Table of Contents

[I. Introduction to Django 2](#_Toc121530620)

[II. Tools Setup 2](#_Toc121530621)

[1. Download django 2](#_Toc121530622)

[2. Create Django project and Django app 2](#_Toc121530623)

[3. Project Structure 2](#_Toc121530624)

[III. Code Development 4](#_Toc121530625)

[1. Database Setup and CRUD operations function definition 4](#_Toc121530626)

[2. Create URLs 5](#_Toc121530627)

[3. Backend Code using Pymongo 5](#_Toc121530628)

[a. Read 5](#_Toc121530629)

[b. Create 6](#_Toc121530630)

[a. Update 7](#_Toc121530631)

[d. Delete 8](#_Toc121530632)

[e. Queries 8](#_Toc121530633)

[f. Aggregation 9](#_Toc121530634)

[g. Data Manipulation 10](#_Toc121530635)

[h. Create html templates 10](#_Toc121530636)

[i. Explanation of html format used to design UI 11](#_Toc121530637)

[j. Explanation of Django APIs used in development of CRUD operations 13](#_Toc121530638)

[IV. Execution and Testing 15](#_Toc121530639)

[1. Run server 15](#_Toc121530640)

[V. References: 18](#_Toc121530641)

1. Introduction to Django

In the mongoDB website(https://www.mongodb.com/compatibility/mongodb-and-django), it mentions django as the recommended web framework to build applications working with MongoDB databases. In this project we Django as the development web server together with pymongo for writing code to do CRUD operations on our database.

Django puts data in requests. We call it POST when a user submits a form with the attribute method=”post”

1. Tools Setup
2. Download django

**pip install pymongo**

**pip install dnspython**

1. Create Django project and Django app

django-admin startproject proj\_1

cd proj\_1

python manage.py startapp djangoapp

pip list

Make sure to have the same versions as the list below if not make sure to install with those specific versions.

***Verify Output from pip list:***

***pip 22.3.1***

***pymongo 4.3.3***

***dnspython 2.2.1***

***Django 4.1.3***

1. Project Structure

Text

Description automatically generated

Default files generated when creating a Django application are as follows:

settings.py - a normal Python module with module-level variables representing Django settings.

urls.py – define url paths and corresponding function to call

views.py – define views or so called

admin.py – not utilized (more likely used with djongo)

apps.py - not utilized (more likely used with djongo)

models.py - not utilized (more likely used with djongo)

tests.py – not utilized

1. Code Development
2. Adding our app for list of activated apps in the Django instance when it is run

Need to add the djangoapp in the installed apps so theat in the future the urls will be recognized.

We are using pymongo not djongo hence need to comment out the DATABASES declaration in the settings.

Text

Description automatically generated

1. Database Setup and CRUD operations function definition

Database setup is instead done on the views.py where we create the connection string, get the client from there and the instance of our database.

It is also in the views where we create the function definitions for each collection.

A screenshot of a computer

Description automatically generated with medium confidence

1. Create URLs

We assign different URLs accordingly to CRUD operation for each collection

Text

Description automatically generated

1. Backend Code using Pymongo
   1. Read

def show\_product(request, productId):

    product\_info = myCollection\_Products.find\_one({"productId": productId})

* 1. Create

# Function called when adding a new product where it will check if there is form data first or not

# add\_product

def prod\_info(request):

    # Upon next access is when the user clicks submit he is simply sending it back to the same URL now with method POST means with data info this FUNCTION which will be able to detect the method POST and begin to oarse data from the form and build the document to insert

    # HttpRequest.POST -  A dictionary-like object containing all given HTTP POST parameters, providing that the request contains form data

    if request.method == "POST":

        form = request.POST

        print(form)

        format = "%Y-%m-%dT%H:%M" #https://www.freecodecamp.org/news/how-to-convert-a-string-to-a-datetime-object-in-python/

        creationDate = datetime.strptime(form['creationDate'], format)

        expiryDate = datetime.strptime(form['expiryDate'], format)

        # Here we manually render out a html form

        # form values are taken from POST httresponse once the user has submitted where it is taken from the tag name in the html

        product\_info = {

            'productId' : form['productId'],

            'productName' : form['productName'],

            'productDescription' : form['productDescription'],

            'productCategory' : form['productCategory'],

            'price' : float(form['price']),

            'totalStock' : int(form['remaining']), #totalstock at first is always equal to the remaining one

            'stockDetails' : #If it's the first time adding the product then for sure there will only be one stock first hence we hardcode the array as only having 1 element at first creation

                 [

                {

                'creationDate' : creationDate,

                'expiryDate' : expiryDate,

                'quantity' : int(form['quantity']),

                'remaining' : int(form['remaining'])

                },

                 ],

            'isOffered' : form['isOffered']

        }

        myCollection\_Products.insert\_one(product\_info)

        return redirect('/product/show/all')  #If you addded a new product you are automatically redirected to the product summary

    else:

        form = {

        }

    # The first access to the URL with no POST data it will submit a blank form

    return render(request,'product\_add.html',{'form':form})

* 1. Update

def update\_product(request, productId):

    form = request.POST

    format = "%Y-%m-%dT%H:%M" #https://www.freecodecamp.org/news/how-to-convert-a-string-to-a-datetime-object-in-python/

    creationDate = datetime.strptime(form['creationDate'], format)

    expiryDate = datetime.strptime(form['expiryDate'], format)

    product\_new = {

        "$set":

        {

            'productId' : form['productId'],

            'productName' : form['productName'],

            'productDescription' : form['productDescription'],

            'productCategory' : form['productCategory'],

            'price' : float(form['price']),

            'totalStock' : int(form['remaining']), #At edit reset should always be same as remaining

            'stockDetails' :

                [

                {

                'creationDate' : creationDate,

                'expiryDate' : expiryDate,

                'quantity' : int(form['quantity']),

                'remaining' : int(form['remaining'])

                },

                ],

            'isOffered' : form['isOffered']

        }

    }

    myCollection\_Products.update\_one({"productId": productId}, product\_new)

    return redirect('/product/show/all')

1. Delete

def delete\_product(request, productId):

    product = myCollection\_Products.delete\_one({"productId": productId})

    return redirect("/product/show/all")

1. Queries

def show\_products(request):

    product\_infos = []

    ExpiredFilter = {}

    myQueryDict = {}

    if request.GET.get('getExpired') == 'getExpired':

            print("getExpired clicked!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")

            print(type(datetime.now()))

            ExpiredFilter = {'stockDetails':{'expiryDate': {'$lte':datetime.now()}}} #not working :()

            myQueryDict.update(ExpiredFilter)

    # Accessing with filter inputs

    if request.method == "POST"  :

        form = request.POST

        print(form)

        productNameFilter = {}

        productDescriptionFilter = {}

        productCategoryFilter = {}

        totalStockFilter = {}

        ExpiredFilter = {}

        if form['productNameFilter'] != '':

            productNameFilter = {'productName': {'$regex':form['productNameFilter'], '$options' : 'i'}}

            myQueryDict.update(productNameFilter)

        if form['productDescriptionFilter'] != '':

            productDescriptionFilter = {'productDescription': {'$regex':form['productDescriptionFilter'], '$options' : 'i'}}

            myQueryDict.update(productDescriptionFilter)

        if form['productCategory'] != '': #

            productCategoryFilter = {'productCategory': {'$regex':form['productCategory'], '$options' : 'i'}}

            myQueryDict.update(productCategoryFilter)

        if form['totalStockFilter'] != '':

            totalStockFilter = {'totalStock': {'$lte':int(form['totalStockFilter'])}}

            myQueryDict.update(totalStockFilter)

        # myQueryDict = {'productName': productNameFilter, 'productDescription': productDescriptionFilter, 'productCategory': productCategoryFilter, 'totalStock': totalStockFilter, 'stockDetails': ExpiredFilter}

        # {productName: {$regex:'che', '$options' : 'i'},productDescription: {$regex:'br', '$options' : 'i'}, productCategory: {$regex:'br', '$options' : 'i'}, totalStock: {$lte:15}}

        print(myQueryDict)

        for product\_info in myCollection\_Products.find(myQueryDict):

            product\_infos.append(product\_info)

    # Access first time no filter so just get all {}

    else:

        for product\_info in myCollection\_Products.find({}, {}):

            product\_infos.append(product\_info)

    # render() Combines a given html template with a given context dictionary and \

    # returns an HttpResponse object with that rendered text built in the template for the UI display

    return render(request,"product\_show\_all.html",{'product\_infos':product\_infos})

1. Aggregation

# Compute total Stock

        # https://www.tutorialspoint.com/how-to-sum-every-field-in-a-sub-document-of-mongodb

        myPipeline =  [

                        {'$match':

                            {'productId':productId}

                        },

                        {'$unwind': "$stockDetails" },

                        {'$group':

                            {

                                '\_id': '$\_id',

                                'productId': {'$first': '$productId'},

                                'totalStock': {'$sum': '$stockDetails.remaining'}

                            }

                        }

                    ]

        myAggGroup = myCollection\_Products.aggregate(myPipeline)

        myAggGroupList = list(myAggGroup)

        print(myAggGroupList)

        # [{'\_id': ObjectId('63918fac5a3a753879383892'),

        #                         'productId': '5550',

        #                         'totalStock': 42}]

        updatedTotalStock = myAggGroupList[0]['totalStock']

        # Update the totalStock element after adding new stock

        product\_elem\_new = {

            "$set":

            {

                'totalStock' : updatedTotalStock,

            }

        }

        myCollection\_Products.update\_one({"productId": productId}, product\_elem\_new)

1. Data Manipulation

 format = "%Y-%m-%dT%H:%M" #https://www.freecodecamp.org/news/how-to-convert-a-string-to-a-datetime-object-in-python/

        creationDate = datetime.strptime(form['creationDate'], format)

        expiryDate = datetime.strptime(form['expiryDate'], format)

1. Create html templates

For our Gui, we create a new folder named templates to which we group the html files corresponding to the design of each page url

Text

Description automatically generated

1. Explanation of html format used to design UI
   1. <form> tag

Django provides a Form class which is **used to create HTML forms**.

            <div class="form-group row">

                <label for=id\_productId class="col-sm-2 col-form-label">productNameFilter</label>

                <div class="col-sm-4">

                    <input type="text" name="productNameFilter" id="id\_productNameFilter"  maxlength="100"

                        value="" />

                </div>

            </div>

Name attribute will refer to the form key identifier

In above example key identifier is productNameFilter and so the value inputted in the value attribute will be assign to the value of form[‘productNameFilter’]

* 1. “get”

Submit action is monitored through method=’get’

    <form method='get' action='#'>

        <input type="submit" value="getExpired" name="getExpired"/>

    </form>

* 1. “POST”

we have an HTML form so that user can submit form data as method = “POST” httpresponse

    <form method="POST" class="post-form" action="/product/add">

        {% csrf\_token %}

POST response is sent to the action url which then handles the request accordingly:

def prod\_info(request):

    # Upon next access is when the user clicks submit he is simply sending it back to the same URL now with method POST means with data info this FUNCTION which will be able to detect the method POST and begin to oarse data from the form and build the document to insert

    # HttpRequest.POST -  A dictionary-like object containing all given HTTP POST parameters, providing that the request contains form data

    if request.method == "POST":

        form = request.POST

        print(form)

from here the form containing all the form data can be analyzed

* 1. Render

    # render() Combines a given html template with a given context dictionary and \

    # returns an HttpResponse object with that rendered text built in the template for the UI display

    return render(request,"product\_show\_all.html",{'product\_infos':product\_infos})

* 1. Redirect simply directs access to given url

return redirect("/product/show/all")

* 1. Improve Design through bootstrap

[Bootstrap · The most popular HTML, CSS, and JS library in the world. (getbootstrap.com)](https://getbootstrap.com/)

<!-- CSS only -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">

1. Explanation of Django APIs used in development of CRUD operations
   1. CREATE

In views we define the functions that will handle the data submitted from the POST in html.

Based on the corresponding collection html page from where POST came from we build the information and assign the value from the form

1. if request.method == "POST":
2. form = request.POST
3. print(form)
4. customer\_info = {
5. 'emailAddress' : form['emailAddress'],
6. 'firstName' : form['firstName'],
7. 'lastName' : form['lastName'],
8. 'billingAddress' :
9. {
10. 'bStreet' : form['bStreet'],
11. 'bUnit' : form['bUnit'],
12. 'bCity' : form['bCity'],
13. 'bProvince' : form['bProvince'],
14. 'bZip' : form['bZip'],
15. 'bMobileNumber' : form['bMobileNumber']
16. },
17. 'deliveryAddress' :
18. {
19. 'dStreet' : form['dStreet'],
20. 'dUnit' : form['dUnit'],
21. 'dCity' : form['dCity'],
22. 'dProvince' : form['dProvince'],
23. 'dZip' : form['dZip'],
24. 'dMobileNumber' : form['dMobileNumber']
25. },
26. 'paymentMethod' :
27. {
28. 'paymentType' : form['paymentType'],
29. 'nameOnCard' : form['nameOnCard'],
30. 'cardScheme' : form['cardScheme'],
31. 'cardLast4' : form['cardLast4'],
32. 'cardExpiryDate' : form['cardExpiryDate']
33. }
34. }

We then use .insert one to insert the new document onto the given collection.

        print(customer\_info)

        myCollection\_Customers.insert\_one(customer\_info)

After insertion, we want the user to go the the main page that shows the whole collection

        return redirect('/customer/show')

1. Execution and Testing
2. Run server

Python manage.py runserver

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

1. References:

<https://www.youtube.com/watch?v=vNegKfkopoQ>

<https://docs.djangoproject.com/en/4.1/ref/models/options/#table-names>

<https://www.youtube.com/watch?v=FLZYLog369s>

<https://code.djangoproject.com/wiki/NoSqlSupport>

<https://www.youtube.com/watch?v=sjMlffIVGf8>

<https://www.youtube.com/watch?v=c45JJfgMyWo>

<https://www.youtube.com/watch?v=I17uA1sVQ2g>

<https://www.digitalocean.com/community/tutorials/how-to-connect-a-django-app-to-mongodb-with-pymongo>

<https://www.mongodb.com/compatibility/mongodb-and-django>

<https://www.mongodb.com/compatibility/mongodb-and-django>

https://www.javatpoint.com/django-crud-application