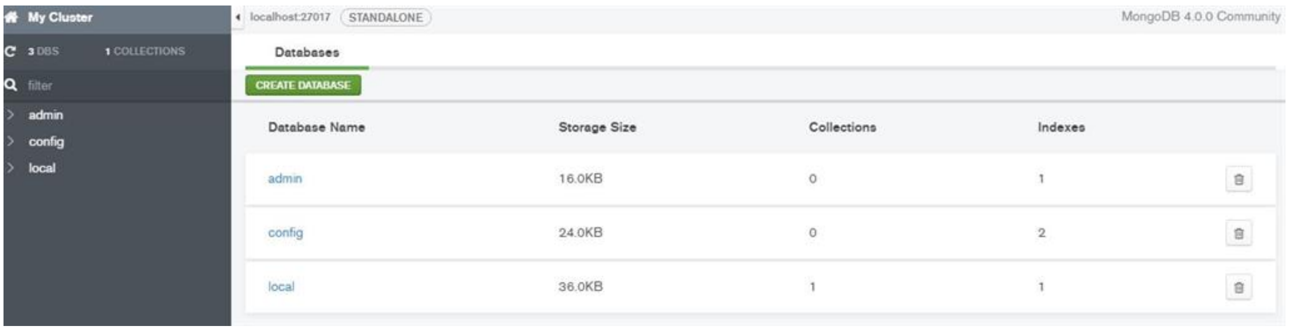
MongoDB Compass is the official graphical user interface for MongoDB.

By checking below this installer will automatically download and install the latest version of MongoDB Compass on this machine. You can learn more about MongoDB Compass here: https://www.mongodb.com/products/comp

[✓] Install MongoDB Compass

By default, `mongodb` is running in localhost **27017** port.

We can see that there are 3 databases (`admin`, `config` and `local`) that are created automatically in the installation time.



Database Name	Storage Size	Collections	Indexes
admin	16.0KB	0	1
config	24.0KB	0	2
local	36.0KB	1	1

Now we are going to install `PyMongo` library in python. `PyMongo` is a simple but powerful python distribution containing tools for working with `mongodb` and is the recommended way to work with `mongodb` from python. It's a kind of ODM (object document mapper). There is `mongoengine` and so many libraries also available as ODM for python.

Hide Copy Code

```
>pip install pymongo
```

We need to install another Python library `bson` too. It is used for getting `objectId` property of `mongodb` document.

Hide Copy Code

```
>pip install bson
```

Now we are all set and ready to start our Python program.

Create a folder named *FlaskwithMongo* and add two sub folders, *static* and *templates* inside it.

There is a convention to name like *static* and *templates* used in `jinja2` framework.

Please open the folder in any of the code editors. I am using Visual Studio code as an IDE.

Create a new file named *app.py*. This is the only Python file we have used in this application.

First, we are going to import the required libraries into our application.

Hide Copy Code

```
from flask import Flask, render_template,request,redirect,url_for # For flask
implementation
from bson import ObjectId # For ObjectId to work
from pymongo import MongoClient
import os
```

Now we are going to declare the app variable in our program.

Hide Copy Code

```
app = Flask(__name__)
```

This app variable is used in the entire application.

Now we are going to declare two variables, `title` and `heading` which are later used in `jinja2` template.

Hide Copy Code

```
title = "TODO sample application with Flask and MongoDB"
heading = "TODO Reminder with Flask and MongoDB"
```

We must declare the connection string for `mongodb` and select a database. Also, we are going to select our collection name to a variable.

Hide Copy Code

```
client = MongoClient("mongodb://127.0.0.1:27017") #host uri
db = client.mymongodb #Select the database
todos = db.todo #Select the collection name
```

As I mentioned earlier, `mongodb` is running in the port **27017** by default. Please note that we have selected `mymongodb` as database and `todo` as collection name. When we create our first transaction, `pymongo` will generate the database and collection automatically. Unlike SQL server or any other RDBMS, `mongodb` doesn't require predefined schemas.

In flask, it is easy to implement routing unlike any other programming language.

For that, we are using `render_template`, `request`, `redirect` and `url_for`. All these methods helped us to establish redirection and show HTML templates in the browser.

Hide Copy Code

```
def redirect_url():
    return request.args.get('next') or \
        request.referrer or \
        url_for('index')
```

In the above code, we defined a method named `redirect_url` and it is used to redirect page to index page.

Hide Copy Code

```
@app.route("/")
@app.route("/uncompleted")

def tasks ():
    #Display the Uncompleted Tasks
    todos_l = todos.find({"done":"no"})
    a2="active"
    return render_template('index.html',a2=a2,todos=todos_l,t=title,h=heading)
```

In the above code, we defined a method named `tasks` and it is used for two routes. One for default "/" route and another for "*/uncompleted*" route. In this code, we defined a variable `todos_l` and it gets the documents from `mongodb` filter by condition done equal to no. We defined one more variable named `a2` and it is used for controlling active records. Both `todos_l` and `a2` variables passed with other two variables, title and heading and passed to the `jinja2` template *index.html* which we must create in the template folder.

Now we are going to finish all the remaining route definitions for adding and deleting the documents to `mongodb`.

app.py

Hide Shrink ▲ Copy Code

```
from flask import Flask, render_template,request,redirect,url_for # For flask
implementation
from bson import ObjectId # For ObjectId to work
from pymongo import MongoClient
import os

app = Flask(__name__)

title = "TODO sample application with Flask and MongoDB"
heading = "TODO Reminder with Flask and MongoDB"

client = MongoClient("mongodb://127.0.0.1:27017") #host uri
db = client.mymongodb #Select the database
todos = db.todo #Select the collection name

def redirect_url():
    return request.args.get('next') or \
        request.referrer or \
        url_for('index')

@app.route("/list")
def lists ():
    #Display the all Tasks
    todos_l = todos.find()
    a1="active"
    return render_template('index.html',a1=a1,todos=todos_l,t=title,h=heading)

@app.route("/")
@app.route("/uncompleted")
def tasks ():
    #Display the Uncompleted Tasks
    todos_l = todos.find({"done":"no"})
    a2="active"
    return render_template('index.html',a2=a2,todos=todos_l,t=title,h=heading)

@app.route("/completed")
def completed ():
    #Display the Completed Tasks
    todos_l = todos.find({"done":"yes"})
    a3="active"
    return render_template('index.html',a3=a3,todos=todos_l,t=title,h=heading)

@app.route("/done")
def done ():
    #Done-or-not ICON
    id=request.values.get("_id")
    task=todos.find({"_id":ObjectId(id)})
    if(task[0]["done"]=="yes"):
        todos.update({"_id":ObjectId(id)}, {"$set": {"done":"no"}})
    else:
        todos.update({"_id":ObjectId(id)}, {"$set": {"done":"yes"}})
    redir=redirect_url()
    return redirect(redir)

@app.route("/action", methods=['POST'])
def action ():
    #Adding a Task
    name=request.values.get("name")
    desc=request.values.get("desc")
    date=request.values.get("date")
    pr=request.values.get("pr")
    todos.insert({ "name":name, "desc":desc, "date":date, "pr":pr, "done":"no"})
    return redirect("/list")

@app.route("/remove")
def remove ():
    #Deleting a Task with various references
    key=request.values.get("_id")
    todos.remove({"_id":ObjectId(key)})
    return redirect("/")

@app.route("/update")
def update ():
    id=request.values.get("_id")
    task=todos.find({"_id":ObjectId(id)})
    return render_template('update.html',tasks=task,h=heading,t=title)

@app.route("/action3", methods=['POST'])
def action3 ():
    #Updating a Task with various references
    name=request.values.get("name")
    desc=request.values.get("desc")
    date=request.values.get("date")
    pr=request.values.get("pr")
    id=request.values.get("_id")
    todos.update({"_id":ObjectId(id)}, {'$set':{ "name":name, "desc":desc, "date":date,
"pr":pr }})
    return redirect("/")

@app.route("/search", methods=['GET'])
def search():
    #Searching a Task with various references
    key=request.values.get("key")
    refer=request.values.get("refer")
    if(key=="_id"):
        todos_l = todos.find({refer:ObjectId(key)})
    else:
        todos_l = todos.find({refer:key})
    return render_template('searchlist.html',todos=todos_l,t=title,h=heading)

if __name__ == "__main__":
    app.run()
```

We must add the below three HTML files in the *templates* folder (*index.html*, *searchlist.html* and *update.html*).

All these three files used the features of `jinja2` framework to render the model values from *app.py*.

In addition, we must add *emoji.css*, *emoji.js*, *style.css* and *twemoji.min.js* files in the *static\assets* folder.

There are two images, *no.png* and *yes.png* in *static\images* folder.

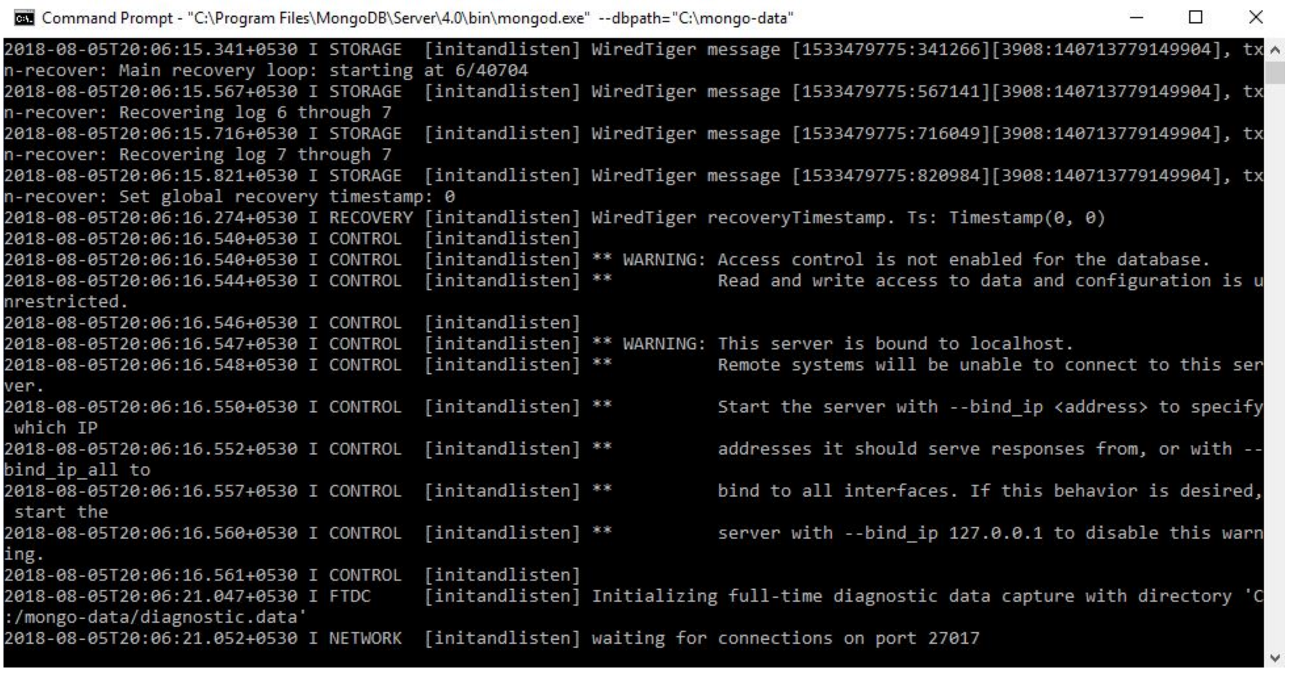
We are ready to run our sample application.

Please start our `mongodb` instance with the below command:

Hide Copy Code

```
"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="C:\mongo-data"
```

Please note that `mongodb` now runs with data folder, *C:\mongo-data*.


Command Prompt - "C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="C:\mongo-data"

```
2018-08-05T20:06:15.341+0530 I STORAGE  [initandlisten] WiredTiger message [1533479775:341266][3908:140713779149904], tx
n-recover: Main recovery loop: starting at 6/40704
2018-08-05T20:06:15.567+0530 I STORAGE  [initandlisten] WiredTiger message [1533479775:567141][3908:140713779149904], tx
n-recover: Recovering log 6 through 7
2018-08-05T20:06:15.716+0530 I STORAGE  [initandlisten] WiredTiger message [1533479775:716049][3908:140713779149904], tx
n-recover: Recovering log 7 through 7
2018-08-05T20:06:15.821+0530 I STORAGE  [initandlisten] WiredTiger message [1533479775:820984][3908:140713779149904], tx
n-recover: Set global recovery timestamp: 0
2018-08-05T20:06:16.274+0530 I RECOVERY [initandlisten] WiredTiger recoveryTimestamp. Ts: Timestamp(0, 0)
2018-08-05T20:06:16.540+0530 I CONTROL  [initandlisten]
2018-08-05T20:06:16.540+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-08-05T20:06:16.544+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is u
nrestricted.
2018-08-05T20:06:16.546+0530 I CONTROL  [initandlisten]
2018-08-05T20:06:16.547+0530 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost.
2018-08-05T20:06:16.548+0530 I CONTROL  [initandlisten] **          Remote systems will be unable to connect to this ser
ver.
2018-08-05T20:06:16.550+0530 I CONTROL  [initandlisten] **          Start the server with --bind_ip <address> to specify
 which IP
2018-08-05T20:06:16.552+0530 I CONTROL  [initandlisten] **          addresses it should serve responses from, or with --
bind_ip_all to
2018-08-05T20:06:16.557+0530 I CONTROL  [initandlisten] **          bind to all interfaces. If this behavior is desired,
 start the
2018-08-05T20:06:16.560+0530 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this warn
ing.
2018-08-05T20:06:16.561+0530 I CONTROL  [initandlisten]
2018-08-05T20:06:21.047+0530 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with directory 'C
:/mongo-data/diagnostic.data'
2018-08-05T20:06:21.052+0530 I NETWORK  [initandlisten] waiting for connections on port 27017
```

By default, it is listening to the port **27017**.

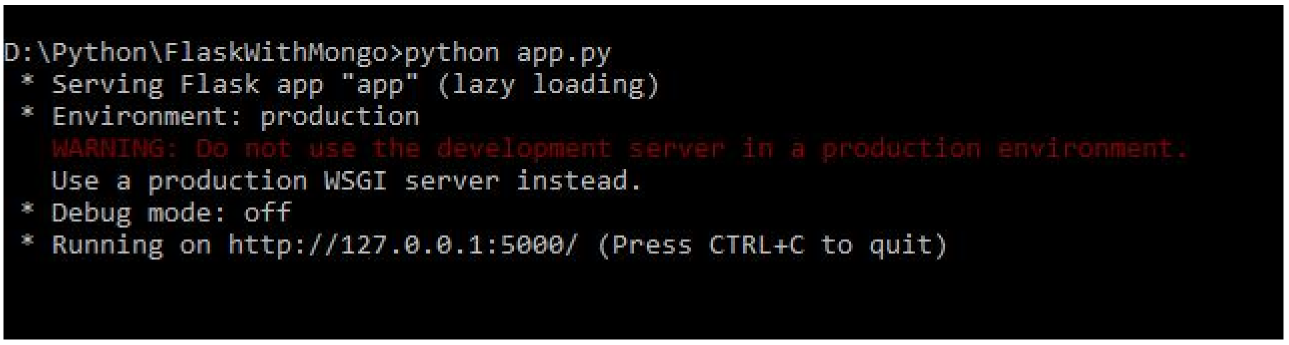
Please open another command prompt window in the same Python application folder and run python with below command.

For example:

Hide Copy Code

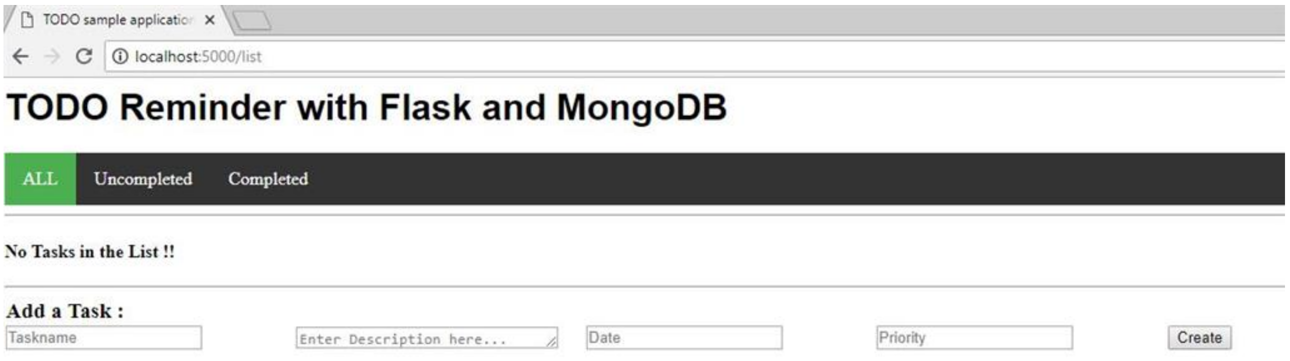
```
D:\Python\FlaskWithMongo>python app.py
```

Our local web server is running in port **5000** by default.



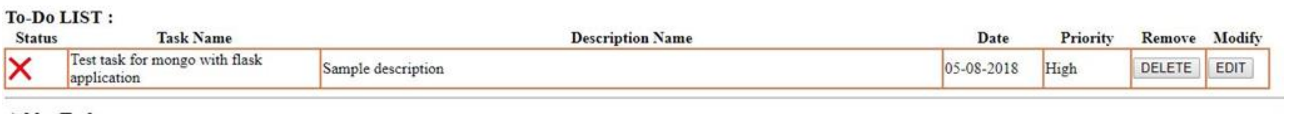
```
D:\Python\FlaskWithMongo>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Application is now running successfully.



You can now add a sample task as task name “**Test task for mongo with flask application**”, description “**Sample description**”, date “**05-08-2018**" and priority “**High**”

After clicking Create button, we can immediately view the task details in the grid.

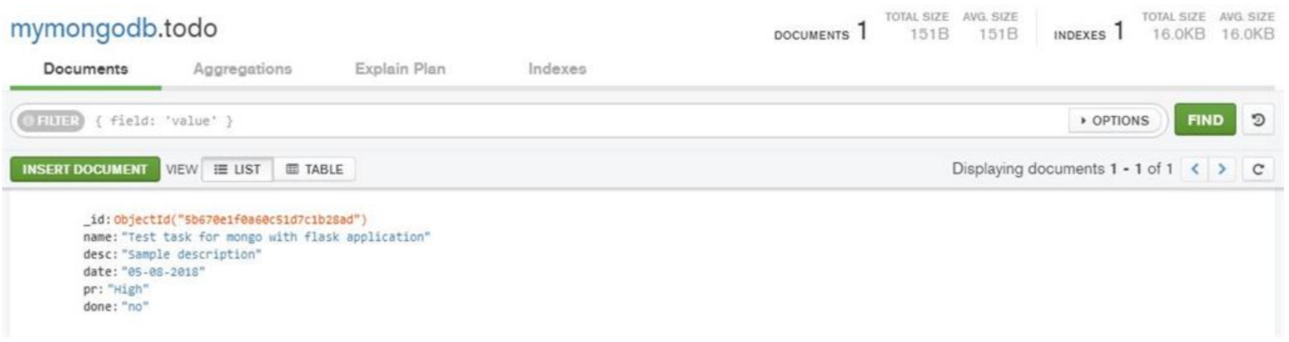


To-Do LIST :

Status	Task Name	Description Name	Date	Priority	Remove	Modify
✗	Test task for mongo with flask application	Sample description	05-08-2018	High	DELETE	EDIT

Please note that there is a new database named `mymongodb` and collection `todo` is created now. Using this application, you can edit and remove the existing documents easily.

You can check the document details in the **MongoDB Compass Community** too.



```
_id: ObjectId("5b670e1f0a60c51d7c1b28ad")
name: "Test task for mongo with flask application"
desc: "Sample description"
date: "05-08-2018"
pr: "High"
done: "no"
```

Happy coding with Flask and MongoDB!!!

The source code is available at Github.

License

Share

About the Author

Sarathlal Saseendran

Technical Lead Orion India Systems P Ltd
India

A passionate human being loves to learn new things always.

Full stack developer interested in .Net, Azure, Scala, Spark, Angular 6 and now Python.

https://www.linkedin.com/in/sarathlal-saseendran-28478b70

You may also be interested in...

Dockerize A Simple Web-Application Created By Using Python, Flask and PostgreSQL

Complete Sudoku Game for Windows using VB.NET 2013

MongoDB and C#

OOP in Python - part 1

Slightly Less Costly but Much Usable C++ Reflection with Singular Inheritance Rule

Single Page Apps with Flask and Angular 4|5 Tutorial Series

Comments and Discussions

You must Sign In to use this message board.

Search Comments

First Prev Next

Can't find the html files for downloading in the article
Member 9608930 8-May-19 3:27

Hi Sarathlal,

Can you please provide the download link for the html files mentioned in your articles.

Sign In · View Thread

Can't find the html files for downloading in the article
Member 9608930 8-May-19 3:27

Hi Sarathlal,

Can you please provide the download link for the html files mentioned in your articles.

Sign In · View Thread

A small comment (repeated)
Richard MacCutchan 6-Aug-18 21:05

> **Quote:**
> "C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="C:\mongo-data"
>
> Here C:\mongo-data folder is used for saving mongodb files.

You should not be storing user data in the root of the C: drive. It can lead to access problems and application crashes. Use a proper data directory such as the user's `Documents` folder, or an appropriate place in in `AppData` (`Local` or `Roaming`).

Sign In · View Thread

Re: A small comment (repeated)

Sarathlal Saseendran 6-Aug-18 21:22

Thank you Richard for your valuable suggestion.

Sign In · View Thread

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin