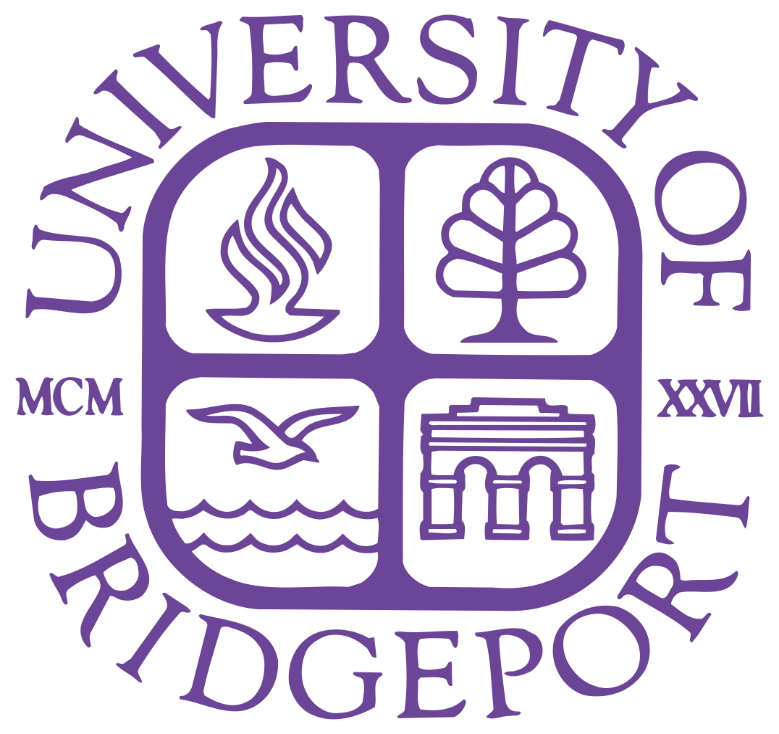
UNIVERSITY OF BRIDGEPORT

SPRING-2017



CPSC652 Hadoop and NoSQL Databases

Term Project Phase-3

Mongo DB Implementation Manual

By Group 3:

Nitish Makam Prashanth ID: 1005189

**Introduction:**

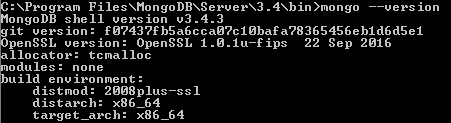
Although EJDB was one of our databases for implementation, due to platform compatibility and installation issues, here is an implementation of MongoDB instead. We chose MongoDB as a substitute since EJDB is heavily inspired by MongoDB. The implementation will be a simple collection of blog posts. Each blog post will have an author, title, content, tags, and date information. Features included are insert, search, update, remove, and bulk upload. Here is a link to a quick demo video: <https://youtu.be/OqtXFu8mrck>



Screenshot of the program (MongoDB\_implementation.py)

**Prerequisites:**

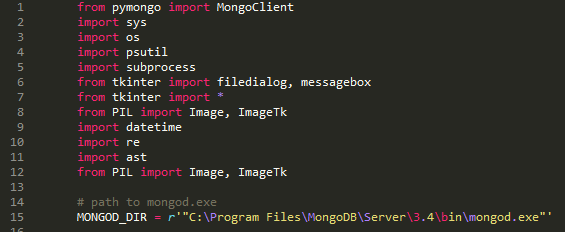
To run the program successfully, you must have first installed MongoDB.



As a reference, my operating system was Windows 7 64-bit, so I installed the version recommended by the MongoDB website. If you are in windows you must have created C:\data\db or customize the path for recording data when you set up MongoDB.

Next you must have python 3 installed without problems and when you run MongoDB\_Implementation.py, run it in python 3 and not in python 2. Then you must install or already have all the libraries listed in the first section of the python code. These are easily installable through pip install command.

Lastly, you must have MongoDB running while you run this program. The program automatically checks for this, but you must correct the path, MONGO\_DIR, in the python code if the MongoDB directory does not match with the code.

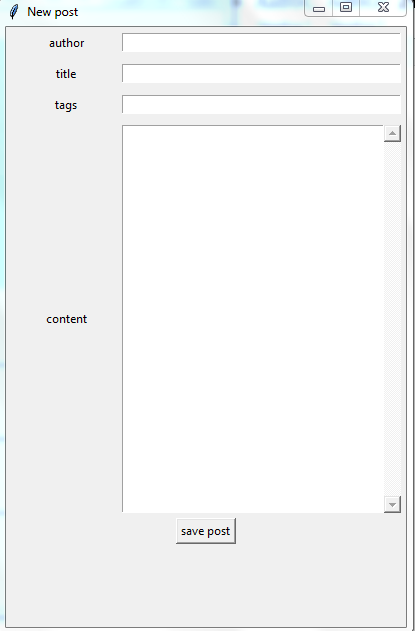


**Instructions:**

To run the program, all you need to do is click on MongoDB\_Implementation.py or call it in the terminal. Then a window will appear as below.

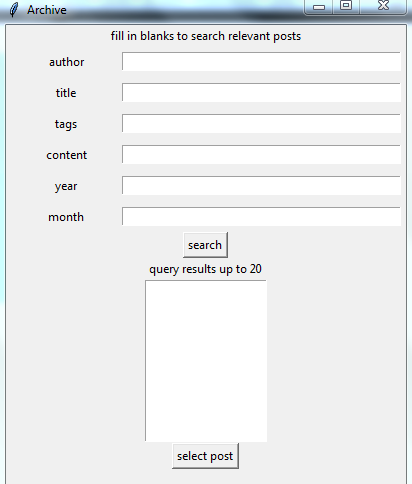


My program has already 2 blog posts, but yours will have 0. To add a post, simply click on the ‘write a new post’ button. Another popup window will appear as below.

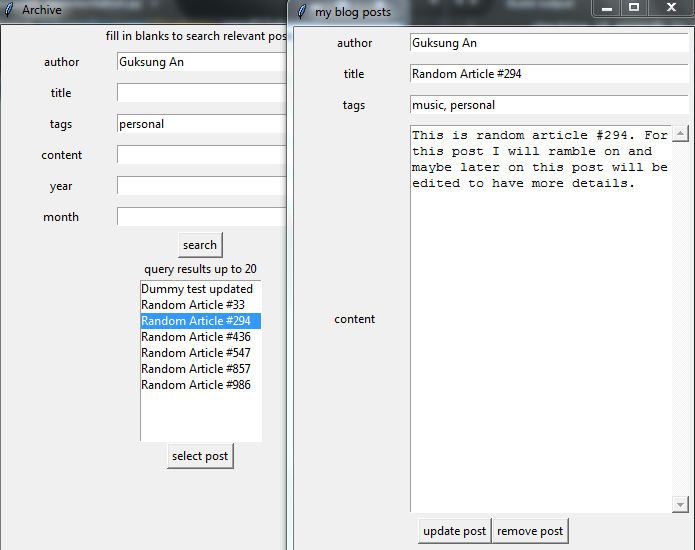


Simply fill in the fields and after you click on the ‘save post’ button, the blogpost object will be stored into the MongoDB database under BlogCollection.blogposts. The blog post will also contain an \_id that is unique among all blog posts and time information that is captured when you saved the post.

What if you want to view the blog posts you saved? Then you click on the ‘search archive’ button. Another popup window will appear as below.

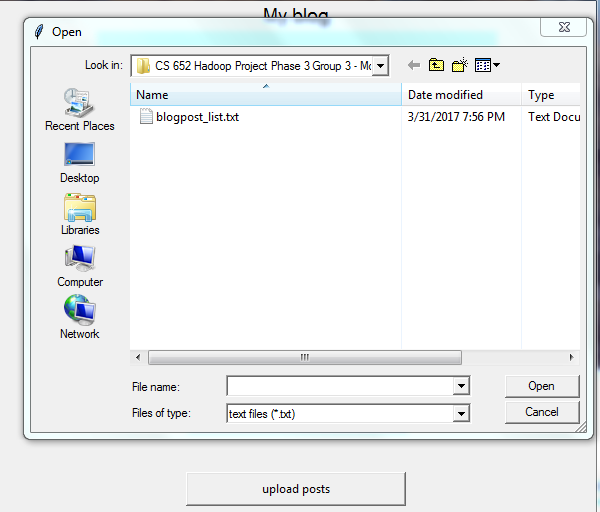


Simply fill in the information as you need to search posts containing such information. Any fields you leave blank will not be considered in the query. After you press search, you may select out of the posts that returned. Here is an example:



As you can see, the tags and author of the selected post match the query. In addition, you may update the post by changing some fields or remove the post entirely.

If you do not want to tediously add blog posts to test the interface, you may bulk upload by clicking on ‘upload posts’ button. Clicking so will open a file dialog looking for a text file. We have provided a blogpost\_list.text that has been generated by random\_posts\_generator.py. You may generate the random list again by executing random\_posts\_generator.py.



If you want to make your own text file, ensure that the text is in the same format as the blogpost\_list.txt. It is essentially a collection string representations of python object separated by each line. After successfully uploading posts into the database and restarting the program, you will see the number of blog posts change.

**Source Code**

**from** **pymongo** **import** MongoClient

**import** **sys**

**import** **os**

**import** **psutil**

**import** **subprocess**

**from** **tkinter** **import** filedialog, messagebox

**from** **tkinter** **import** \*

**from** **PIL** **import** Image, ImageTk

**import** **datetime**

**import** **re**

**import** **ast**

**from** **PIL** **import** Image, ImageTk

*# path to mongod.exe*

MONGOD\_DIR = r'"C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe"'

*'''*

*Check mongod before running this script...*

*'''*

**print**('checking if mongodb is running...')

processes\_names = []

**for** p **in** psutil.process\_iter():

**try**:

processes\_names.append(p.name())

**except** psutil.NoSuchProcess:

**pass**

mongodb\_running = 'mongod.exe' **in** processes\_names

**if** mongodb\_running:

**print**('mongodb is running. will proceed to interact with the database')

**else**:

**print**('mongodb was not running.')

**print**('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

**print**('running mongod.exe now')

**print**('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

subprocess.Popen(MONGOD\_DIR, shell=True)

**try**:

client = MongoClient()

**except** pymongo.errors.ConnectionFailure **as** e:

**print**('Could not connect to MongoDB', e)

sys.exit(1)

**try**:

**print**('databases:', client.database\_names())

**except** pymongo.erros.ServerSelectionTimeoutError **as** e:

**print**('Run mongod command before running this script')

**raise** e

sys.exit(1)

collection = client.BlogCollection

db = collection.blogposts

**class** **Window**(Tk):

**def** \_\_init\_\_(self, parent=None, title='my blog posts'):

Tk.\_\_init\_\_(self, parent)

self.parent = parent

self.title(title)

self.geometry('600x600')

self.initialize()

**def** initialize(self):

label0 = Label(self, text='My blog', font=('TkDefaultFont', 14))

label0.pack()

img = Image.open('typewriter.jpg')

img = img.resize((400, 250), Image.ANTIALIAS)

photo = ImageTk.PhotoImage(img)

foreground = Label(image=photo, width=400, height=250)

foreground.image = photo

foreground.pack()

Label(self, text='Number of blogs: {}'.format(db.count())).pack()

button0 = Button(self, text='write a new post', command=self.newpost)

button1 = Button(self, text='search archive', command=self.search)

button2 = Button(self, text='upload posts', command=self.loadfile)

**for** button **in** (button0, button1, button2):

button.config(width=30, pady=5)

button.pack(pady=20)

frame = Frame()

frame.pack()

self.mainloop()

**def** newpost(self):

npost = NewPost()

**def** search(self):

archive = PostArchive()

**def** loadfile(self):

openfiletypes = [('text files','.txt')]

*# not specifying initialdir opens current working directory*

fname = filedialog.askopenfilename(filetypes=openfiletypes)

**if** **not** fname:

**return**

**with** open(fname, 'r') **as** file:

**for** line **in** file:

data = line.strip('**\n**')

blogpost = ast.literal\_eval(data)

*# literal\_eval does not support datetime*

blogpost['last edit time'] = datetime.datetime.now()

db.insert\_one(blogpost)

**class** **PostArchive**(Toplevel):

**def** \_\_init\_\_(self):

Toplevel.\_\_init\_\_(self)

self.title('Archive')

self.results\_limit = 20

self.initUI()

**def** initUI(self):

self.geometry('400x600')

Label(self, text='fill in blanks to search relevant posts').pack()

self.fields = ('author', 'title', 'tags', 'content', 'year', 'month')

self.widgets = dict()

**for** field **in** self.fields:

self.widgets[field] = dict()

row = self.widgets[field]['frame'] = Frame(self)

self.widgets[field]['label'] = Label(row, text=field, width=15)

self.widgets[field]['entry'] = Entry(row)

row.pack(side=TOP, fill=X, padx=5, pady=5)

self.widgets[field]['label'].pack(side=LEFT)

self.widgets[field]['entry'].pack(side=RIGHT, expand=YES, fill=X)

self.widgets[self.fields[0]]['entry'].focus()

search = Button(self, text='search', command=self.search)

search.pack()

Label(self, text='query results up to {}'.format(self.results\_limit)).pack()

self.results\_list = Listbox(self)

self.results\_list.pack()

select = Button(self, text='select post', command=self.select)

select.pack()

**def** search(self):

query = dict()

**for** field **in** self.fields:

f = self.widgets[field]['entry'].get()

**if** f:

query[field] = f

self.results = [post **for** post **in** db.find(query)]

self.results\_list.delete(0, END)

**for** title **in** (post['title'] **for** post **in** self.results[:self.results\_limit]):

self.results\_list.insert(END, title)

**if** **not** self.results:

messagebox.showinfo(message='no blog post found')

**def** select(self):

**try**:

choice = self.results\_list.curselection()[0]

**print**('choice', choice)

blogpost = self.results[choice]

showpost = UpdatePost(blogpost)

**except** **IndexError**:

messagebox.showerror(message='you have not selected a post')

**class** **Post**(Toplevel):

**def** \_\_init\_\_(self):

Toplevel.\_\_init\_\_(self)

self.initUI()

**def** initUI(self):

self.geometry('400x600')

self.fields = ('author', 'title', 'tags', 'content')

self.widgets = dict()

**for** field **in** self.fields:

self.widgets[field] = dict()

row = self.widgets[field]['frame'] = Frame(self)

self.widgets[field]['label'] = Label(row, text=field, width=15)

self.widgets[field]['entry'] = Entry(row)

row.pack(side=TOP, fill=X, padx=5, pady=5)

self.widgets[field]['label'].pack(side=LEFT)

self.widgets[field]['entry'].pack(side=RIGHT, expand=YES, fill=X)

self.widgets[self.fields[0]]['entry'].focus()

scrollbar = Scrollbar(self.widgets['content']['frame'])

scrollbar.pack(side=RIGHT, fill=Y)

*# replace entry with text*

self.widgets['content']['entry'].destroy()

self.widgets['content']['entry'] = Text(self.widgets['content']['frame'], wrap=WORD, yscrollcommand=scrollbar.set)

self.widgets['content']['entry'].pack()

scrollbar.config(command=self.widgets['content']['entry'].yview)

**def** getpost(self):

blogpost = dict()

**for** field **in** self.fields:

**if** field == 'content':

*# content has text widget instead of entry*

blogpost[field] = self.widgets[field]['entry'].get(1.0, 'end-1c')

**else**:

blogpost[field] = self.widgets[field]['entry'].get()

blogpost['tags'] = re.split(r', \*', blogpost['tags'])

time = datetime.datetime.now()

blogpost['year'] = time.year

blogpost['month'] = time.month

blogpost['day'] = time.day

blogpost['time'] = str(time.time())

blogpost['last edit time'] = time

**return** blogpost

**def** setpost(self, blogpost):

**for** field **in** self.fields:

text = '' **if** field **not** **in** blogpost **else** blogpost[field]

**if** field == 'tags':

text = ', '.join(text)

entry = self.widgets[field]['entry']

**if** type(entry) == Entry:

entry.delete(0, END)

entry.insert(0, text)

**elif** type(entry) == Text:

entry.insert(1.0, text)

**class** **NewPost**(Post):

**def** initUI(self):

Post.initUI(self)

save = Button(self, text='save post', command=self.savepost)

save.pack()

**def** savepost(self):

blogpost = self.getpost()

**print**('blogpost**\n**', blogpost)

db.insert\_one(blogpost)

messagebox.showinfo(message='successfully saved')

**class** **UpdatePost**(Post):

**def** \_\_init\_\_(self, blogpost):

self.blogpost = blogpost

Post.\_\_init\_\_(self)

**def** initUI(self):

Post.initUI(self)

self.setpost(self.blogpost)

bottom = Frame(self)

bottom.pack()

update = Button(bottom, text='update post', command=self.updatepost)

update.pack(side=LEFT)

remove = Button(bottom, text='remove post', command=self.removepost)

remove.pack(side=LEFT)

**def** \_id(self):

**return** {'\_id': self.blogpost['\_id']}

**def** updatepost(self):

*# remove and insert because of http://stackoverflow.com/questions/25128974/cant-replace-mongo-document*

self.removepost()

updated = self.getpost()

**for** date **in** ('year', 'month', 'day', 'time'):

updated[date] = self.blogpost[date]

*# updated['last edit time'] = datetime.datetime.now()*

db.insert\_one(updated)

messagebox.showinfo(message='successfully updated')

*# db.update\_one(self.\_id(), self.getpost())*

**def** removepost(self):

db.remove(self.\_id())

messagebox.showinfo(message='successfully removed')

*# messagedialog*

**def** main():

app = Window()

**if** \_\_name\_\_ == '\_\_main\_\_':

main()