iNeuron

REVIEWS				
Product	Name	Rating	Comment Heading	Comments
samsungA8	Soumya Guha Roy	4	Beauty & the beast combined.	I have been using the device since last 3 days. Here is my review Pros: 1. Excellent display 2. Fast focus and fast capture camera 3. Nice UI 4. Decent battery backup (after 24 hours, battery remains ~30% with medium usage. 5. Good performance. 6. Unread notification count is spot on, like we have in iOS. 7. The metallic design is superb, and feels premium. Go for the gold one, than the white. The white one looks plast...
samsungA8	deepak pagar nashik	5	superb ,wesome,stylish, powerful ,metal finishing with complete security handset by samsung	i bought this stunning metal body slim phone before 15 days ,and i m really happy to use it ,this cell having 32 gb internal memory due to this not i used my memory card . its hybrid sim phone but no problem because u have more internal capacity . Samsung Galaxy A8 is a awesome device again from Samsung. A8 could have had Dual SIM along with SD card (u can used here one SIM and one

Review scraper from scratch till deployment

iNeuron

Table of Contents

Preface

This book is intended towards helping all the data scientists out there. It is a step by step guide for creating a web scraper, in this case, a review scrapper right from scratch and then deploying it to the heroku cloud platform. Text scrappers are extensively used in the industry today for competitive pricing, market studies, customer sentiment analysis, etc. This book takes a simple example of an online cell phone purchase and tries to explain the concepts simply, extensively, and thoroughly to create a review scrapper right from scratch and then its deployment to a cloud environment.

Happy Learning!

Web Scraping(Text)

1. Introduction:

Web scraping is a technique using which the webpages from the internet are fetched and parsed to understand and extract specific information similar to a human being. Web scrapping consists of two parts:

- Web Crawling→ Accessing the webpages over the internet and pulling data from them.
- HTML Parsing→ Parsing the HTML content of the webpages obtained through web crawling and then extracting specific information from it.

Hence, web scrappers are applications/bots, which automatically send requests to websites and then extract the desired information from the website output.

Let's take an example:

how do we buy a phone online?

1. We first look for a phone with good reviews
2. We see on which website it's available at the lowest price
3. We check whether it's delivered in our area or not
4. If everything looks good, then we buy the phone.

What if there is a computer program that can do all of these for us? That's what web scrappers necessarily do. They try to understand the webpage content as a human would do.

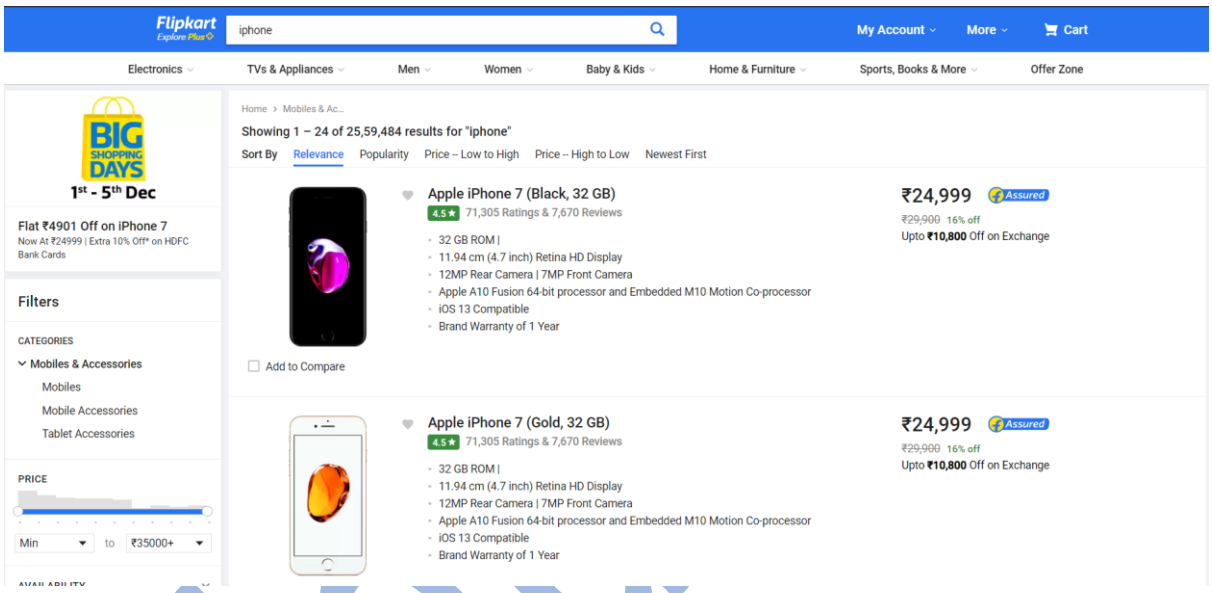
Other examples of the applications of web scrapping are:

- Competitive pricing.
- Manufacturers monitor the market, whether the retailer is maintaining a minimum price or not.
- Sentiment analysis of the consumers, whether they are happy with the services and products or not.
- To aggregate news articles.
- To aggregate Marketing data.
- To gain financial insights from the market.
- To gather data for research.
- To generate marketing leads.

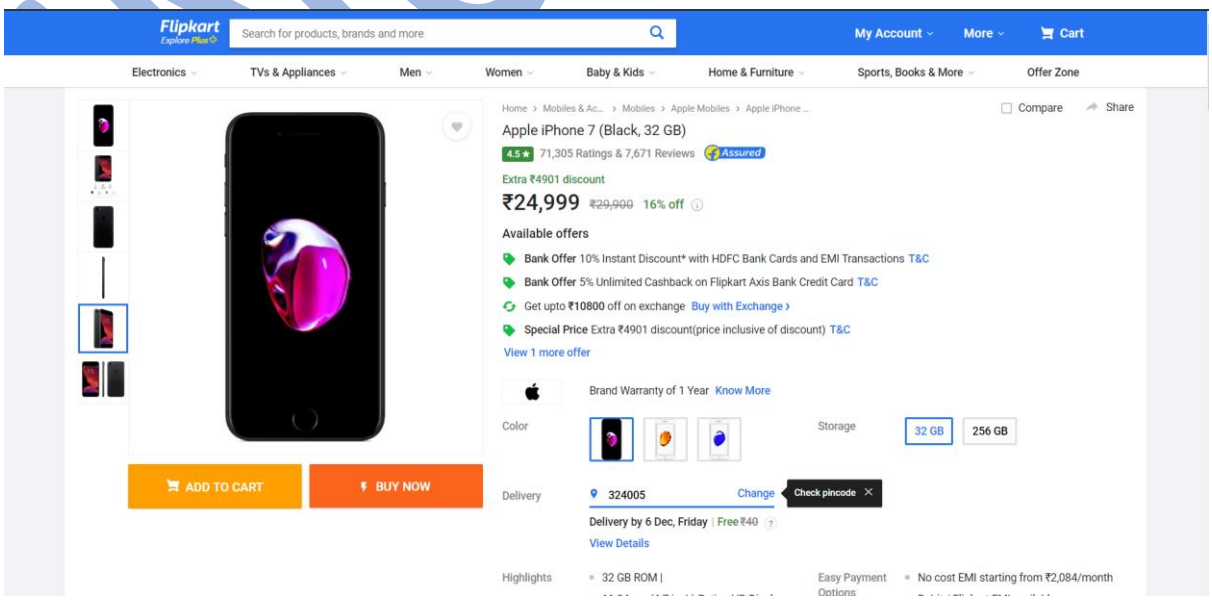
- To collect trending topics by media houses.
  And, the list goes on.

In this document, we'll take the example of buying a phone online further and try to scrap the reviews from the website about the phone that we are planning to buy.

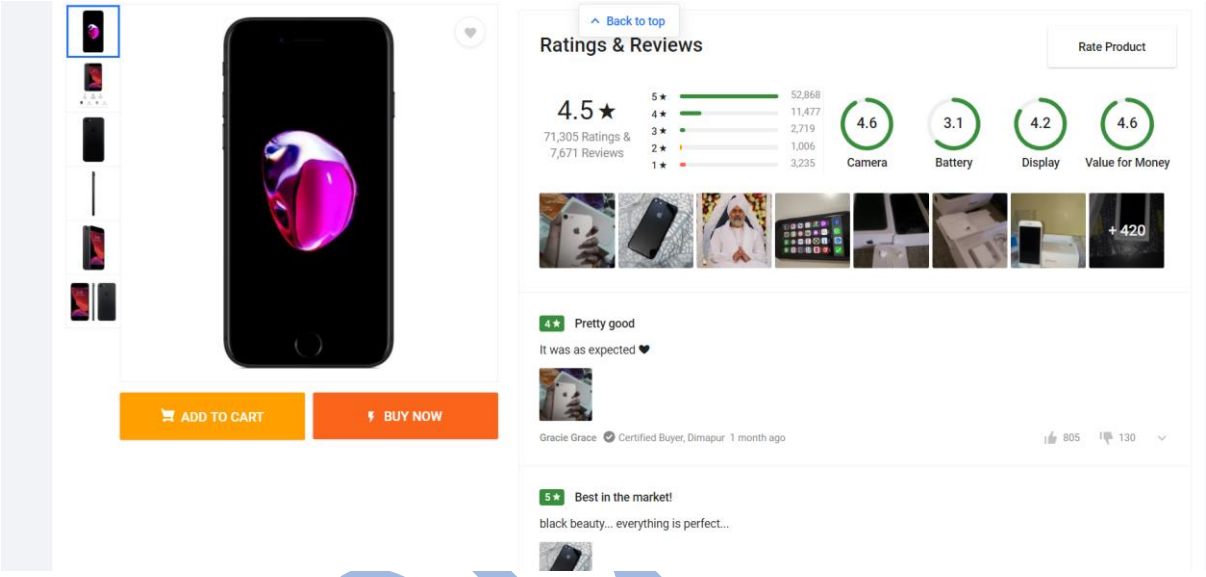
For example, if we open filpkart.com and search for 'iPhone', the search result will be as follows:



Then if we click on a product link, it will take us to to the following page:



If we scroll down on this page, we'll get to see the comments posted by the customers:



Our end goal is to build a web scraper that collects the reviews of a product from the internet.

2. Prerequisites:

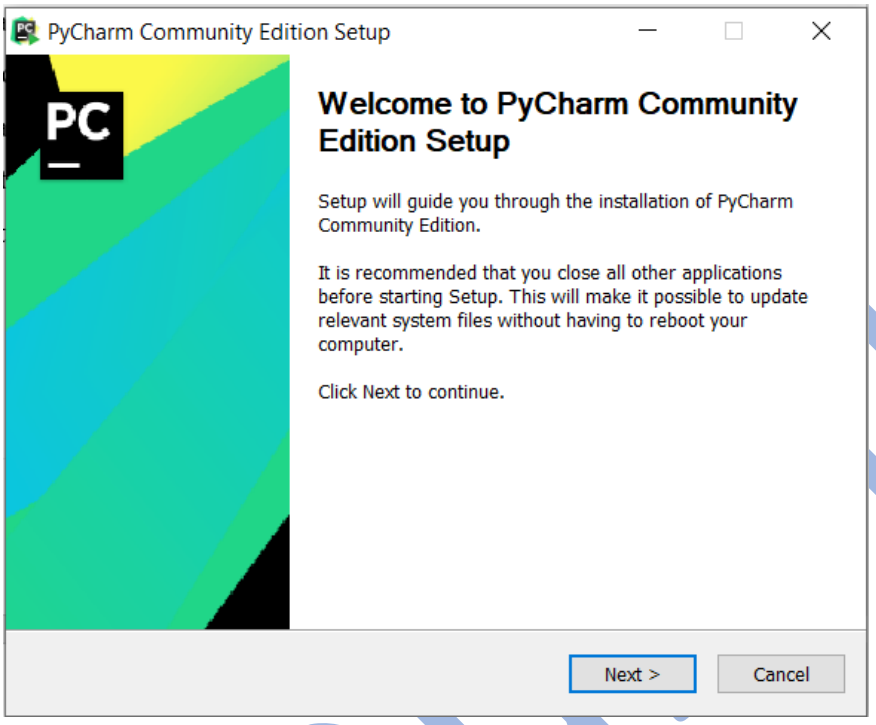
The things needed before we start building a python based web scraper are:

- Python installed.
- A Python IDE (Integrated Development Environment): like PyCharm, Spyder, or any other IDE of choice (Explained Later)
- Flask Installed. (A simple command: pip install flask)
- MongoDB installed (Explained Later).
- Basic understanding of Python and HTML.
- Basic understanding of Git (download Git CLI from https://gitforwindows.org/)

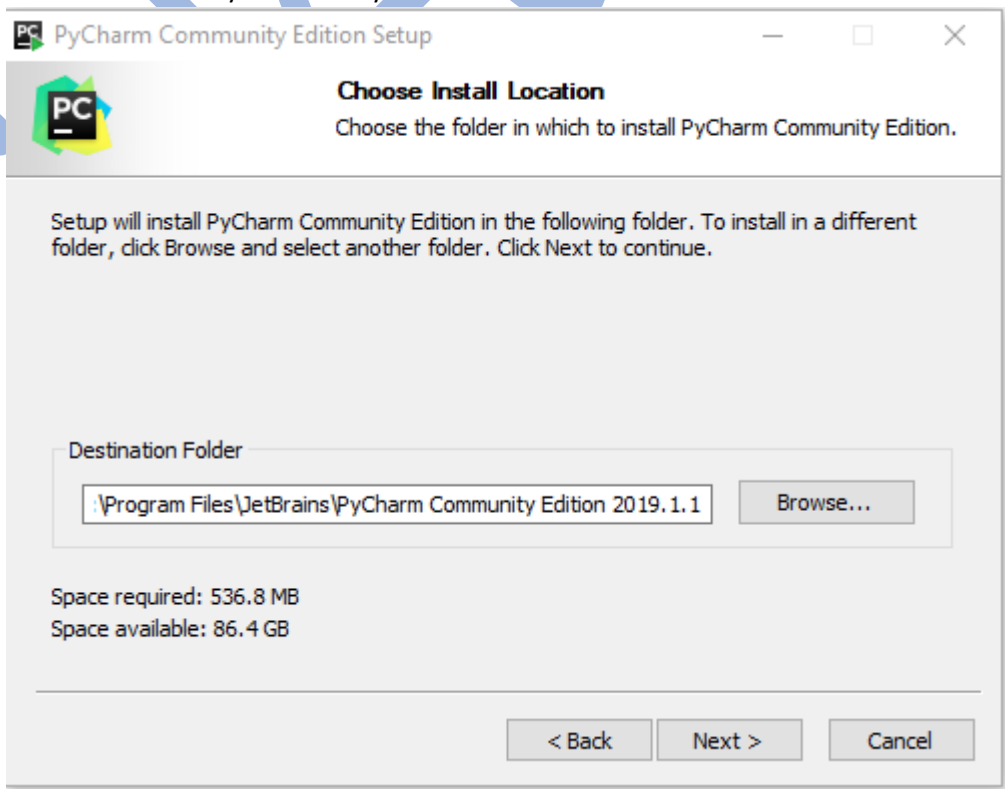
2.1 PyCharm Installation:

a) Go to the link https://www.jetbrains.com/pycharm/download/#section=windows and download the community edition.

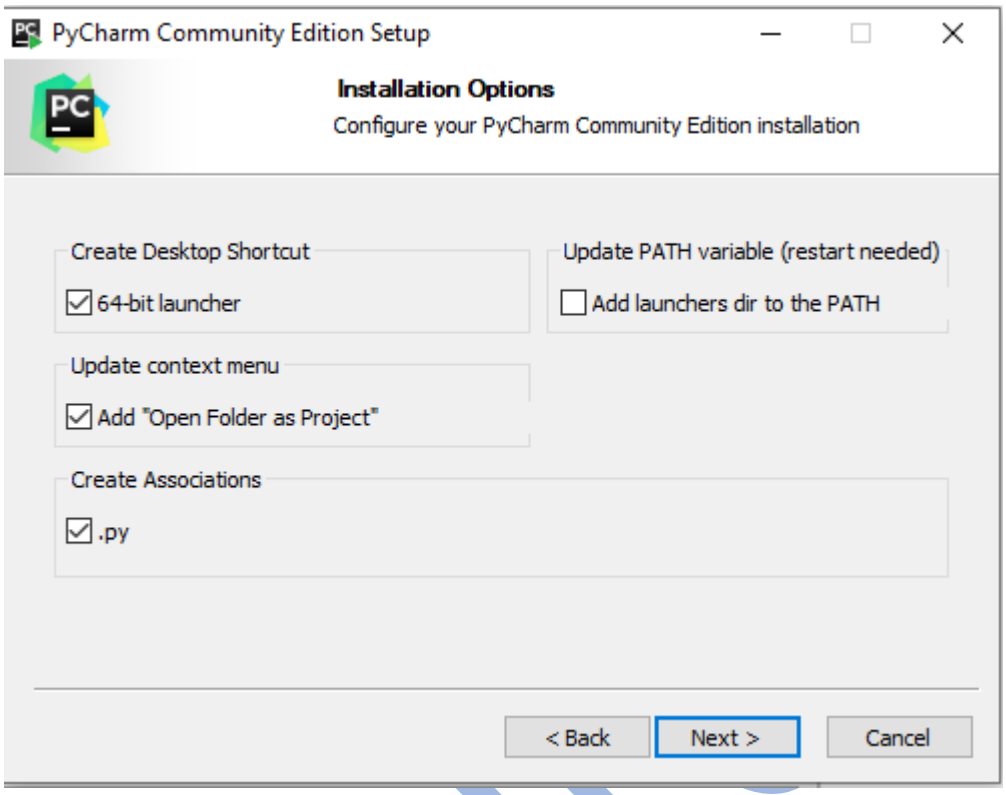
b) Double click on the installation file to start the installation process and click next to continue.



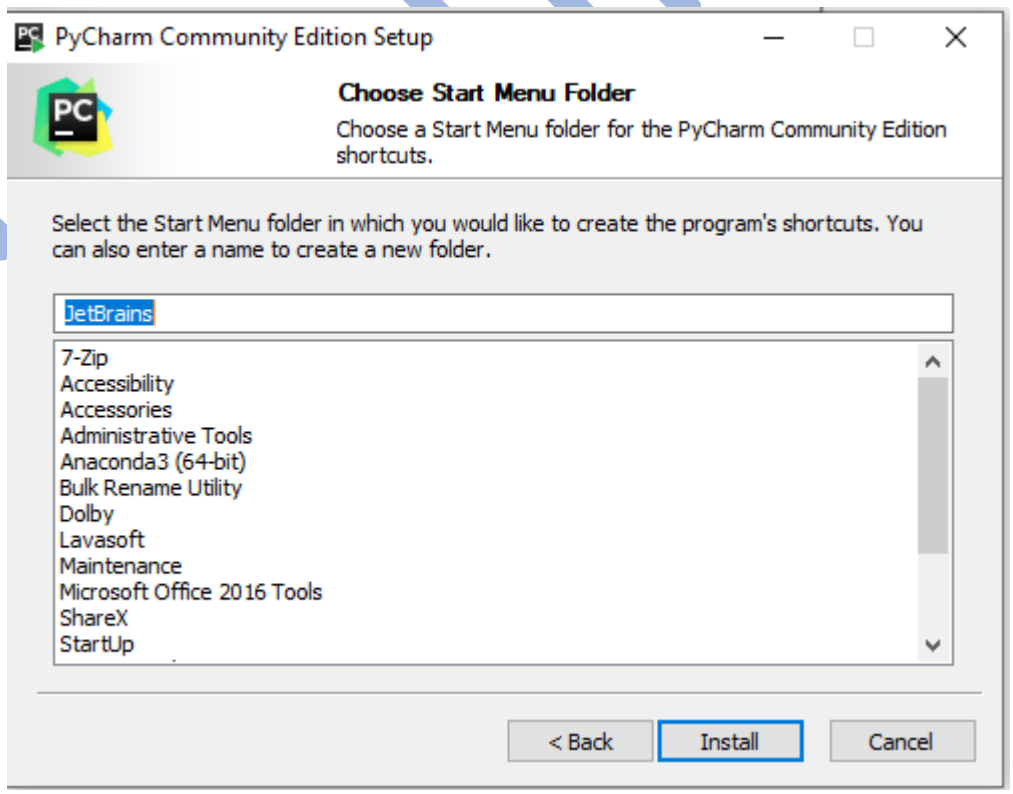
c) Select the directory to install PyCharm and then click next.



d) Check the appropriate checkboxes and then click next.

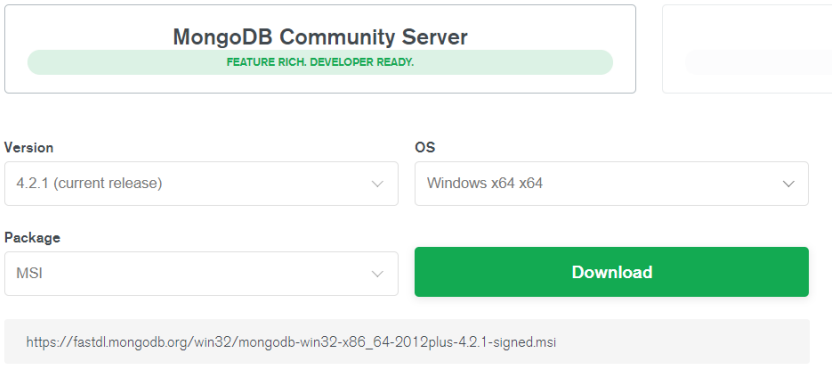


e) Choose the name of the start menu folder and then click on install to finish the installation.

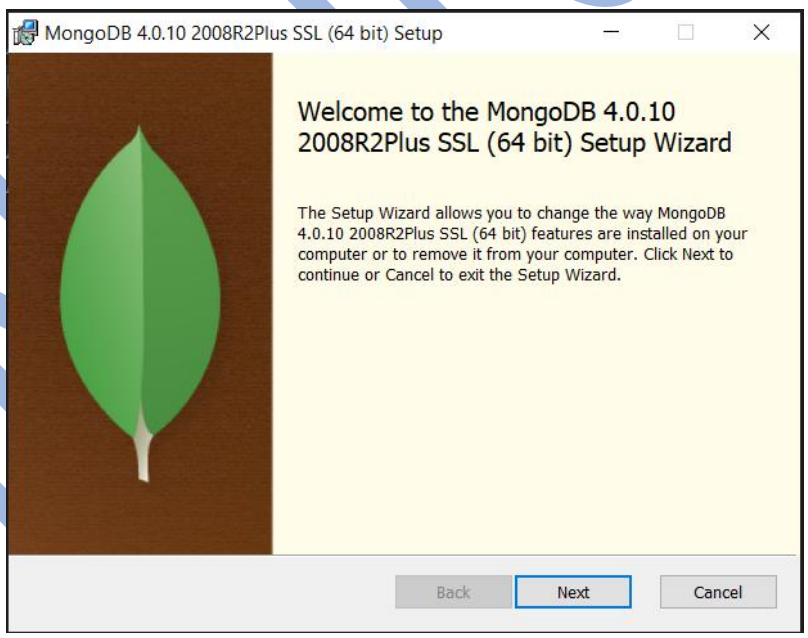


2.2 MongoDB Installation:

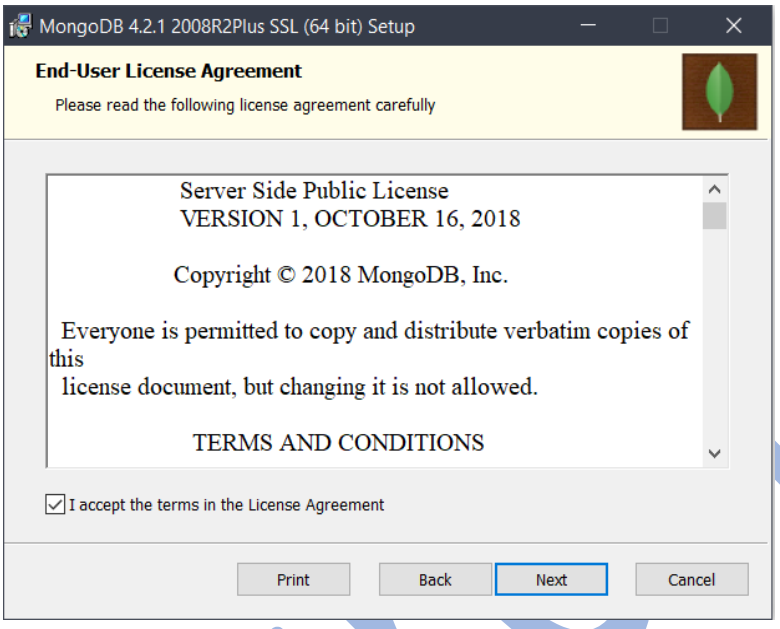
1. Go to the page: https://www.mongodb.com/download-center/community and select the MongoDB installation to download based on your operating system.



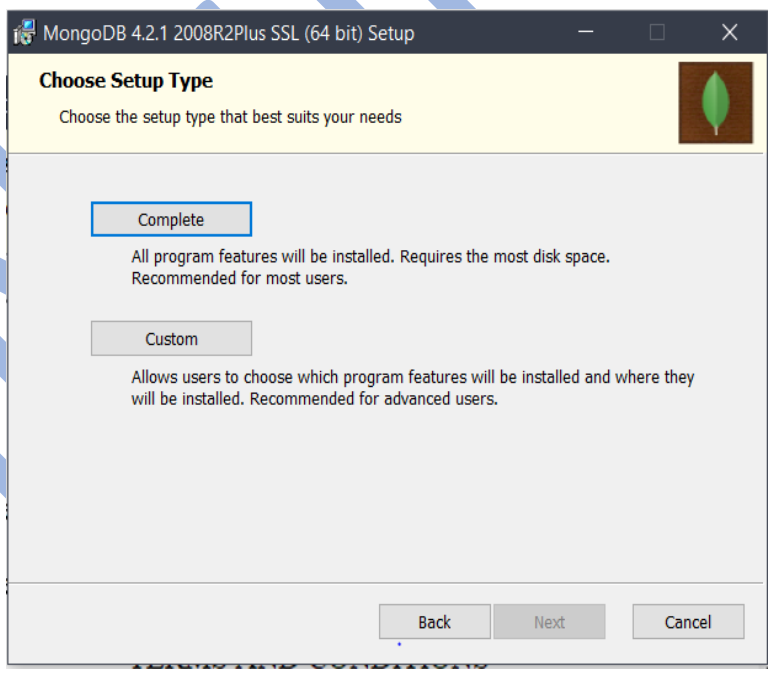
2. After the installer gets downloaded, double click on the installer file to start installing the application.



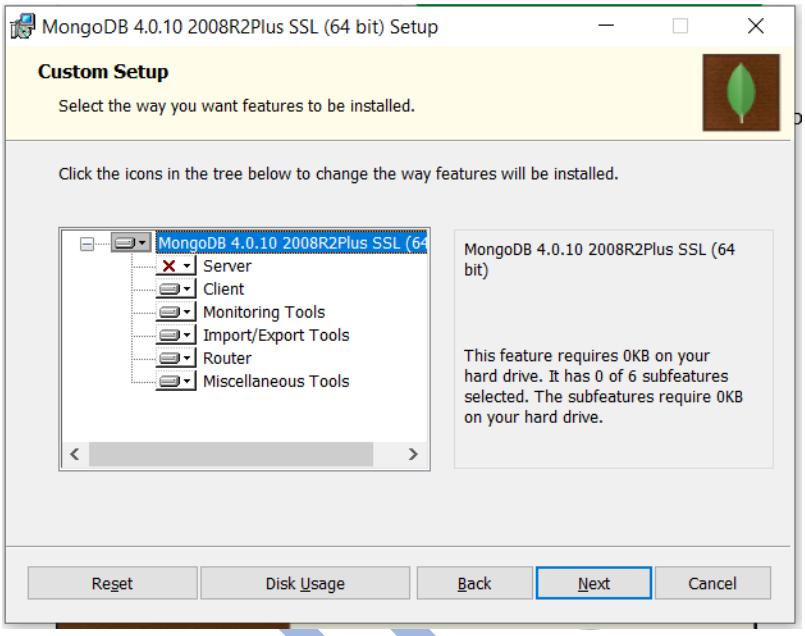
3. Click on the next button to move to the next step and accept the agreement.



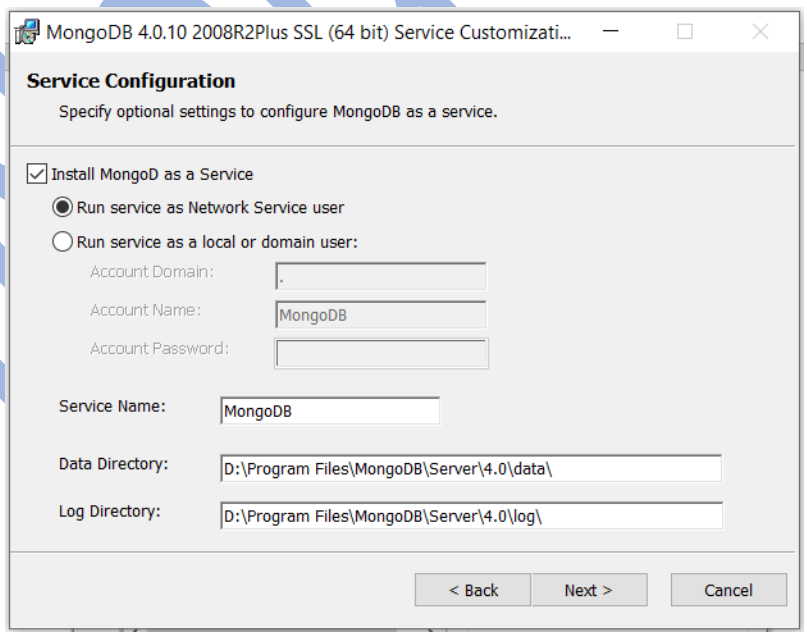
4. Select the type of installation:



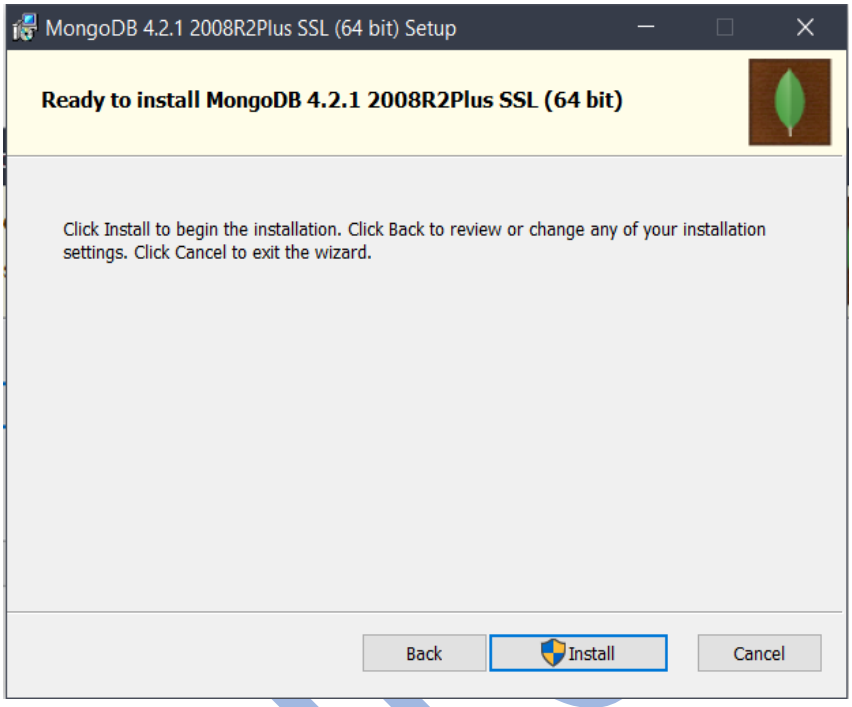
5. Select the features to install.



6. Click on next and then configure/customize the way you want the application to be installed.

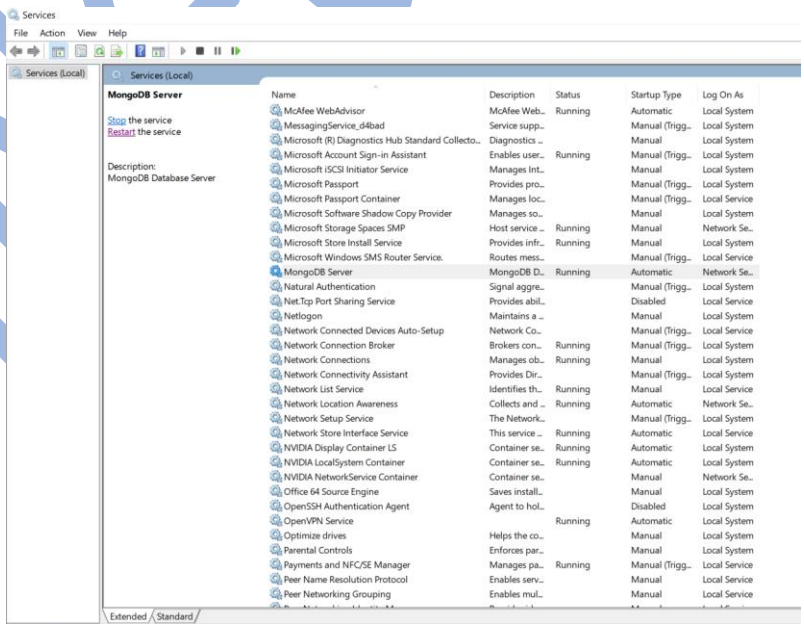


7. Click next and then click on install to start the MongoDB installation.

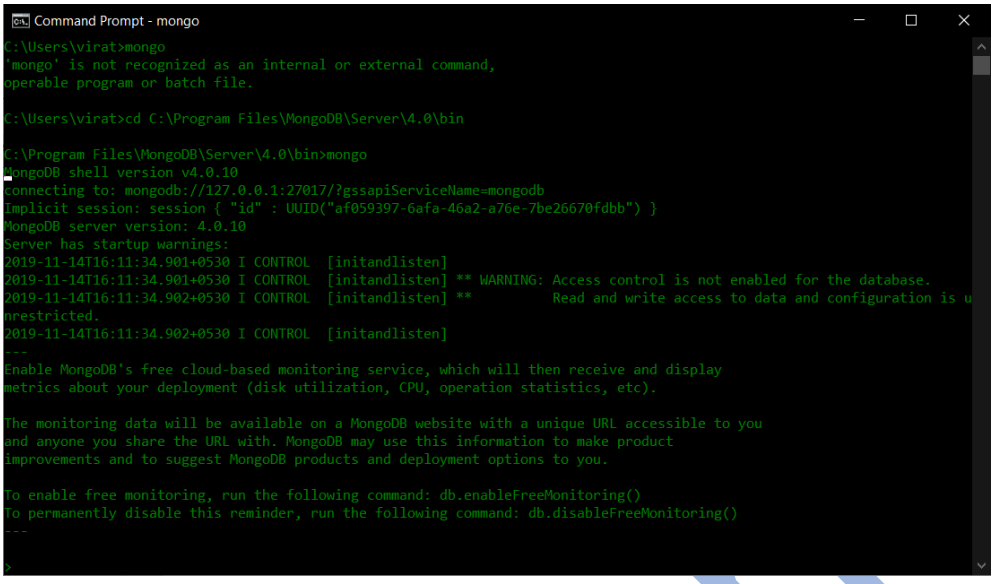


2.3 Starting MongoDB:

1. Go the services section and then start the MongoDB service if not already started.



2. Now, to check whether the database server is up or not, go to the bin directory of the MongoDB installation and run the 'mongo' command as shown. If the command runs successfully, it means that the server is up and running, and we can proceed.

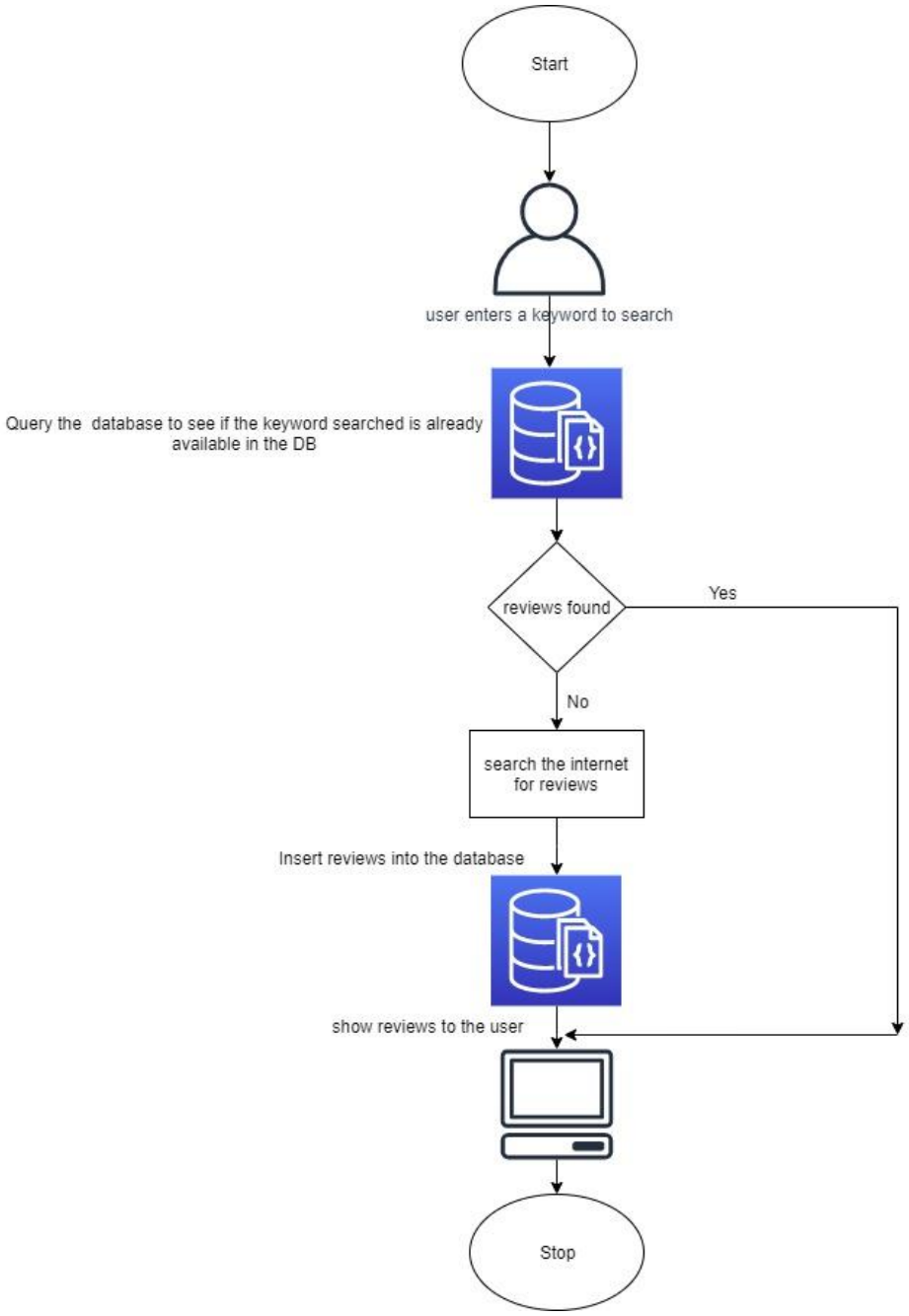

Command Prompt - mongo
C:\Users\virat>mongo
'mongo' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\virat>cd C:\Program Files\MongoDB\Server\4.0\bin
C:\Program Files\MongoDB\Server\4.0\bin>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("af059397-6afa-46a2-a76e-7be26670fdbb") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-11-14T16:11:34.901+0530 I CONTROL [initandlisten]
2019-11-14T16:11:34.901+0530 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-11-14T16:11:34.902+0530 I CONTROL [initandlisten] ** Read and write access to data and configuration is u
nrestricted.
2019-11-14T16:11:34.902+0530 I CONTROL [initandlisten]

Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

>

3. Application Architecture:

The architecture of the application is:



4. Python Implementation:

Note: I have used PyCharm as an IDE for this documentation

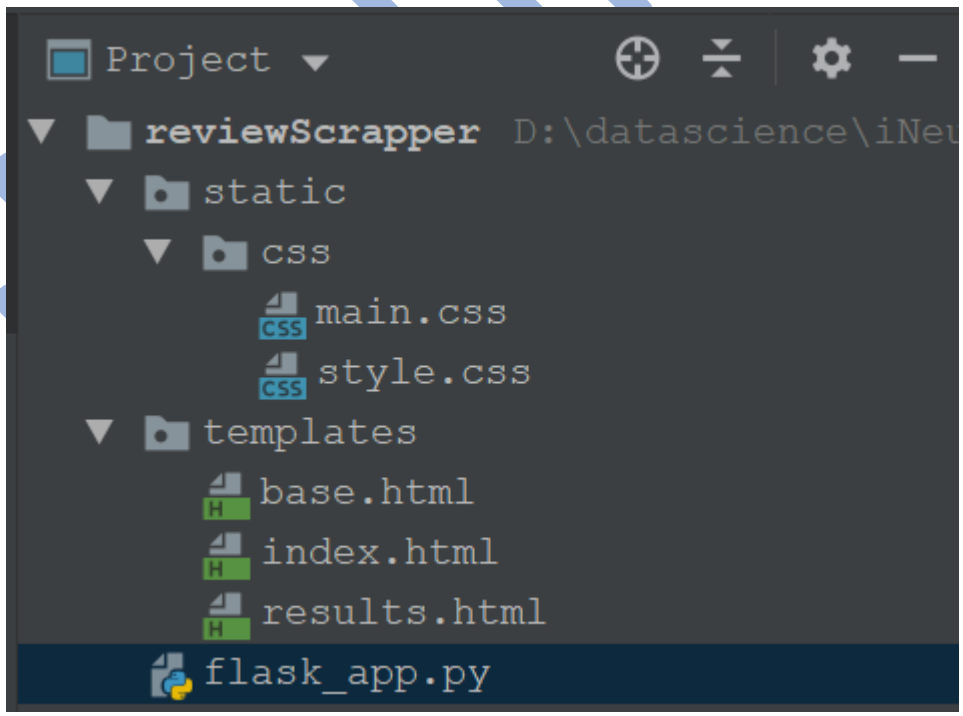
1. Let's create a folder called 'reviewScrapper' on our local machines.

2. Inside that folder, let's create two more folders called 'static' and 'templates' to hold the code for the UI of our application. Inside 'static', let's create a folder called 'css' for keeping the stylesheets for our UI.
3. Let's create a file called 'flask_app.py' inside the 'reviewScrapper' folder.
4. Inside the folder 'css', create the files: 'main.css' and 'style.css'. The files are attached here for reference.

5. Inside the folder 'templates', create three HTML files called: 'base.html','index.html', and 'results.html'. The files are attached here for reference.

- base.html→ It acts as the common building block for the other two HTML pages.
- index.html→ Home page of our application.
- results.html→ Page to show the reviews for the searched keyword.

6. Now, the folder structure should look like:



7. Now, let's understand the flow:
   a) When the application starts, the user sees the page called 'index.html'.
   b) The user enters the search keyword into the search box and presses the submit button.

c) The application now searches for reviews and shows the result on the 'results.html' page.

8. Understanding flask_app.py.

a) Import the necessary libraries:

```
from flask import Flask, render_template, request,jsonify
from flask_cors import CORS,cross_origin
import requests
from bs4 import BeautifulSoup as bs
from urllib.request import urlopen as uReq
import pymongo
```

b) Initialize the flask app

```
app = Flask(__name__)  # initialising the flask app with the name
'app'
```

c) Creating the routes to redirect the control inside the application itself. Based on the route path, the control gets transferred inside the application.

```
@app.route('/',methods=['POST','GET']) # route with allowed
methods as POST and GET
```

d) Now let's understand the 'index()' function.

i. If the HTTP request method is POST(which is defined in index.html at form submit action), then first check if the records for the searched keyword is already present in the database or not. If present, show that to the user.

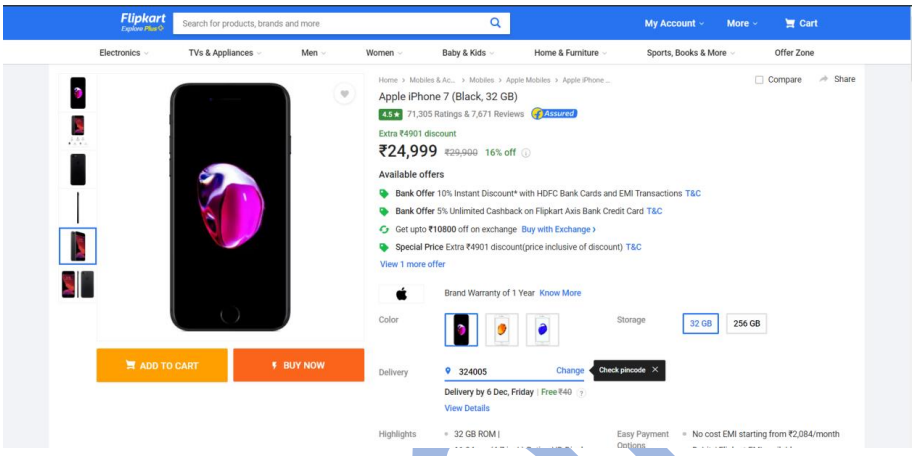
```
dbConn = pymongo.MongoClient("mongodb://localhost:27017/")
# opening a connection to Mongo
db = dbConn['crawlerDB'] # connecting to the database
called crawlerDB
reviews = db[searchString].find({}) # searching the
collection with the name same as the keyword
if reviews.count() > 0: # if there is a collection with
searched keyword and it has records in it
    return render_template('results.html',reviews=reviews)
# show the results to user
```

ii. If the searched keyword doesn't have a database entry, then the application tries to fetch the details from the internet, as shown below:

```
flipkart_url = "https://www.flipkart.com/search?q=" +
searchString # preparing the URL to search the product on
Flipkart
uClient = uReq(flipkart_url) # requesting the webpage from
the internet
flipkartPage = uClient.read() # reading the webpage
uClient.close() # closing the connection to the web server
```

```
flipkart_html = bs(flipkartPage, "html.parser") # parsing the webpage as HTML
```

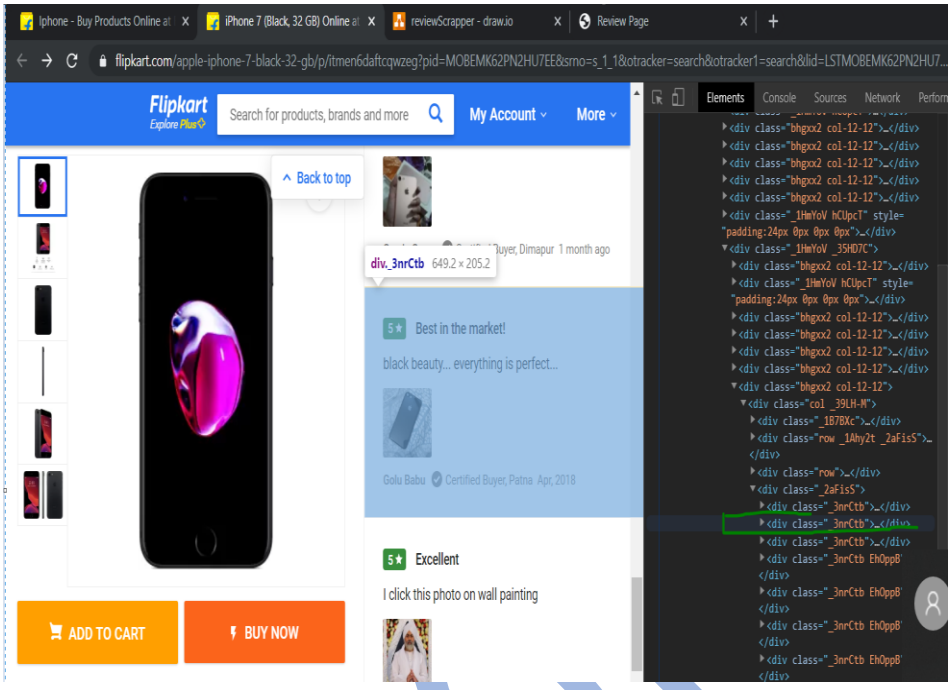
iii. Once we have the entire HTML page, we try to get the product URL and then jump to the product page. It is similar to redirecting to the following page:



The equivalent Python code is:

```
productLink = "https://www.flipkart.com" + box.div.div.div.a['href'] # extracting the actual product link
prodRes = requests.get(productLink) # getting the product page from server
prod_html = bs(prodRes.text, "html.parser") # parsing the product page as HTML
```

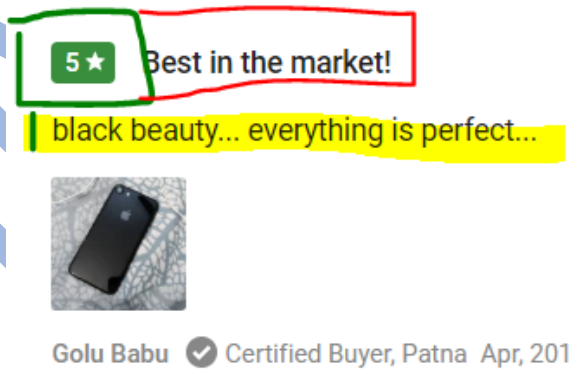
iv. On the product page, we need to find which HTML section contains the customer comments. Let's do inspect element(ctrl+shift+i) on the page first to open the element-wise view of the HTML page. There we find the tag which corresponds to the customer comments as shown below:



Python code for implementing the same is:

```
commentboxes = prod_html.find_all('div', {'class':
"_3nrCtb"}) # finding the HTML section containing the
customer comments
```

v. Once we have the list of all the comments, we now shall extract the customer name(in grey), the rating(in green), comment heading(marked in red), and the customer comment(highlighted in yellow) from the tag.



The Python code for the same is:

```
reviews = [] # initializing an empty list for reviews
#  iterating over the comment section to get the details of
the customer and their comments
for commentbox in commentboxes:
```

```
    try:
        name = commentbox.div.div.find_all('p', {'class':
'_3LYOAd _3sxSiS'})[0].text

    except:
        name = 'No Name'

    try:
        rating = commentbox.div.div.div.div.text

    except:
        rating = 'No Rating'

    try:
        commentHead = commentbox.div.div.div.p.text
    except:
        commentHead = 'No Comment Heading'
    try:
        comtag = commentbox.div.div.find_all('div',
{'class': ''})
        custComment = comtag[0].div.text
    except:
        custComment = 'No Customer Comment'
```

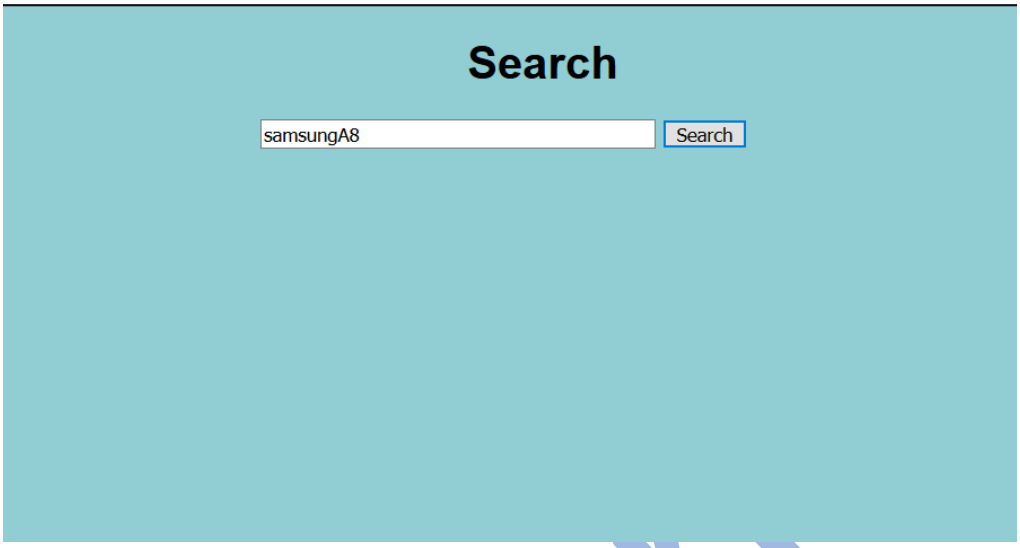
If you notice, the parsing is done using the try-except blocks. It is done to handle the exception cases. If there is an exception in parsing the tag, we'll insert a default string in that place.

vi. Once we have the details, we'll insert them into MongoDB. After that, we'll return the 'results.html' page as the response to the user containing all the reviews. The python code for that is:

```
mydict = {"Product": searchString, "Name": name, "Rating":
rating, "CommentHead": commentHead,
              "Comment": custComment} # saving that detail
to a dictionary
    x = table.insert_one(mydict) #insertig the dictionary
containing the rview comments to the collection
    reviews.append(mydict) #  appending the comments to the
review list
return render_template('results.html', reviews=reviews) #
showing the review to the user
```

e) After this, we'll just run our python app on our local system, and it'll start scraping for reviews as shown below:

Home Page:



Search Results:

REVIEWS

Product	Name	Rating	Comment Heading	Comments
samsungA8	Soumya Guha Roy	4	Beauty & the beast combined.	I have been using the device since last 3 days. Here is my review Pros: 1. Excellent display 2. Fast focus and fast capture camera 3. Nice UI 4. Decent battery backup (after 24 hours, battery remains ~30% with medium usage. 5. Good performance. 6. Unread notification count is spot on, like we have in iOS. 7. The metallic design is superb, and feels premium. Go for the gold one, than the white. The white one looks plast...
samsungA8	deepak pagar nashik	5	superb ,wesome,stylish, powerful ,metal finishing with complete security handset by samsung	i bought this stunning metal body slim phone before 15 days ,and i m really happy to use it ,this cell having 32 gb internal memory due to this not i used my memory card . its hybrid sim phone but no problem because u have more internal capacity . Samsung Galaxy A8 is a awesome device again from Samsung. A8 could have had Dual SIM along with SD card (u can

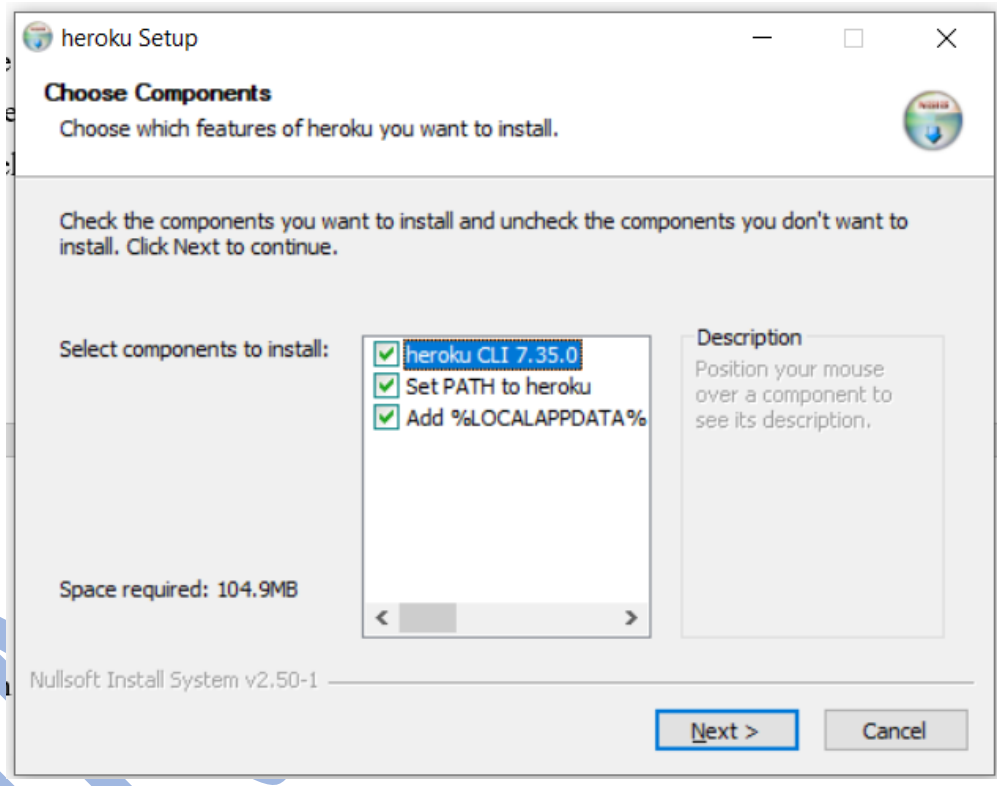
5. Heroku:

The Python app that we have developed is residing on our local machine. But to make it available to end-users, we need to deploy it to either an on-premise server or to a cloud service. Heroku is one such cloud service provider. It is free to use(till 5 applications).

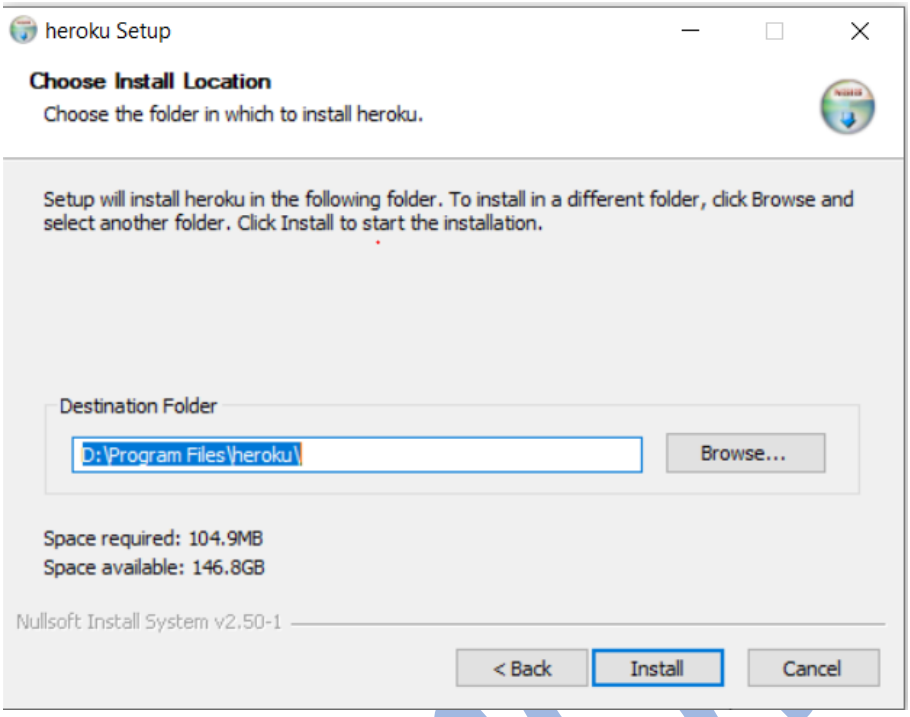
We'll deploy this application to the Heroku cloud, and then anybody with the URL can then consume our app.

6. Heroku Basics:

- We'll first go to heroku.com, and we'll create a new account if we already don't have one.
- We'll download and install the Heroku CLI from the Heroku website: https://devcenter.heroku.com/articles/heroku-cli.
- Double-click the installation file and the following window shall appear:



- Click on next and select the installation directory for the CLI.



- Click on install to complete the installation.

7. Steps before cloud deployment:

We need to change our code a bit so that it works unhindered on the cloud, as well.

a) Add a file called 'gitignore' inside the 'reviewScrapper' folder. This folder contains the list of the files which we don't want to include in the git repository. My gitignore file looks like:

```
.idea
```

As I am using PyCharm as an IDE, and it's provided by the Intellij Idea community, it automatically adds the .idea folder containing some metadata. We need not include them in our cloud app.

b) Add a file called 'Procfile' inside the 'reviewScrapper' folder. This folder contains the command to run the flask application once deployed on the server:

```
web: gunicorn app:app
```

Here, the keyword 'web' specifies that the application is a web application. And the part 'app:app' instructs the program to look for a flask application called 'app' inside the 'app.py' file. Gunicorn is a Web Server Gateway Interface (WSGI) HTTP server for Python.

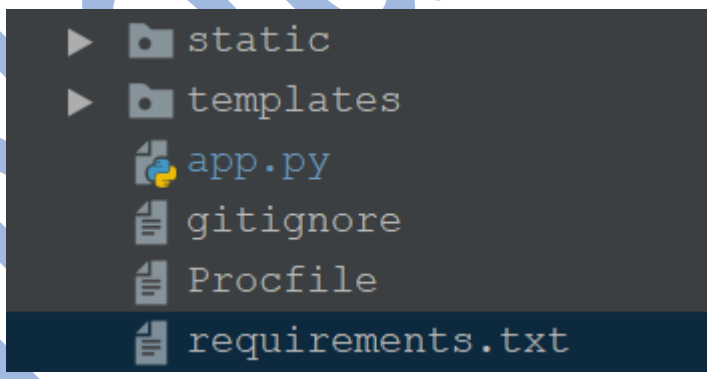
c) Open a command prompt window and navigate to your 'reviewScrapper' folder. Enter the command 'pip freeze > requirements.txt'. This command generates the 'requirements.txt' file. My requirements.txt looks like:

```
beautifulsoup4==4.8.1
bs4==0.0.1
certifi==2019.9.11
Click==7.0
Flask==1.1.1
Flask-Cors==3.0.8
gunicorn==20.0.4
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
numpy==1.17.4
opencv-python==4.1.2.30
Pillow==6.2.1
pymongo==3.9.0
requests==2.21.0
requests-oauthlib==1.2.0
six==1.13.0
soupsieve==1.9.5
Werkzeug==0.16.0
```

requirements.txt helps the Heroku cloud app to install all the dependencies before starting the webserver.

d) We have created a new file 'app.py' inside the review scrapper folder:

i. Remove the first_flask.app file from the directory. Resulting folder structure:



ii. A default route has been added to the app.py file to direct to the home page when the application is initially invoked as shown below:

```
@app.route('/',methods=['GET'])  # route to display the home
page
@cross_origin()
def homePage():
    return render_template("index.html")
```

iii. We have removed the part where we were writing to MongoDB. Consuming MongoDB might incur charges. So, we have removed that part.

iv.

8. Heroku app creation and deployment

b. After installing the Heroku CLI, Open a command prompt window and navigate to your 'reviewScrapper' folder.

c. Type the command `'heroku login'` to login to your heroku account as shown below:

```
D:\datascience\iNeuron\docs\reviewScrapper>heroku login
heroku: Press any key to open up the browser to login or q to exit:
```

d. After logging in to Heroku, enter the command `'heroku create'` to create a heroku app. It will give you the URL of your Heroku app after successful creation.

e. Before deploying the code to the Heroku cloud, we need to commit the changes to the local git repository.

f. Type the command `'git init` to initialize a local git repository as shown below:

```
D:\datascience\iNeuron\docs\reviewScrapper>git init
```

g. Enter the command `'git status'` to see the uncommitted changes

h. Enter the command '`git add .`' to add the uncommitted changes to the local repository.

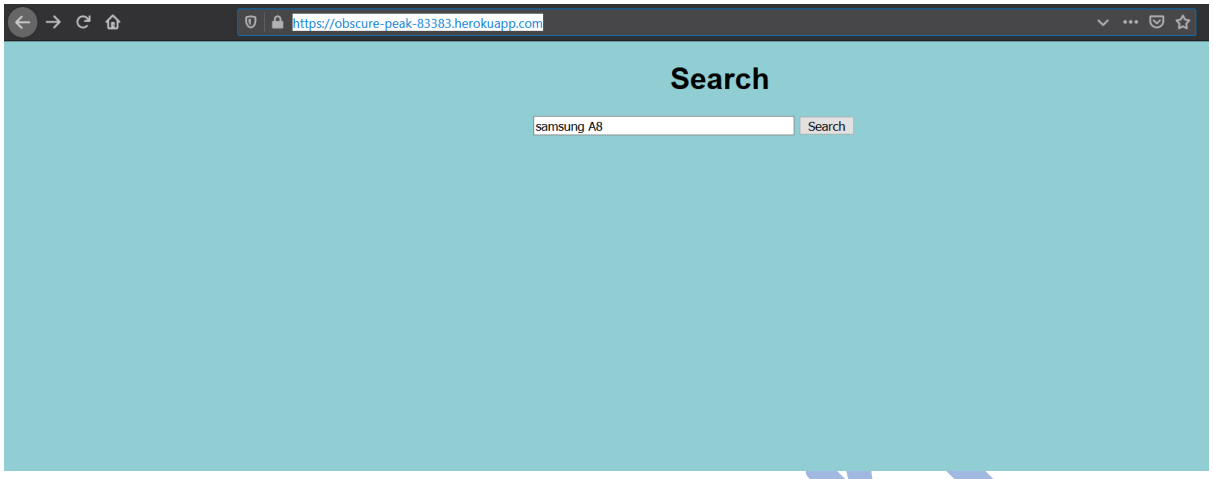
i. Enter the command '`git commit -am "make it better"`' to commit the changes to the local repository.

j. Enter the command '`git push heroku master`' to push the code to the heroku cloud.

k. After deployment, heroku gives you the URL to hit the web API.

l. Once your application is deployed successfully, enter the command '`heroku logs --tail`' to see the logs.

Final Result:



Thank You!