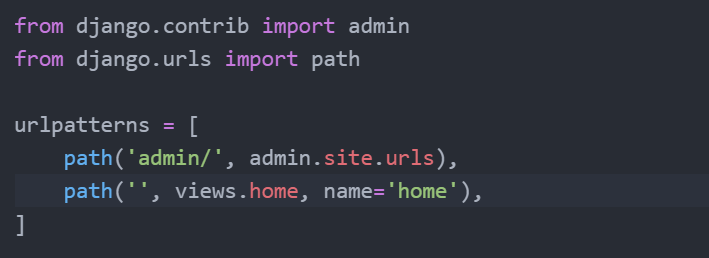
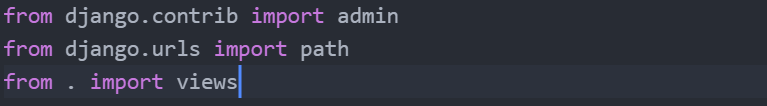
**Great Kart E-Commerce Project**

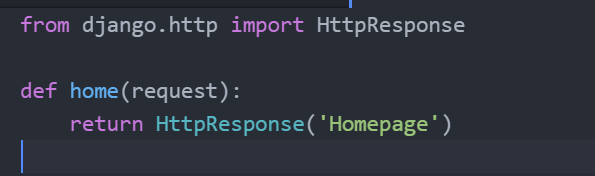
* Here we are using **atom ide** for writing the code and **git** bash for running the commands……. You can also use other editors also like vs code, Py charm etc.
* Now we will start the project by creating a virtual environment 🡪 because we want to develop the application inside a dedicated space inside our system 🡪 If we don’t install than it will going to mess up with the list of packages installed in our global scope of the computer.
* **Creating and Activating the Virtual Environment:**
* Now create a folder on your desktop or any other folder and rename it with your project name (EG: Great Kart) 🡪 Now open that folder and right click on the mouse and select “Git Bash Here” option ( go to web browser and type – install virtual environment python 🡪 there you will get list of commands i.e., actually python with version 3.4 or above gets virtual env already installed) 🡪 Now type ***“python -m venv env”*** [env – name of your virtual env, it can be any name of your wish] with this **virtual env is created** which can be seen inside our Great Kart folder
* We have created a virtual env but still outside of it. So, we need **to activate our virtual env** that is created 🡪 So type ***“source env/Scripts/activate”*** command for Git Bash users (with this command we are actually entering into env folder & then into Scripts folder & then activating the virtual env. With this we have entered inside the virtual env.
* To **get out of the virtual env**, use this command ***“deactivate”***
* To **check out the installed packages** on your system ***“pip freeze”***
* For mac book users – ***“source env/bin/activate”***
* **Django Installation:**
* To install Django, Open Git Bash and enter “Pip install django” (for latest version), but here we are using 3.1 version, so type “***pip install django==3.1***”. [make sure v env is activated]
* To check out installed packages on the v env ***“pip*** freeze***”***
* Now it is the time to create a Django project – ***“django-admin startproject greatkart .”*** (great kart is the project name and . helps in creating the project in the current directory. (if we don’t put the . it will create multiple folders) 🡪 Now the greatkart folder and manage.py is created inside your project folder.
* Now we are good to run django basic command 🡪 Python manage.py runserver 🡪 use the https – “127.0.0.1:8000” 🡪 which displays successful installation of our django on browser.
* **Run Http Response:**
* We’ll remove default django page and insert customised http response,
* Now we will open our project folder inside the Atom IDE
* The db.sqlite3 file is automatically generated when we run the run-server command.
* 1st we need to create a urls pattern in the urls.py file [path(‘’,views.home, name=’home’)]



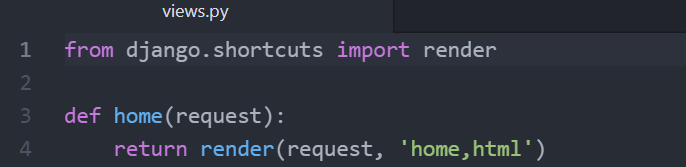
* As we don’t have any views folder here because it is the project level folder 🡪 So we need to manually create views.py file [Select greatkart folder – right click – new file – views.py 🡪 Now views.py file will be created inside the project folder greatkart.
* We have to import views file in the urls.py file 🡪 [ from .import views



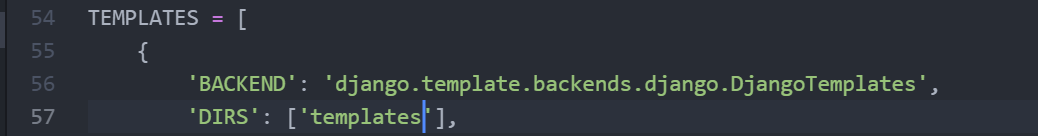
* In **Views.py**, we need to create home function



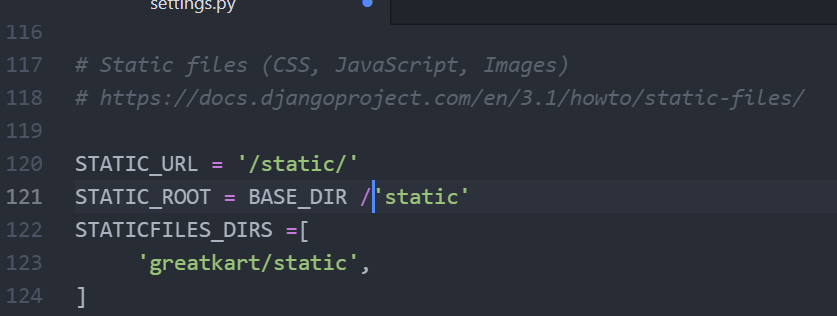
* Now run the server in the git bash & it changes the display page in the browser.
* **Configuring Template:**
* A ready-made bootstrap template is provided in the resource section 🡪 download it and extract the files 🡪 now open the extracted file in a new code editor like VS code, Notepad++, Sublime etc.,
* **Note:** Do not open the extracted bootstrap file inside the default code editor that you are using (In this case Atom is the default editor in usage)
* Now we need to configure the django templates
* Inside views.py file we have written “return HttpResponse(‘Homepage’)” which has been manually edited and is not a good practice in using templates 🡪 because in templates we are having a lot of html tags, css and JS files. Which makes impossible to use all those templates in http response
* So, in **Views.py** file instead of HttpResponse we use render function



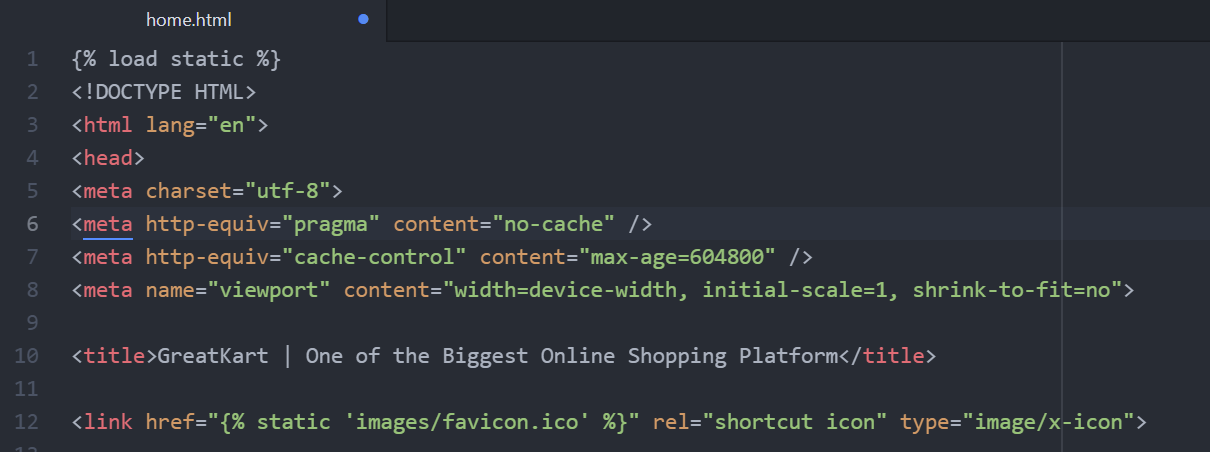
* Next thing is we need to create templates folder [right click on GreatKart parent folder – select new folder and name it as ‘templates.’
* Now inside the templates folder created 🡪 we need to create the html file (right click on templates folder – select ‘new file’ – name it as “home.html”.
* Now we need to tell the settings file (greatkart 🡪 settings.py) that we have created & using templates in this project 🡪 So open Settings.py file & enter ‘templates’ in DIRS



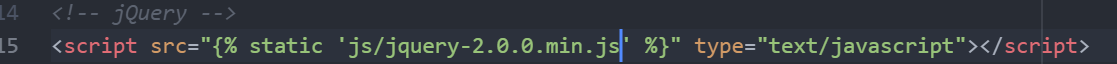
* Now provide some header text in the home.html like, [<h2> This is template </h2>] and run the server.
* **Implementing Bootstrap Template:**
* So far we have successfully run our Django template, Now instead of text message we’re going to use our real template 🡪 Go to text editor in which you opened bootstrap template 🡪 Go to index.html file (copy the whole code) and paste it in the “home.html” file – save it and refresh the browser – which displays basic html interface.
* Now we need to configure the static files in Django 🡪 to do that go to project level folder ‘greatkart’ – right click – new folder – “static” ( Inside this static file we need to bring all the CSS, fonts, images, js folders from bootstrap into static folder) 🡪 To do this go to bootstrap template folder that you downloaded and copy the four folders & now go to the greatkart project folder – inside that open static folder – and paste the four folders there, which will display in your Atom code editor too.
* Now go to ‘settings.py’ file and under STATIC\_URL you need to provide STATIC\_ROOT = BASE\_DIR ‘/static’ 🡪 which mean we are referring to project folder “greatkart” and the “static” folder inside it. The BASE\_DIR is already is already included in the Django settings.py file (line-16)



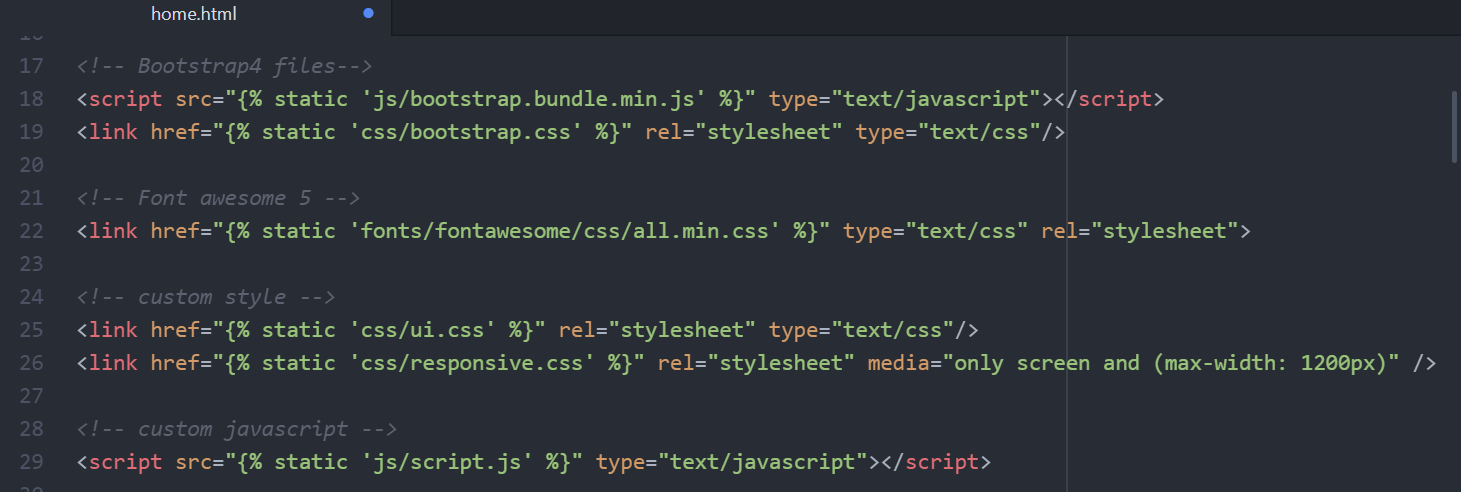
* Once this is done, we need to run a command called – “collect static”. This command collects all the files that we pasted inside static folder - & it will create a new static folder inside the root folder which will be used for the website.
* For this we need to run collect static command in the git bash 🡪 ***“python mange.py collectstatic”***
* Now a new “static” folder is created inside the root folder 🡪 Inside which a new folder “admin” was also created.
* Once this is done, we need to load the static files inside the html file.
* Go to ‘home.html’ file (line-11) cut the hyperlink [<link href=” images/favicon.ico”] 🡪 <link href =” {% static ‘images/favicon.ico’ %}” and also at top (line-1) we need to load static like – [ {% load static %}]



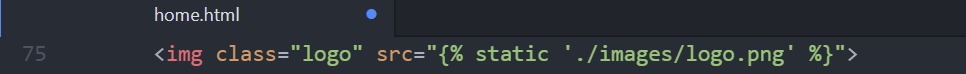
* Now if we reload the page, then the logo appears at the top left of the page (which mean the static file is working).
* similar to it we need to do for all remaining files
* 1st we need to make the js files working



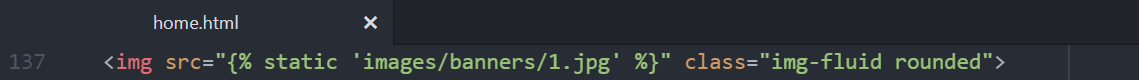
* Then, modify the bootstrap4, font awesome 5, custom style, custom java script files and make it working 🡪 then reload page once saved.



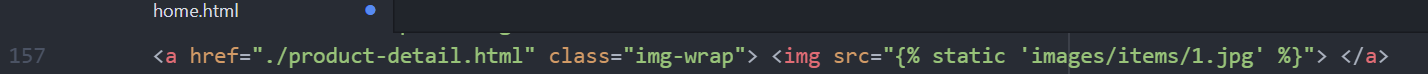
* Now we need to make the static images working
* 1st make the logo of the brand work



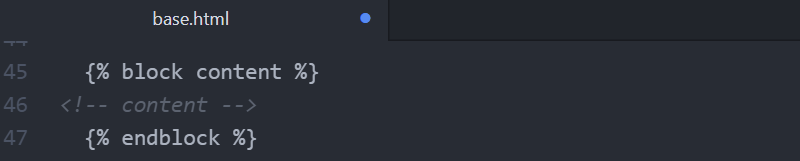
* 2nd make the banner image work



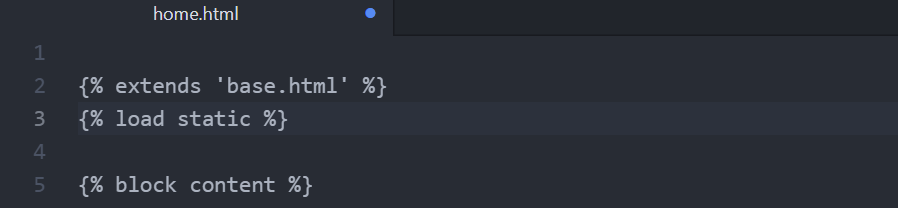
* 3rd make the image 1 working

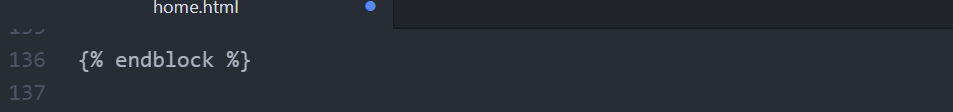


* 4th similar to the image 1, make all the remaining images working.
* Now refresh the page, which makes all product images displayed.
* **Base Template:**
* Now we need to create a base template 🡪 For this go to ‘templates’ folder in the Atom – right click – create new file “base.html” 🡪 once it is created go to ‘home.html’ – cut the entire body portion from it (i.e., where body tag ends - line 1 to 43) and paste it in the ‘base.html’🡪 Thereafter add “block content” & “endblock” in the base.html

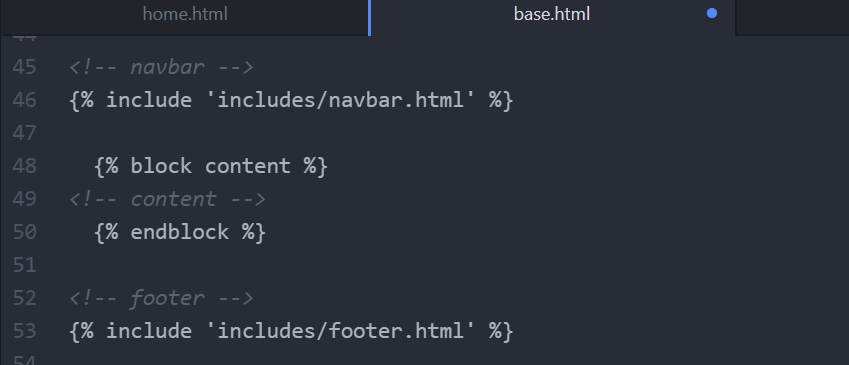


* Now go back to ‘home.html’ and cut the header portion [ <header> to </header> ] and paste it in b/w </body> & {% block content %} – because always comes just above the content.
* Thereafter we need to “extend the base template” from base.html to home.html 🡪 So the command to extend the base template is as follows. You make sure that end block is mentioned at the last.

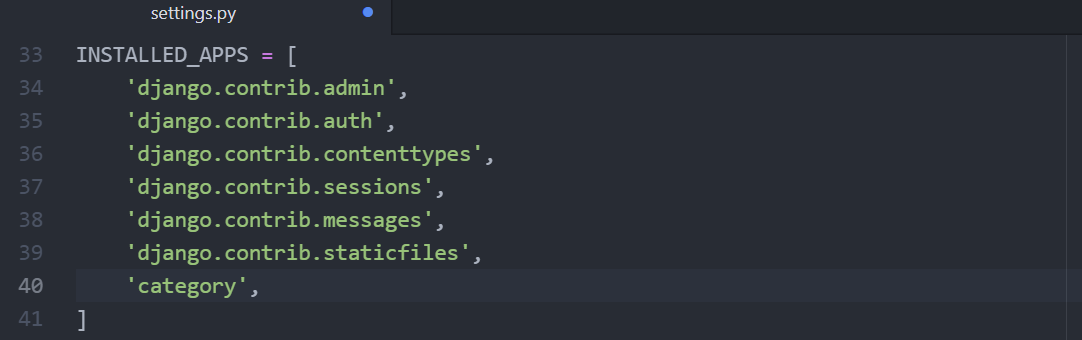




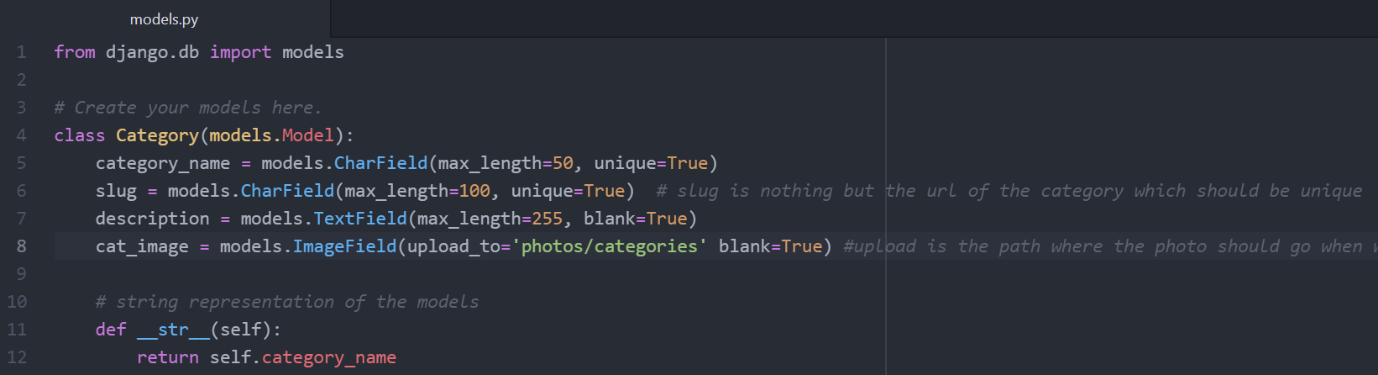
* Now refresh the page & you can able to see the header part visible which is loaded from base template and the body part from home.html.
* After this we go to ‘home.html’ which contains ‘footer’ portion which is common to all the pages 🡪 So we’ll create a new folder inside the ‘templates’ folder called “includes” folder 🡪 Inside the ‘includes’ folder we’ll create a new file called “footer.html” and we’ll also create another file called “navbar.html” inside ‘includes’ folder. (These 2 files are common in all pages).
* Now go to ‘home.html’ cut the footer portion code [<footer> to </footer>] and paste it inside the ‘footer.html’.
* Then go to ‘base.html’ cut the header portion code [<header> to </header>] and paste it inside the ‘navbar.html’.
* Now we need to load the navbar and footer inside the ‘base.html’



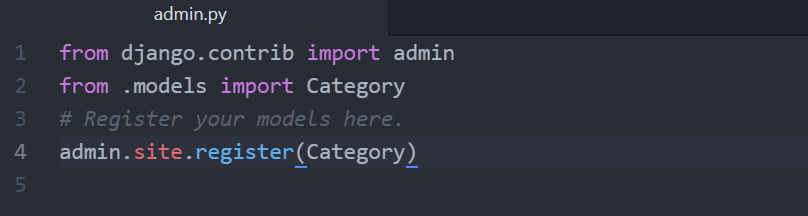
* At the top of ‘navbar.html’ page include the {% load static %} command – save & refresh.
* **Free Practice Templates:**
* Download the practice templates from the resource section and you can use those templates for your practice purpose.
* **Category Model:**
* Let us now create a app – it is obvious for e-commerce site to create certain apps that are like category, product, order, store app etc.,
* 1st & foremost we’ll create a category app
* For this go to ‘git bash’ and give the command “python manage.py startapp category” [Here category is the name of the app].
* As soon as you create the app make sure to register the app in “settings.py” file



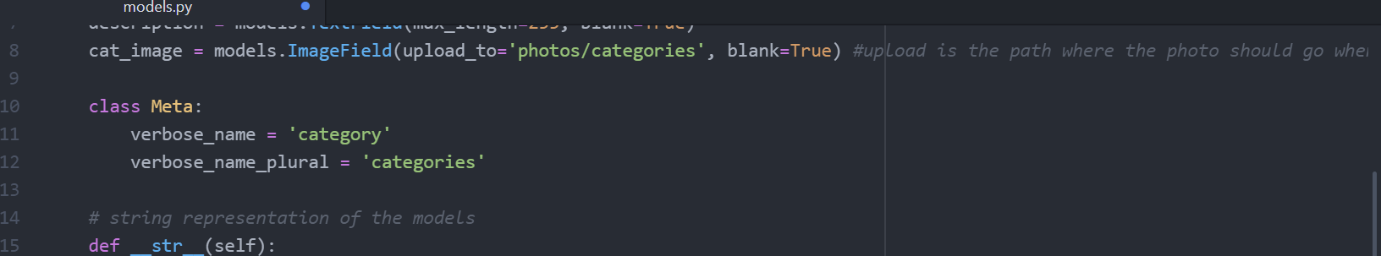
* Now let’s create a model - for the category model we’ll be having 3-4 fields (category name, slug, category image, description.) 🡪 Go to “models.py” file



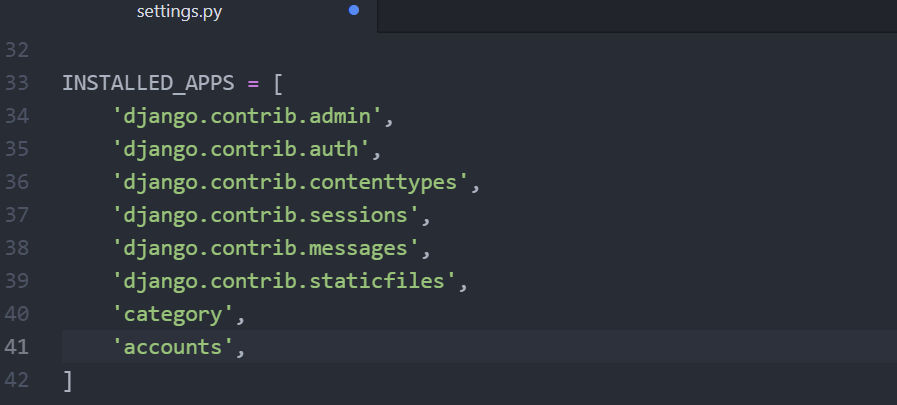
* Now let’s go to the “admin.py” file – we want that model inside the admin panel (register the model here to admin file)



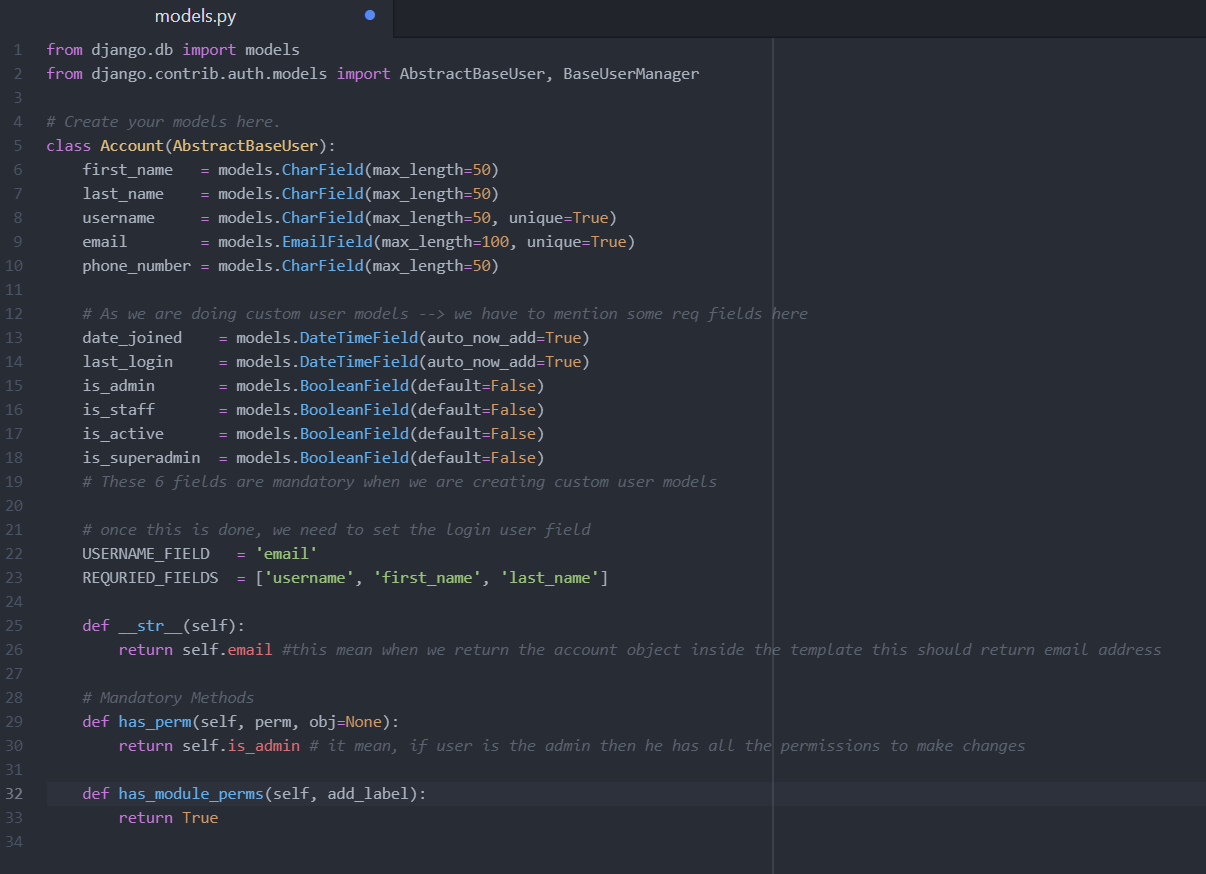
* The next step is to make the migrations – go to ‘git bash’ window and type the command “python manage.py makemigrations” 🡪 which creates “0001\_initial.py” file inside the migrations folder.
* We have to run this migration file – in order to do that type “python manage.py migrate” – which migrates all the files
* Now we need to ‘run the server’ in the git bash window 🡪 go to browser and enter the admin site “127.0.0.1:8000/admin” 🡪 requires to enter ‘username’ & ‘password’, So to enter into this admin panel we need to ‘create a superuser’
* To create a superuser – go to ‘git bash’ and type the command – “winpty python manage.py createsuperuser” (Note: python manage.py createsuperuser – command alone don’t work in the git bash) 🡪 Thereafter provide the ‘username’, ‘email address’ & ‘password [t@78]’ 🡪 with this superuser is created successfully.
* Now again run the server – login into the admin panel.
* **Note**: In the Django Administration Panel – it makes the Django model in plural form (EG: categorys) 🡪 To correct this go to ‘models.py’ file & customize to display the proper Django model name (line-10 to 12) – run server & check admin panel.



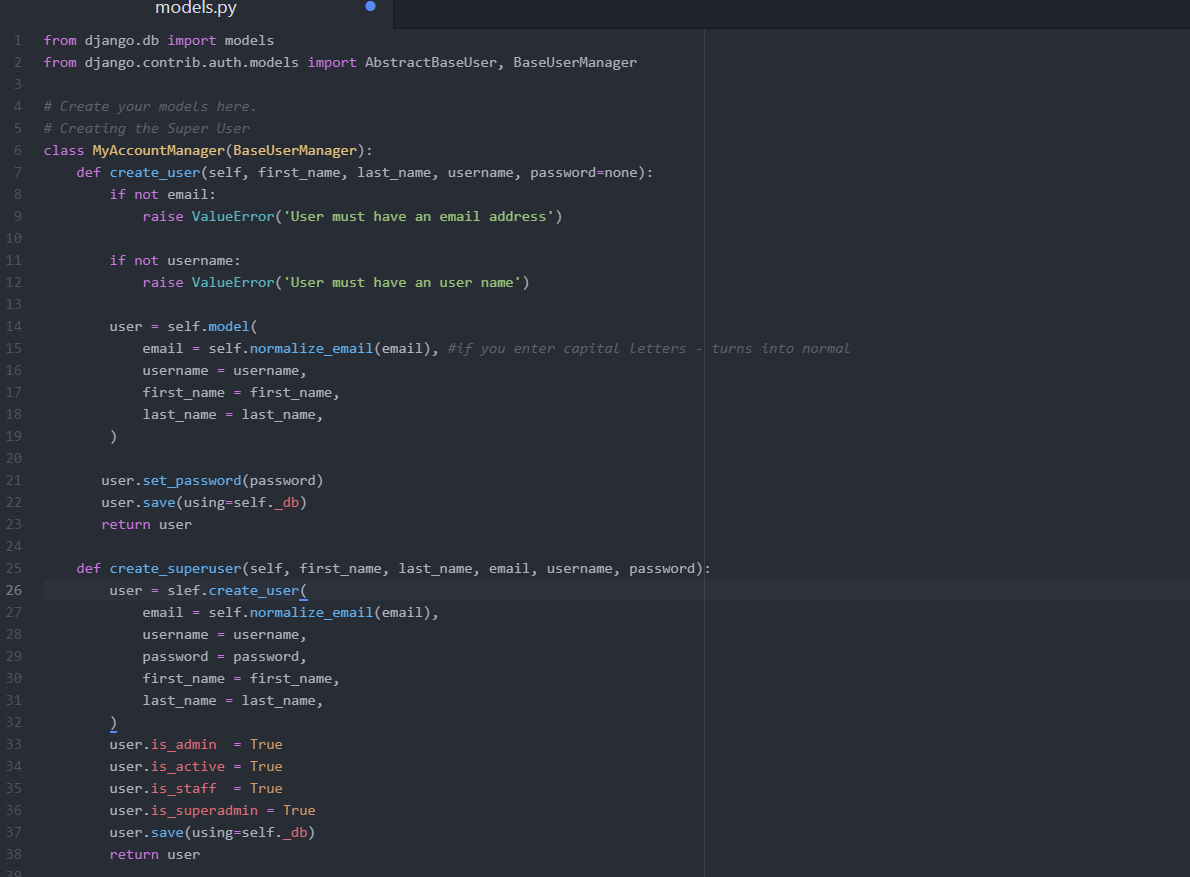
* **Brief About Custom User Model:**
* To login into Django admin panel, we used – Username & Password, which we don’t use in present trends. Instead, we use Email & Password 🡪 so in order to change this we need to create a **custom user model** to bypass or override the default Django Authentication system.
* **Custom User Model:**
* To create a custom user model, 1st we need to create an app (accounts app)
* Go to ‘git bash’ - & type the command “python manage.py startapp accounts”
* Thereafter go to ‘settings.py’ & register the app in the INSTALLED\_APPS section - ‘accounts’



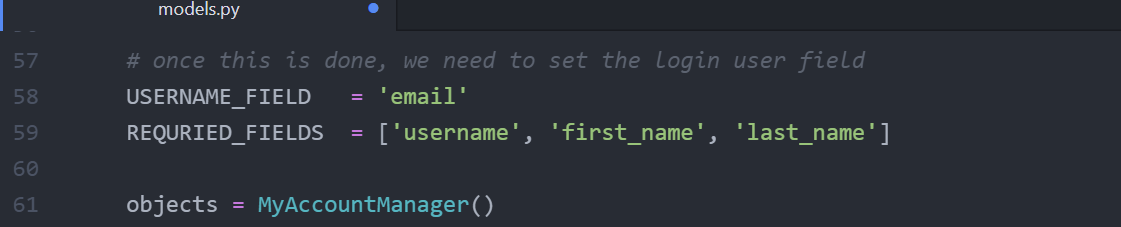
* Now go to ‘accounts’ folder – ‘models.py’ file 🡪 Here we’re going to create an ‘account model’ as well as ‘account manager’ – which is used to handle custom user model



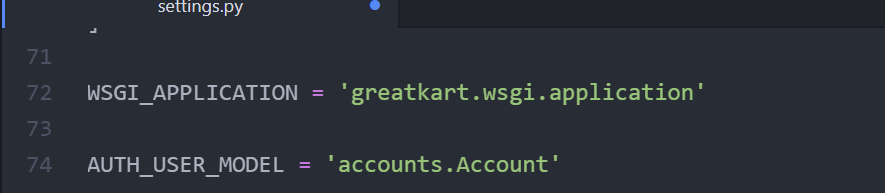
* So far, we have created account model, now we need to ‘create a model for Super Admin’



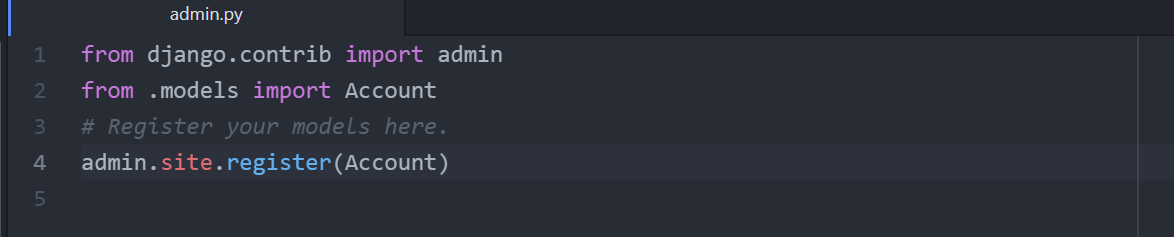
* And also don’t forget to mention the objects = MyAccountManager()



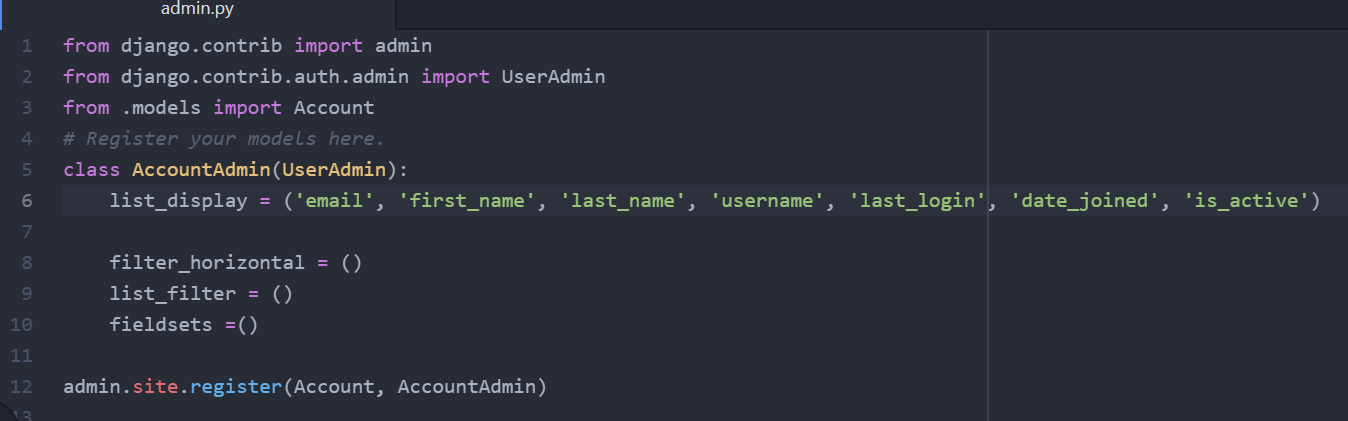
* Thereafter go to ‘settings.py’ – and tell the settings that we are going to use custom user model.



* Now we need to go to ‘admin.py’ file of the ‘accounts’ folder – to register the model.



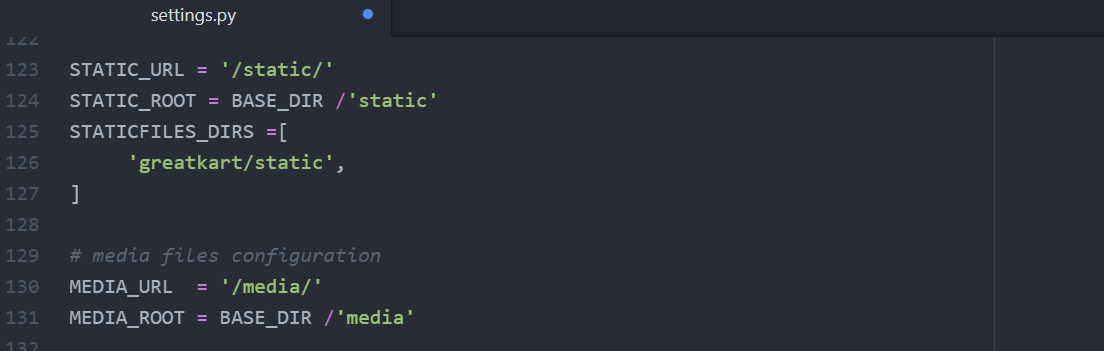
* Before we migrate the changes, we 1st need to “delete existing database” (because this database already contains our category & before whatever we’re doing – old data 🡪 For this delete the ‘dbsqlite3’ database file present in your project.
* In the next step, we have ‘migration’ folder inside the ‘category’ app – delete the “0001\_initial.py” & “0002\_auto\_20201005\_0245.py” files.
* Now we run the server – even though we know it throws error, because we want to make sure our Django will create a new blank database.
* Now apply ‘make migrations’ command and then ‘migrate’ command & ‘run server’ also. 🡪 In the browser you can notice ‘email’ in the ‘username’.
* Now it is good time to “create the super user” – type command “winpty python manage.py createsuperuser” – Email, user name, first name, last name, password (@78). 🡪 Now again run the server. – The custom user model is successful.
* **Admin Modification for Read Only Password Final:**
* In this lecture we’ll modify our model little bit so that – the password inside the Accounts won’t be editable any more.
* So, 1st we go to ‘admin.py’ file of ‘accounts’ folder.



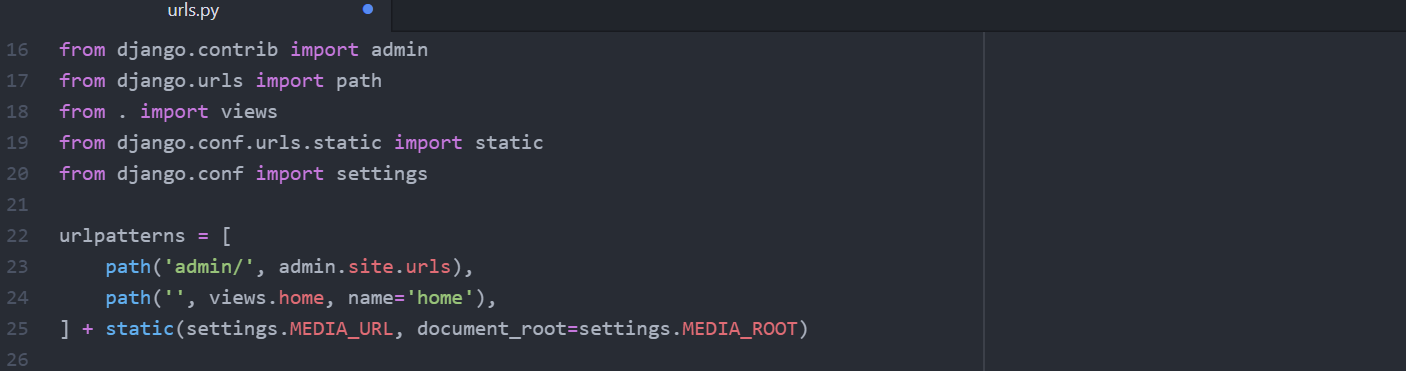
* The above changes make the password read only. Now we’ll make the first name & last name (an active hyperlink) to redirect to account details section as similar to email.



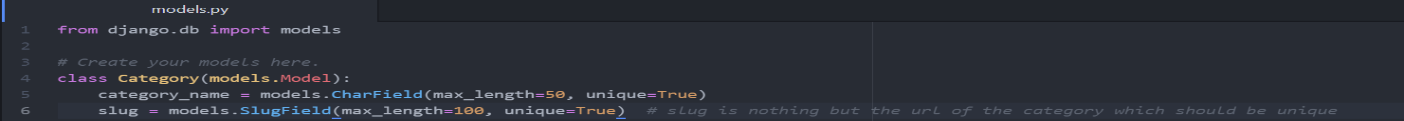
* **Configuring Django Media Files:**
* Next thing is we need to make changes to “add some categories”. Before doing that we need to “configure Django media files” similar as we did to the static files, because it is to make the images working that we upload.
* 1st go to ‘settings.py’ file – add media configuration



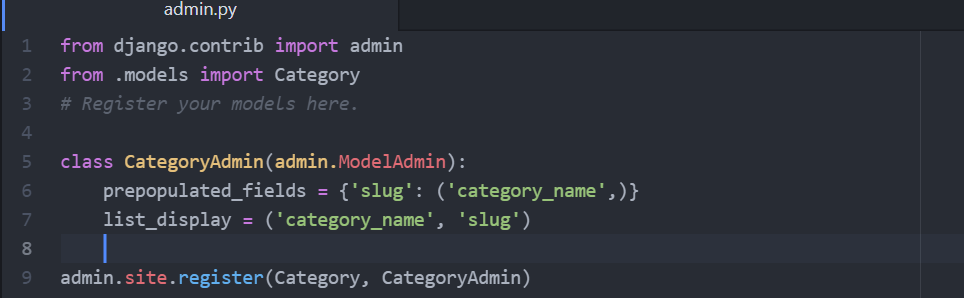
* Thereafter go to ‘urls.py’ file of the ‘greatkart’ folder



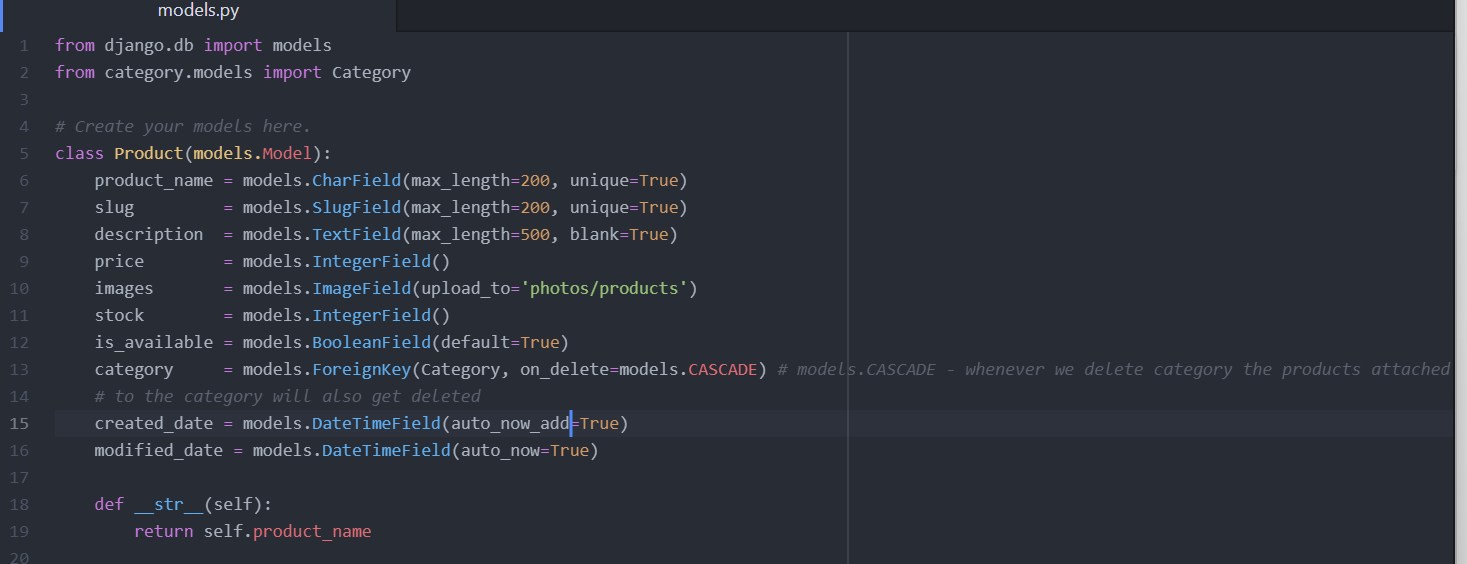
* Now we can able to add the categories in the admin panel 🡪 now go to the resource section of the video where you can get the category image & product image for the reference in a zip file 🡪 extract the files.
* Now we can add the category in the admin section – by providing name, slug, description, upload the image 🡪 save the changes (But the problem here is we need to enter the slug manually which is not recommended, so in next lecture we’ll make the slug generate automatically)
* **Pre-populate the Category Slug:**
* Now to fix the slug – 1st go to “models.py” file of “category” folder and modify the slug field



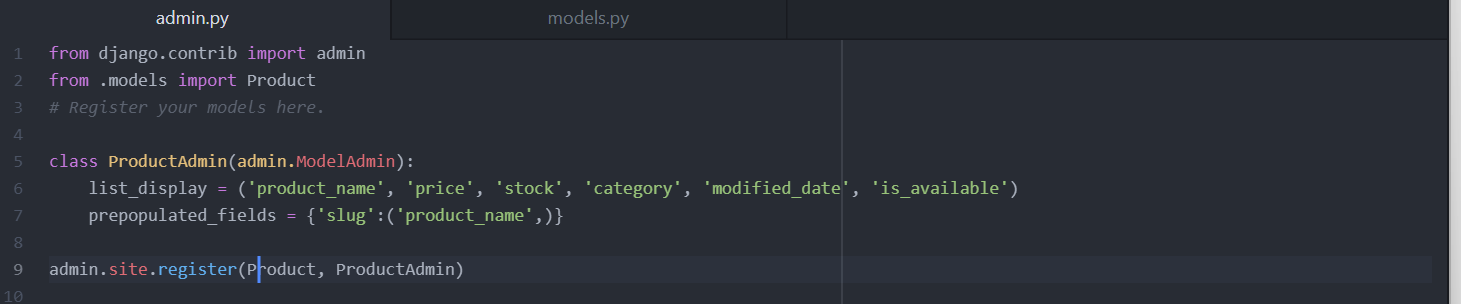
* Now apply ‘makemigrations’ & ‘migrate’ command in the git bash (It is mandatory to apply these changes whenever we make any changes in the models.py file).
* Now go to ‘admin.py’ file of the ‘category’ folder & create a new class called “CategoryAdmin”



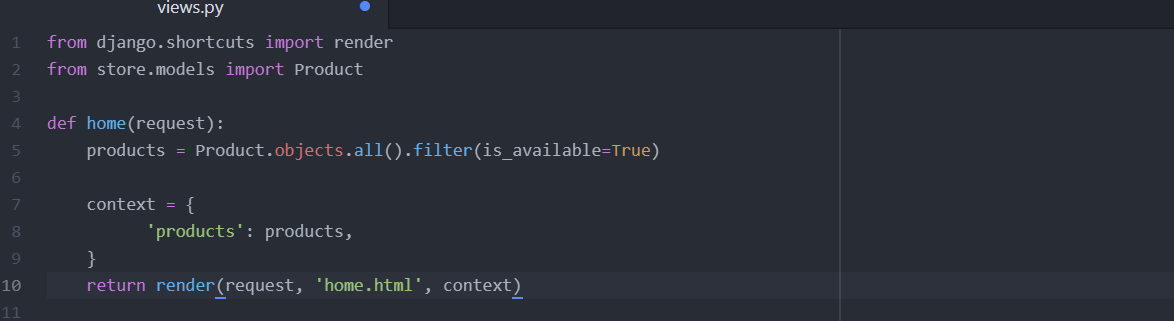
* Now the slug gets pre-populated & add the remaining all categories with particular images of the category 🡪 All these data is saved in “SQLite 3” which comes by default Django installation. The data inside ‘sqlite3’ is not in readable form.
* **Install Sqlite Studio:**
* We’ll download and Install SQ lite database, to see the data inside the database. i.e., to make database readable form. 🡪 Go to ‘Google’ & type ‘sq lite studio’ – download it and install it in the ‘Program files’ of windows c.
* After installing it successfully – open the ‘sqlite’ and there you will find the ‘database’ option on top left menu bar – select ‘add a database’ 🡪 There select the file “db.sqlite3” from the ‘greatkart’ project folder and click ok.
* Now “db (SQLite 3)” database is created inside the SQ Lite Studio 🡪 right click on the database & select “connect to the database” – it will open all the tables in the database.
* Double click on ‘category\_category’ – displays categories that we created and it also contains the data that we saved for each category.
* **Created Store App Production Model:**
* So far, we have added categories, now it’s time to add ‘products’ 🡪 In order to create the products we need to create a separate app called “Store App” – makes easy to manage the product & create the product model.
* So firstly, go to ‘git bash’ activate the server & type the command – “python manage.py startapp store” 🡪 This creates store app inside our project.
* Now go to ‘settings.py’ file & register the app in the ‘Installed Apps’ section.
* Next open ‘models.py’ file of the ‘store’ folder



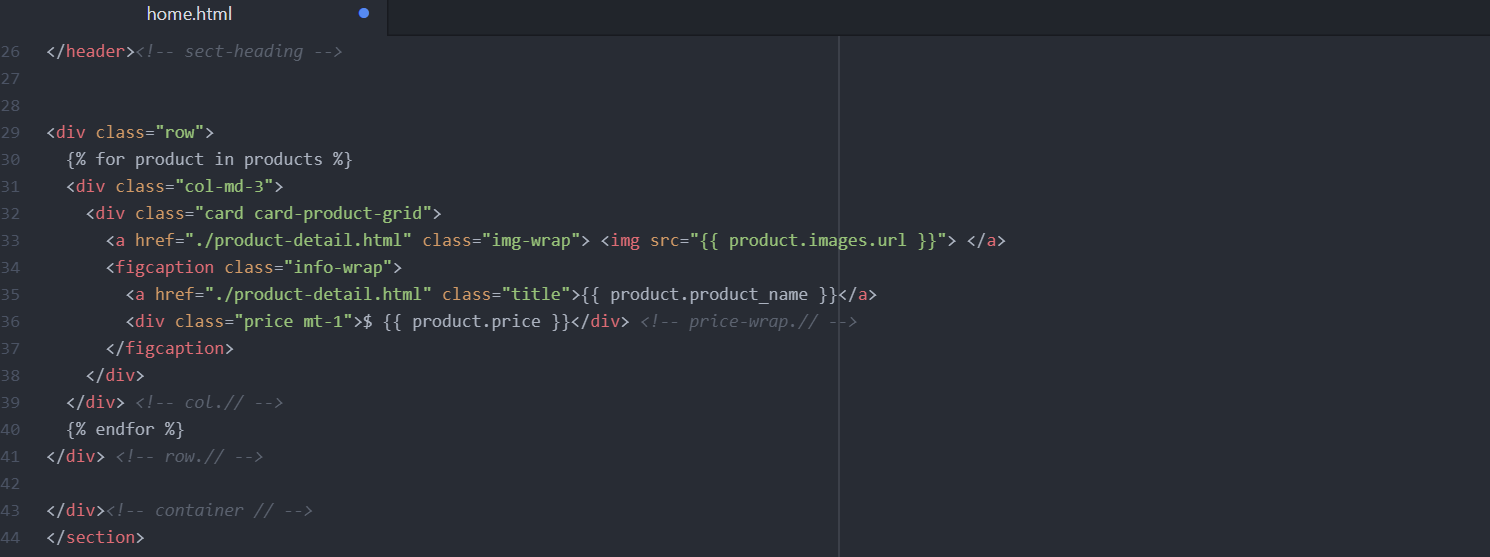
* In the next step go to ‘admin.py’ file in the ‘store app’



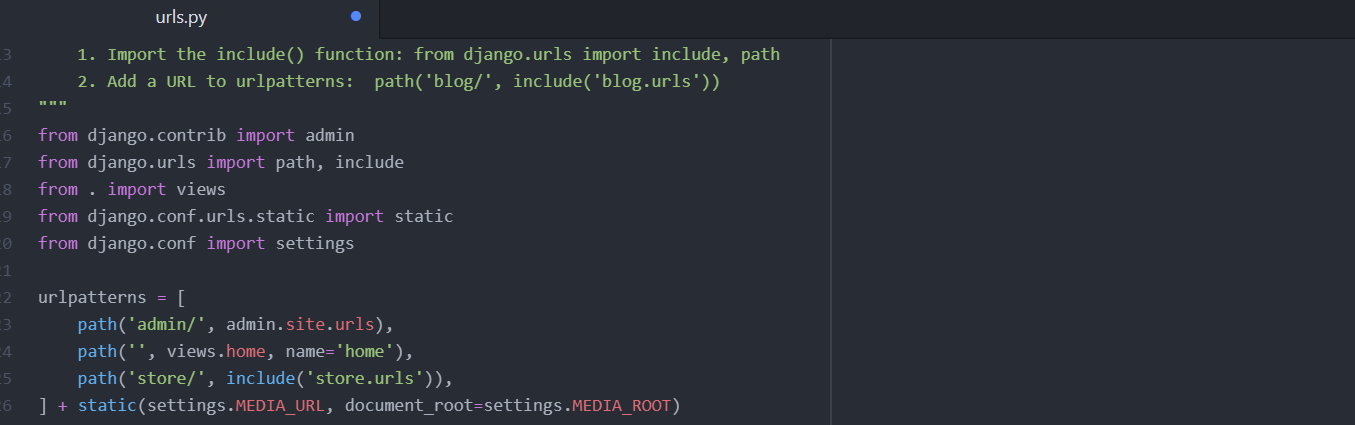
* Go to ‘git bash’ & run commands ‘make migrations’, ‘migrate’ & ‘runserver’ 🡪 with this “products” field is created inside the admin panel.
* **Add Products:**
* Add all the products with proper name, description, price, image, stock availability etc., and also put that product in particular category 🡪 you can access the images of the products from the ‘greatkart images’ folder.
* **Display Products in Homepage:**
* Next job is to display all these products inside our template. We need to display all the products in our homepage 🡪 For this we need to go to ‘home’ view i.e., inside ‘greatkart’ folder go to ‘views.py’.



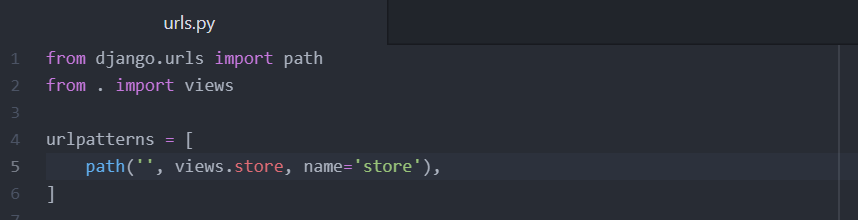
* Now go to the ‘templates’ folder and inside that select ‘home.html’.
* Remove all the <div> to </div> portions – keeping only one <div> tag in action, perform the required changes so that all the products and it details display on the screen.



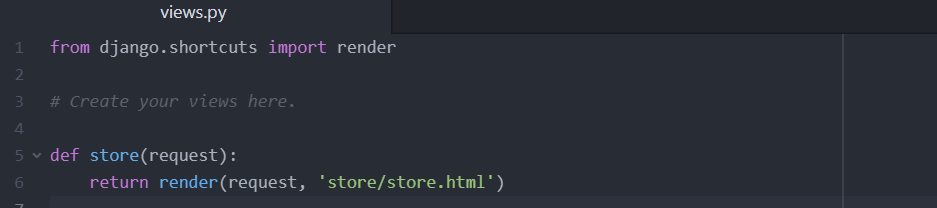
* **Make Store Template:**
* Here we will set up the ‘store page’ – For this 1st we need to setup store urls – so go to the ‘store folder’ (as we don’t have urls.py file inside the store app) 🡪 copy the ‘urls.py’ file from ‘greatkart’ folder & paste it in the ‘store’ folder.
* Now open the ‘urls.py’ of ‘store’ folder & remove unnecessary urls from it.
* Whenever an user comes to any url, 1st he comes to urls.py of greatkart folder (project level directory – main url.py file) - Now we need to root the user from there to ‘urls.py’ of the ‘store’ folder 🡪 So open ‘urls.py’ file of greatkart & add the root.

****

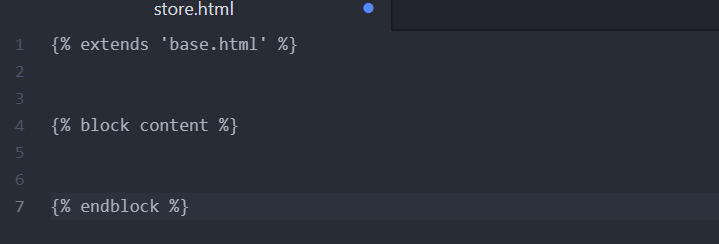
* Now go to ‘urls.py’ file of ‘store’ folder & include the urls

****

* Thereafter go to ‘views.py’ file of the ‘store’ folder

****

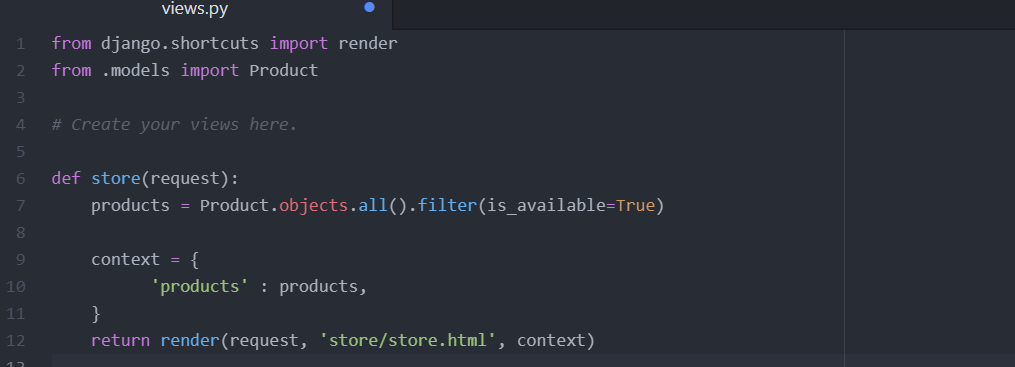
* Now we need to create the “store” folder inside the ‘templates’ folder. And inside that ‘store’ folder create a new file called ‘store.html’.
* Next step is to add the base template (i.e., add header & footer part inside the html page) inside ‘store.html’

****

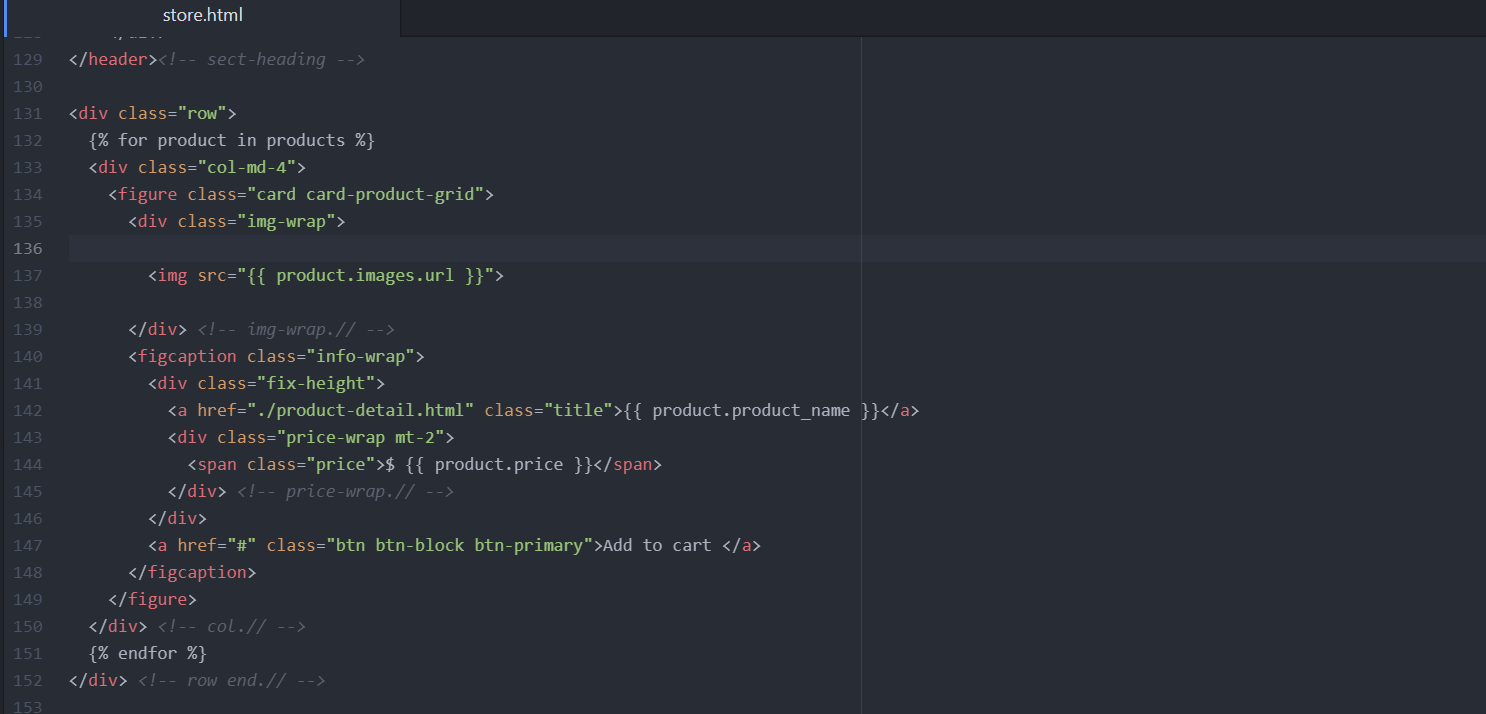
* At next render the html template inside the store page. Inside the ‘block content’ we need to add all the bootstrap code, which we copy from the “store.html” of the “greatkart\_templates” folder 🡪 Here copy the “section top” and “section content” portions and paste it in the store.html.
* Now to load the images include “{% load static %}” command after {% extends ‘base.html’ %} and also include dynamic urls in the image hyperlink sections

****

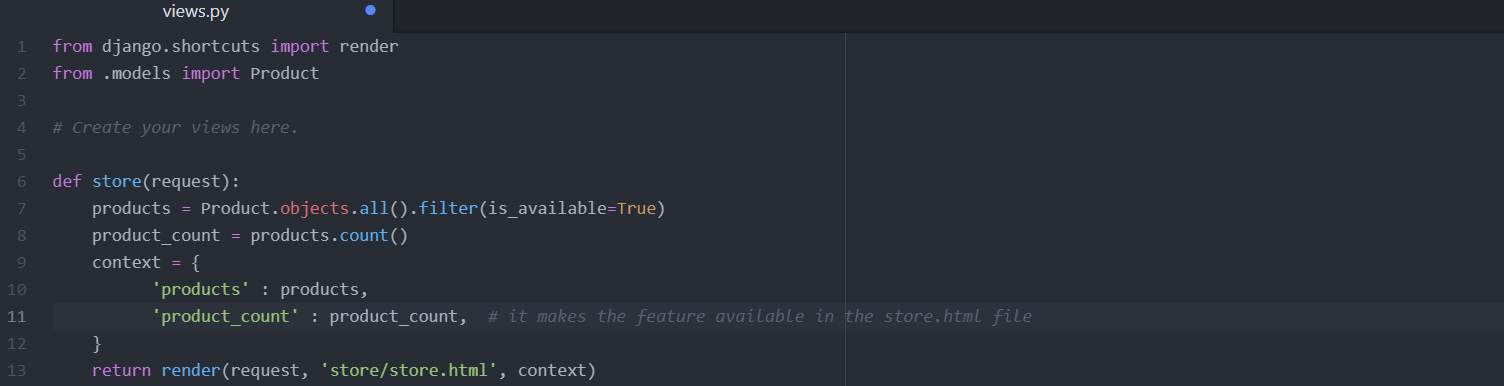
* **Display Products in the Store Page:**
* Now in this lecture we’ll display all the products in the ‘store page’ – So go to ‘views.py’ file of the ‘store’ folder where we’ll render the store.html page.



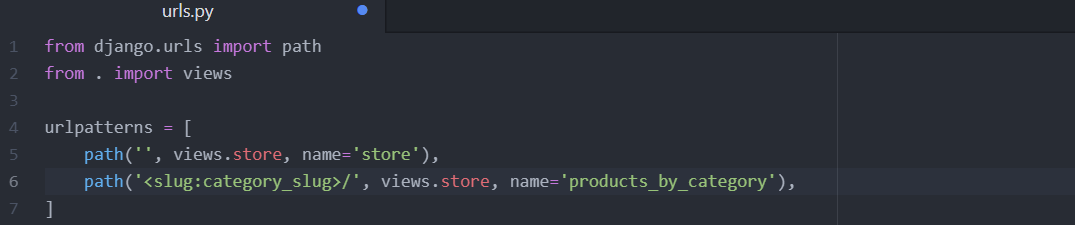
* Now we’ll go to ‘store.html’ and there at 1st you remove all the <div> to </div> tags except the first <div> tag that are below the <header> section



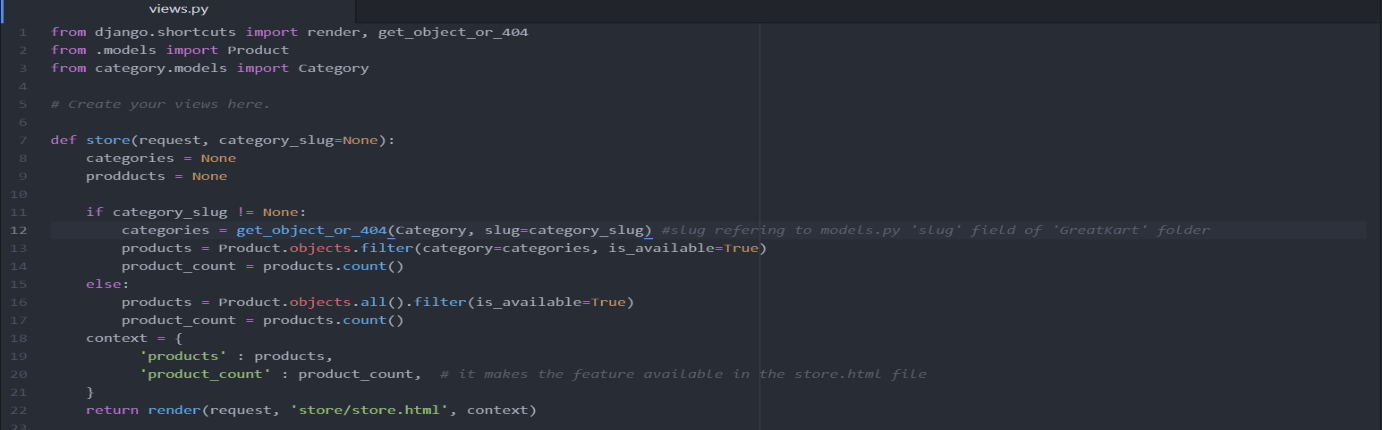
* Now all the products with respective images & price tags are displayed inside our ‘store’. On the top left corner of the store, we notice the static number of products found (32) 🡪 so we’ll make the no of products count & display the dynamic quantity – For this go to ‘views.py’ file of ‘store’ folder. After that insert dynamic url in the store.html.



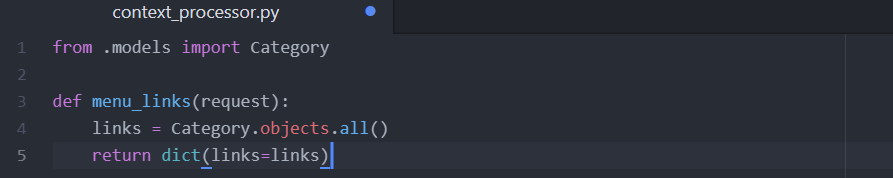
* **Display Products by Category:**
* Whenever the user wants a particular product belonging to a category, then those products should only display inside the store (127.0.0.1:8000/store/shoes)
* 1st of all we’ll make url pattern so that it matches with the categories 🡪 so 1st go to ‘urls.py’ file of the ’store’ folder.



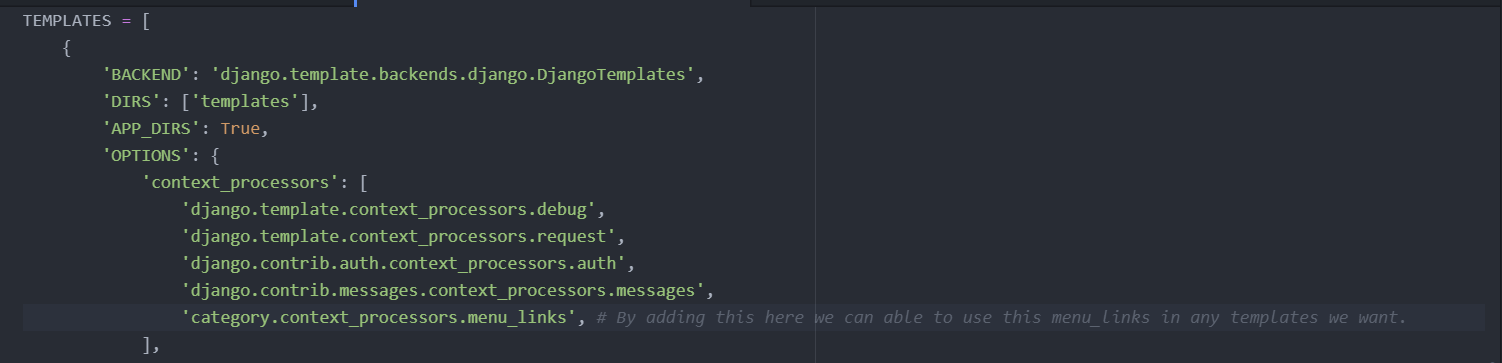
* In the next step go to ‘views.py’ file of the ‘store’ folder (here we are bringing the slug)



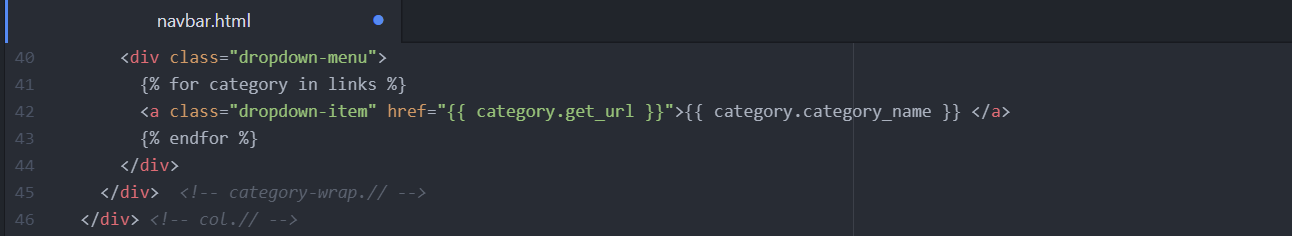
* **Context Processors Display Category Navbar:**
* Now we need to list down all the categories & can able to select the products that are required 🡪 For this we’ll be using python processor or function called “Context Processors”.
* Go to ‘category’ folder – right click on it ‘create new file’ – “context\_processors.py” (It is a python function which takes request as an argument – and it will the return the dictionary of data as a context



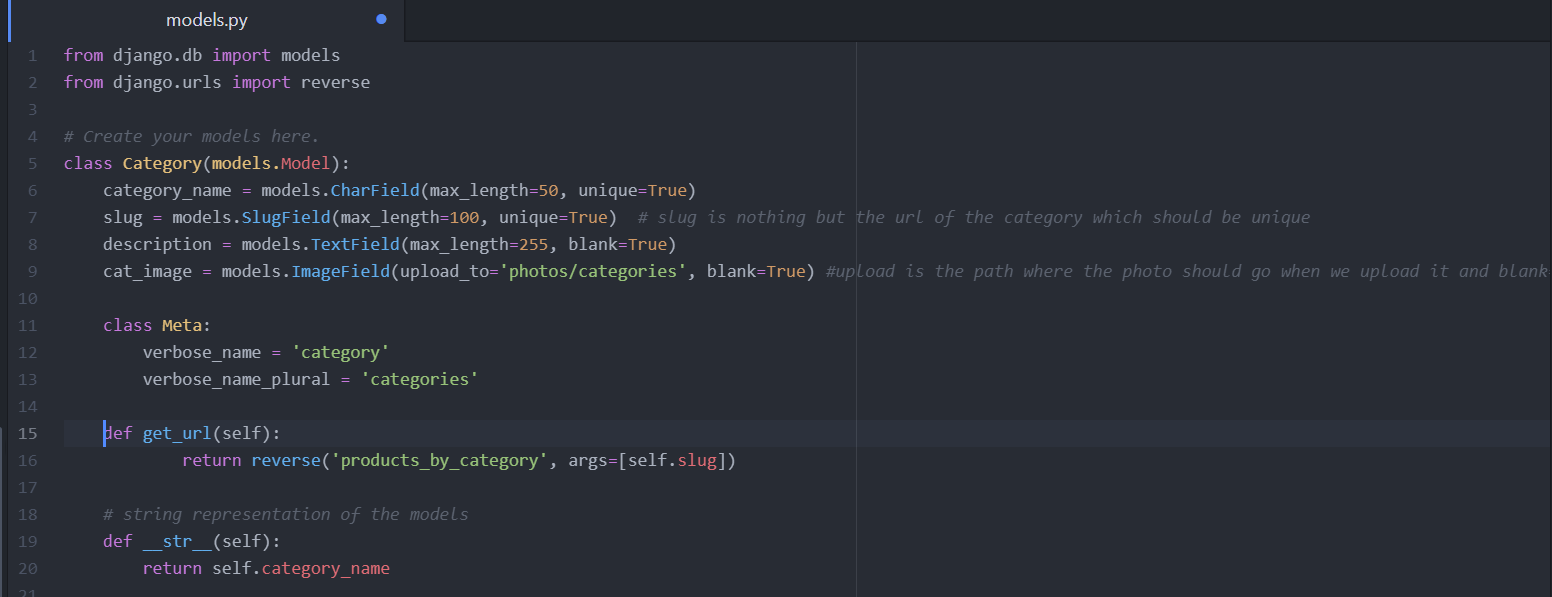
* Since we are using ‘context processor’ we’re going to tell the templates that we are using the context processor – For this go to ‘settings.py’ file & inside the template the ‘Templates’ section adds the details of context processor.



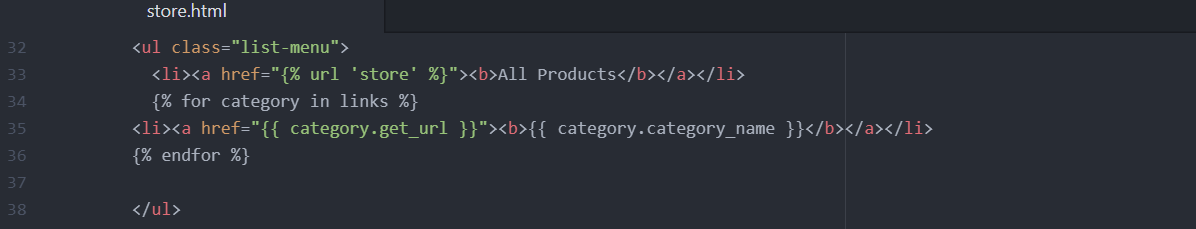
* **Note**: In ‘settings.py’ - **Type Error ‘category.context\_processor.menu\_links’,**
* Now go to ‘navbar.html’ file inside the ‘includes’ folder – there remove the unrequired categories already present & modify one category as a dynamic.



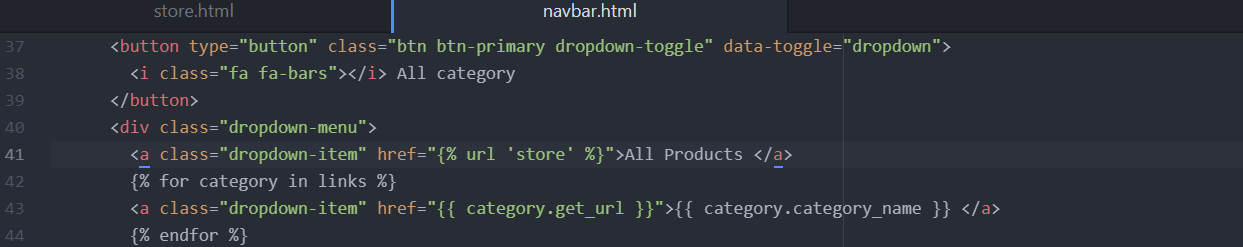
* Now we go to ‘models.py’ file inside the ‘category’ folder & add the ‘get\_url’ function there. With this the category button becomes functional



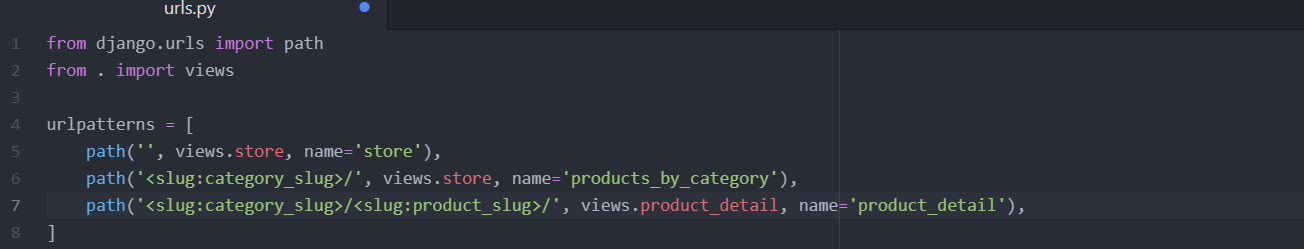
* **Display Category in Store Page:**
* Now we need to list down all the categories in the left side of the store page – For this go to ‘store.html’ file inside the store folder – there remove all the existing categories with one exception



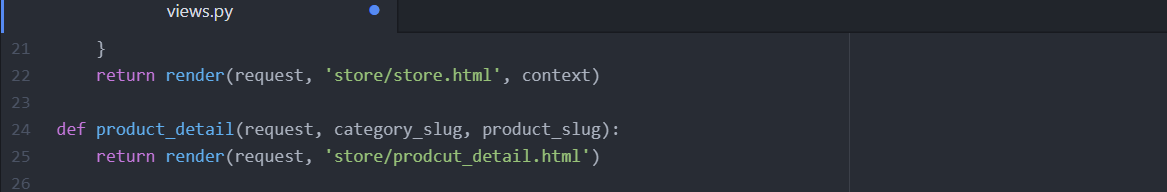
* Now include ‘All products’ option in the ‘navbar.html’ page also



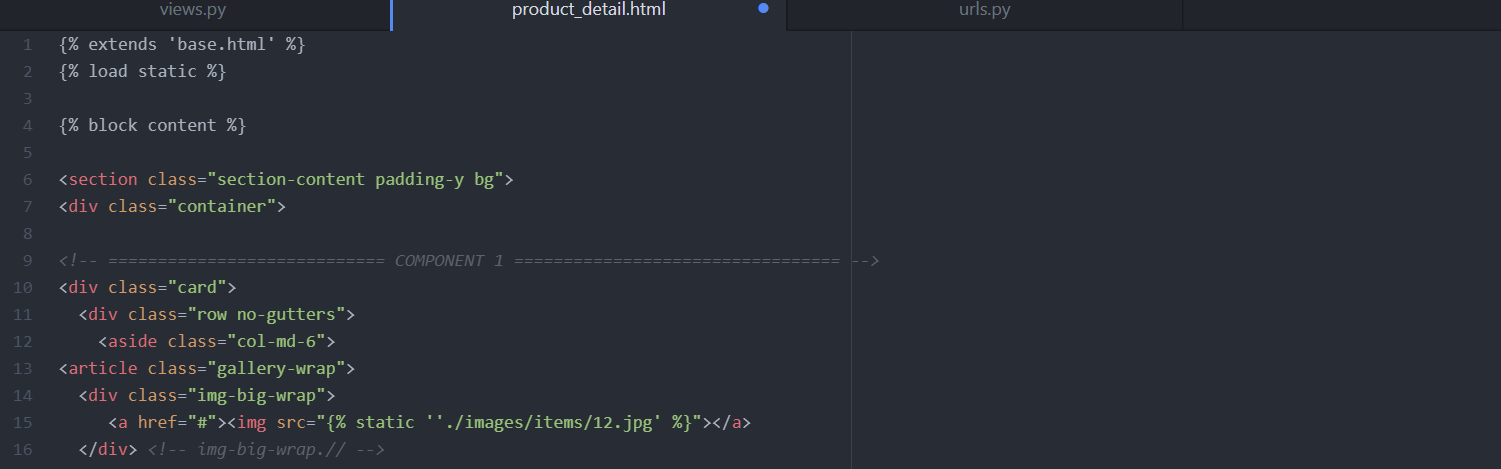
* **Product Detail Url and Make Template:**
* Now we’ll make a ‘Single Product’ page – whenever user click on a product it will redirect to the complete detail about the product. (127.0.0.1:8000/store/category\_slug/product\_slug).
* 1st go to ‘urls.py’ of the ‘store’ folder



* Then we’ll make the product detail view in ‘views.py’ of ‘store’ folder.



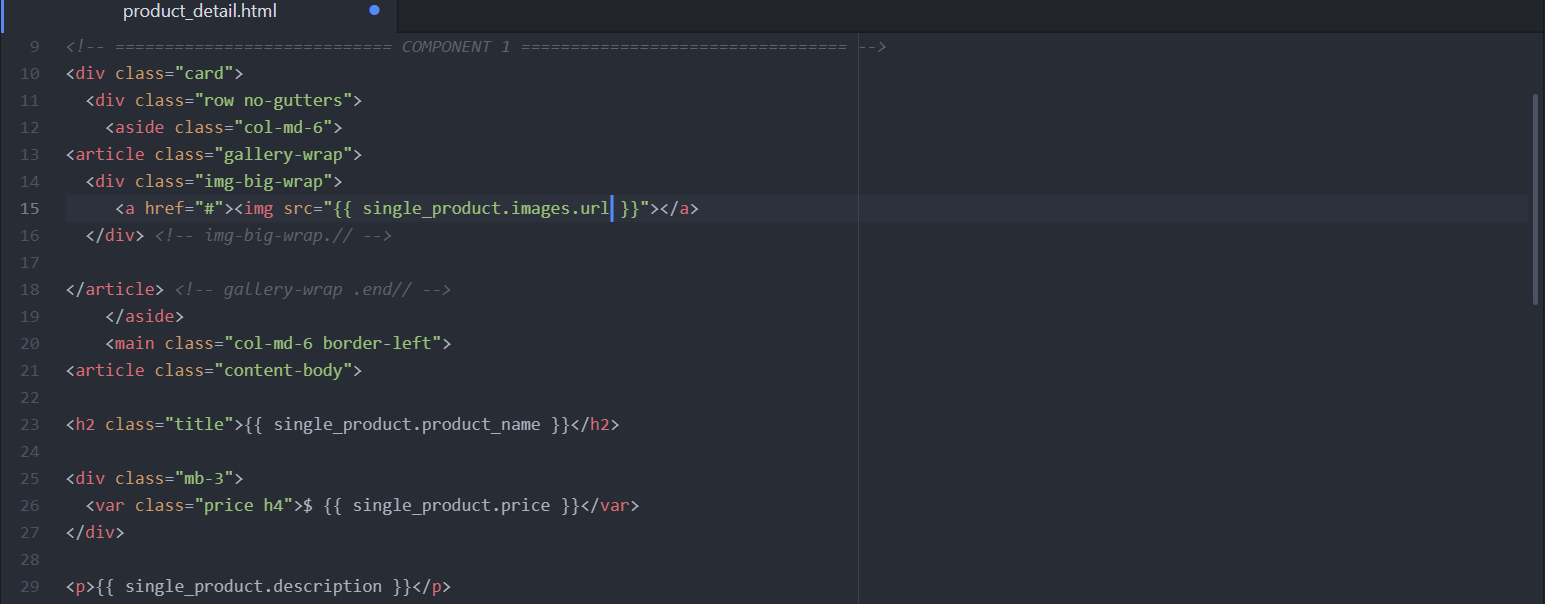
* Now create a new file called ‘product\_detail.html’ inside the ‘store’ folder of the ‘templates’ parent folder. Inside this html page we’re going to include our ‘bootstrap’ template. 🡪 For this go to ‘greatkart template’ folder inside vs code – there open ‘product\_detail.html’ file and there copy the section content present below the header part (i.e., line no 139 to 248). Include the extends, block content & end block commands in the product\_detail.html page and paste the copied content b/w block content & end block.
* Thereafter include the load static command & include the dynamic url for images in the component 1 & customer reviews section.



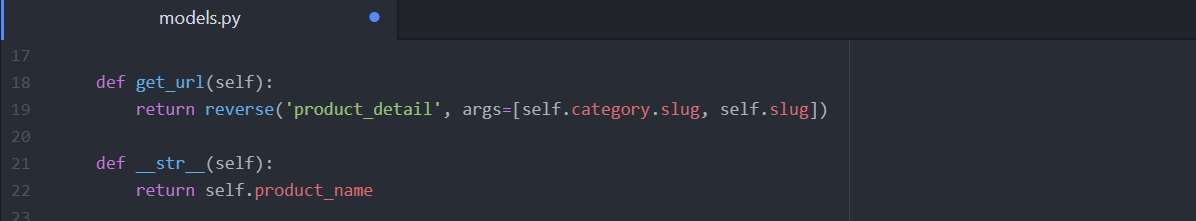
* **Single Product View:**
* Now we work on the display of the complete detail about a product, when user clicks on that particular product in a new page. So, 1st go to the ‘views.py’ file of the ‘store’ folder



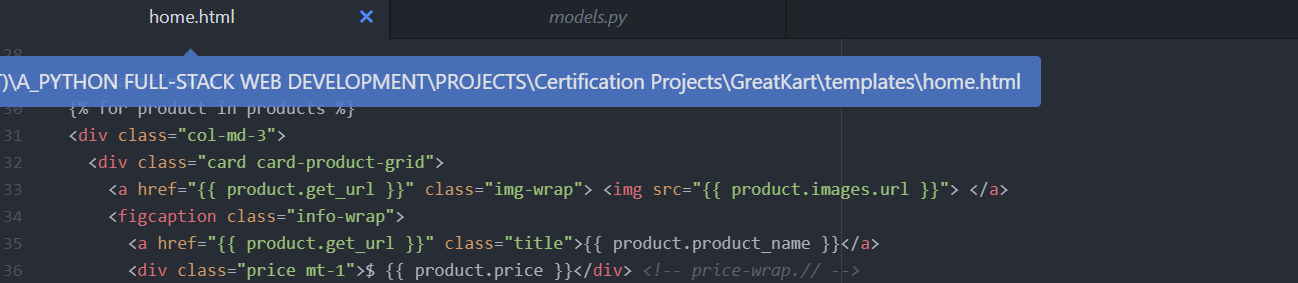
* Then go to ‘product\_detail.html’ file inside the ‘store’ folder and make the dynamic url changes to display particular product with specific name, description, image & price.



* **Get Url for Product:**
* Now we need to fix the link for every product displaying on the store, so that when the user clicks on a particular product it should redirect him to details of that product.
* So now we need to create the ‘get\_url’ function for the products, similar that we did for the categories. For this go to ‘models.py’ file of the ‘store’ folder.



* And also import reverse command ‘from django.urls import reverse’
* Thereafter go to ‘home.html’ page inside the ‘store’ folder & include the dynamic urls to the page 🡪 so that when we click on a product it should redirect to its details.



* Thereafter, we need to activate the hyperlinks for the features present in the ‘navbar.html’ like store, Greatkart logo etc., - For this go to ‘navbar.html’ page of ‘templates’ folder.

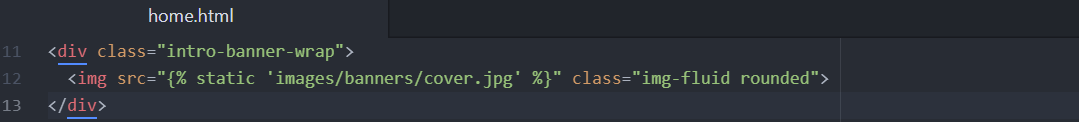




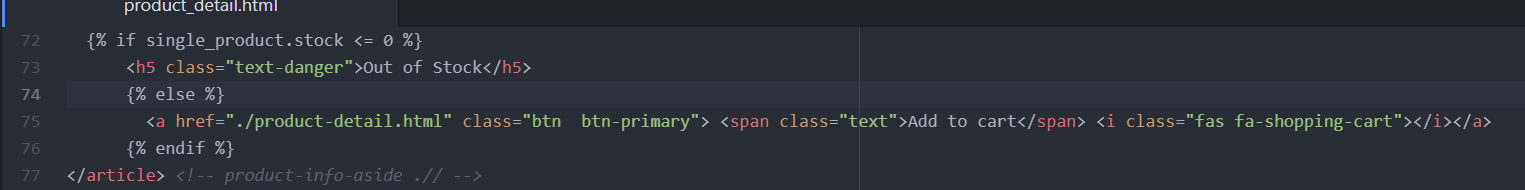
* Then we need to fix the ‘see all’ button in the home page – For this go to ‘home.html’ file of the ‘store’ folder.
* There after fix the ‘product name link’ inside the store – For this go to ‘store.html’ file



* **Changed Cover Photo:**
* Now we remove the cover banner from the home page & replace it with another – For this go to ‘GreatKart\greatkart\static\images\banners’ here we need to paste the cover banner image which we will copy from ‘GreatKart\GreatKart-Images\GreatKart Images’.
* In the next step, go to ‘home.html’ file & change the cover image name there



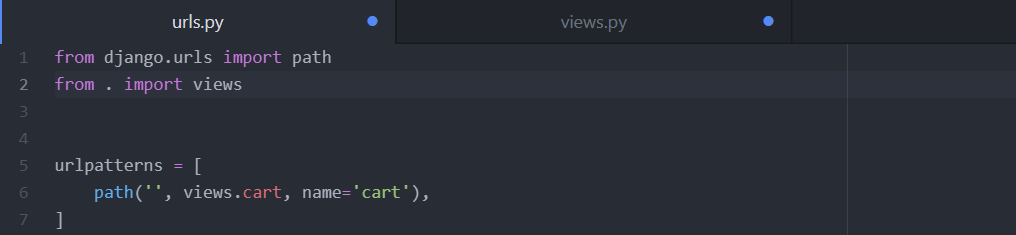
* **Out of Stock Tag:**
* In the product detail page, there is display of ‘Add to Cart’ option. But when there is no stock of that product we’re supposed not to display ‘Add to cart’ option & instead display ‘out of stock’ – For this go to ‘product\_detail.html’ file of ‘templates’ folder



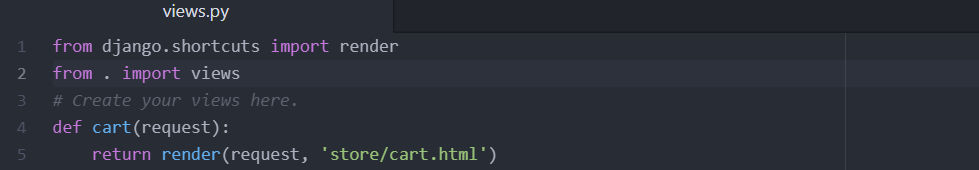
* **Push Code to the Git:**
* Now we push all the code to git hub repository – sign in to git hub & click on ‘New Repository’ - “greatkart-django” (public).
* Now go to ‘git bash’ – here we need to initialise the repository – so type the command “**git init**” – it creates a “. git” folder inside your project.
* Next will create a new file inside the root folder ‘GreatKart’ – named “.gitignore” 🡪 Here we are going to mention all the folders & files that we don’t want to send to git repository (EG: Image/ Media/ Mp3 files etc – usually big in size & are not preferred to move to git repo.
* Now go to browser & type “gitignore.io” - & in the search box type “Django” – it results the git ignore code file which is used for Django projects – copy all the code there & paste it inside the ‘.gitignore’ file of your project & save it.
* Next go to ‘git bash’ & type the command “**git bash -A**” to add all our code to git repository. (-A means we’re adding all the code to the git repository)
* **Note: When you’re using the git for the 1st time it may ask you to set up your identity,** then you’re supposed to use 2 commands
* **$ git config --global user.name "John Doe"**
* **$ git config --global user.email johndoe@example.com**
* Our next command inside the ‘git bash’ is – “’**git commit -m “This is my first commit”** ‘’’.
* Next command is **- “git remote add origin** [**https://github.com/Tharun-Rdy/greatkart-django.git**](https://github.com/Tharun-Rdy/greatkart-django.git)**”. (which you copy & paste from the git repository).**
* Next command is – “**git branch -M main**”.
* Next command is – “**git push -u origin main**”.
* If you want to check any code is needed to push to git then use the command – “git status”.
* **Carts App and Template:**
* Here we work on the cart app functionality – for this go to ‘git bash’ & type command to create the new cart app – “python manage.py startapp carts”.
* Then go to ‘settings.py’ & register the carts app in Installed\_Apps section.
* Now as we don’t have ‘urls.py’ file inside the carts, we create manually one there.
* As at 1st the request comes to project level directory ‘urls.py’ file of ‘greatkart’ folder, we include that request of ‘urls.py’ file of cart folder there.



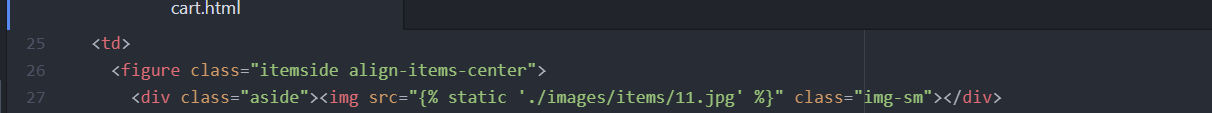
* Now add the ‘url patterns’ in the ‘urls.py’ file of ‘carts’ folder

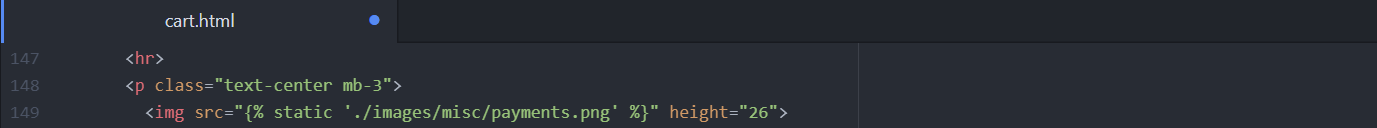


* Then we make the cart function inside the ‘views.py’ file of the ‘carts’ folder.

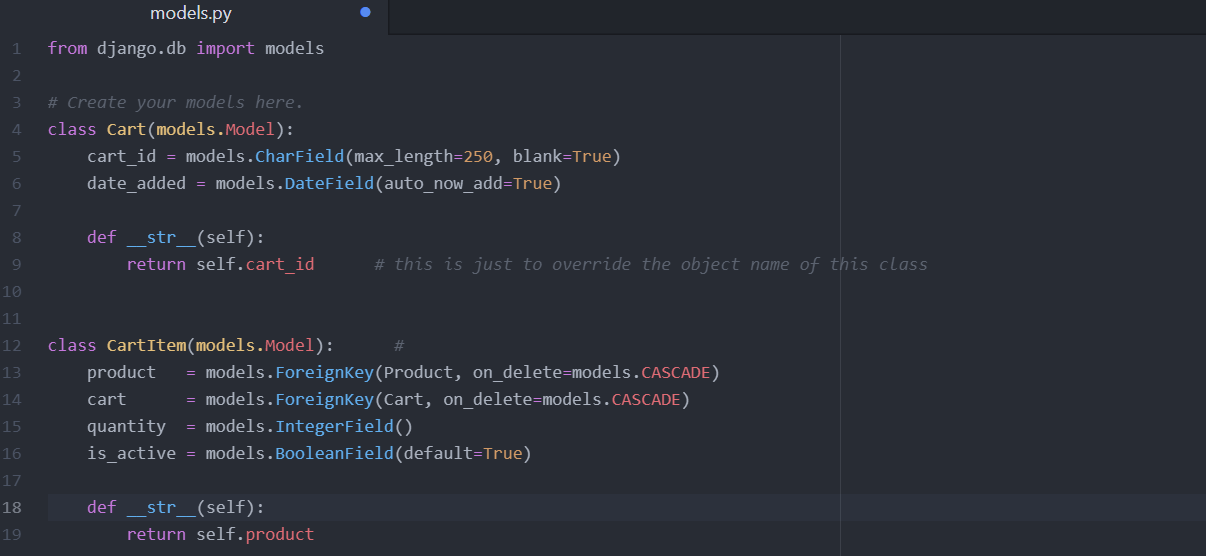


* As we don’t have a ‘cart.html’ file inside the ‘store’ folder – we’ll create a new ‘cart.html’ file in the ‘store’ folder of the parent ‘templates’ folder. 🡪 Inside this ‘cart.html’ file we ‘extend the base template’, initial the ‘block content’ & ‘end block’. 🡪 Thereafter go to sublime text in vs code & copy the “section” portion present after ‘header’ block ends & paste it in b/w ‘block content’ & ‘end block’ of ‘cart.html’ file.
* Thereafter we need to fix the static images & other stuff in ‘cart.html’ – Before that don’t forget to {% load static %} below the base template 🡪 Then we need to include the static command for the 1st product & payment options.

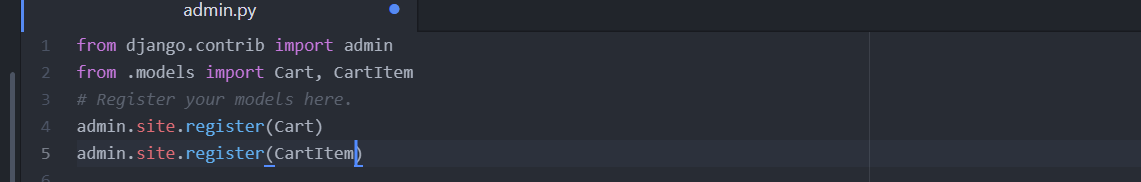




* Now we’ll remove the other 2 product images i.e., 2nd & 3rd <tr> to </tr> & save it.
* **Cart and Cart Item Model:**
* In this lecture we’re going to create a ‘model’ for our ‘cart’ – 1st go to ‘models.py’ file of the ‘carts’ folder.

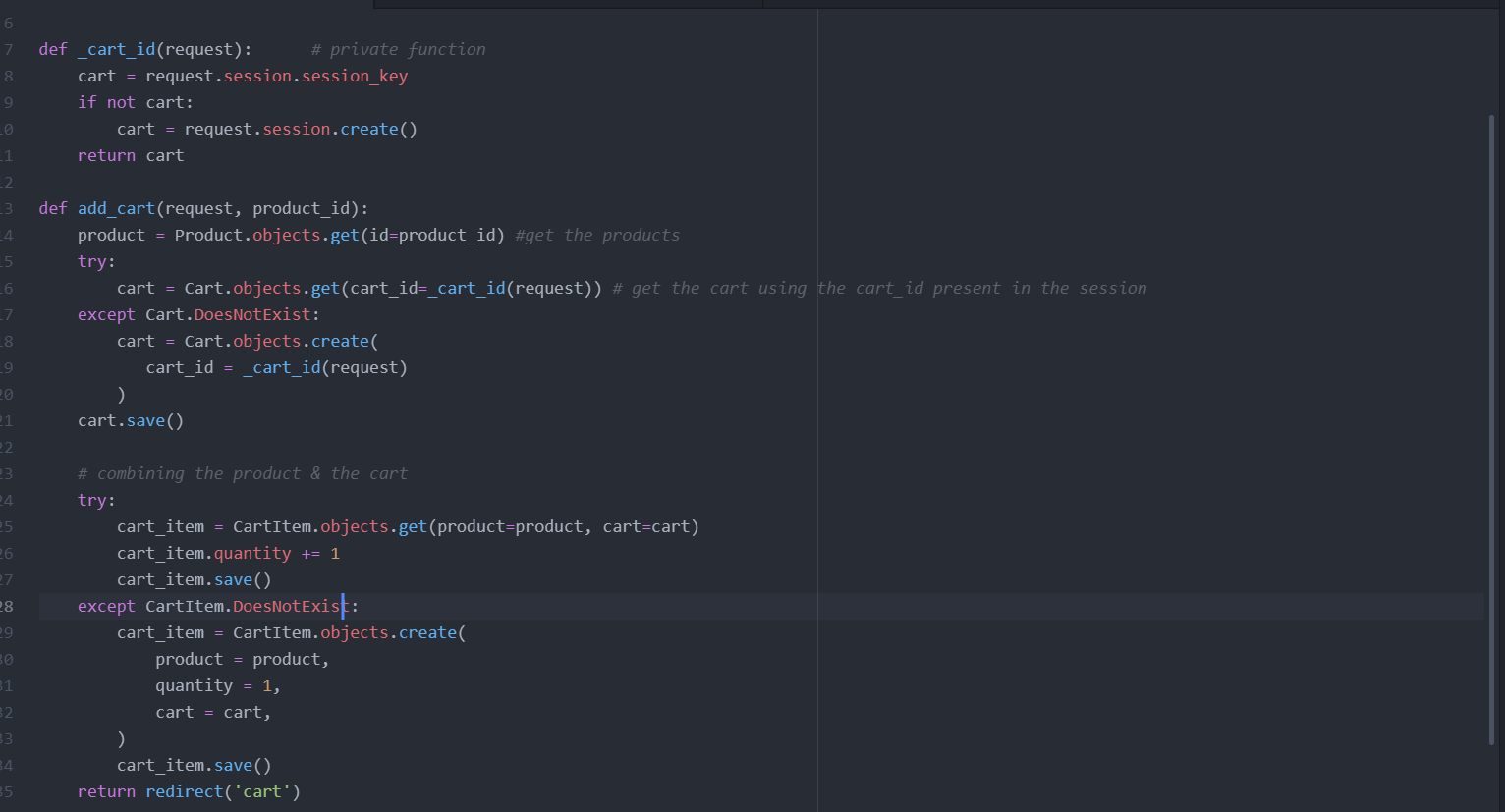


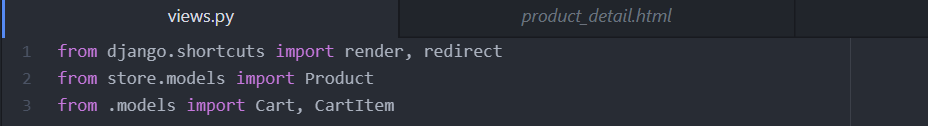
* **Note**: Include **“from store.models import Product”**
* Now let’s run the ‘make migrations’ & ‘migrate’ command inside ‘git bash’ terminal.
* Then we need to register the ‘cart’ & ‘cart Item’ inside the ‘admin.py’ file of ‘cart’ folder.



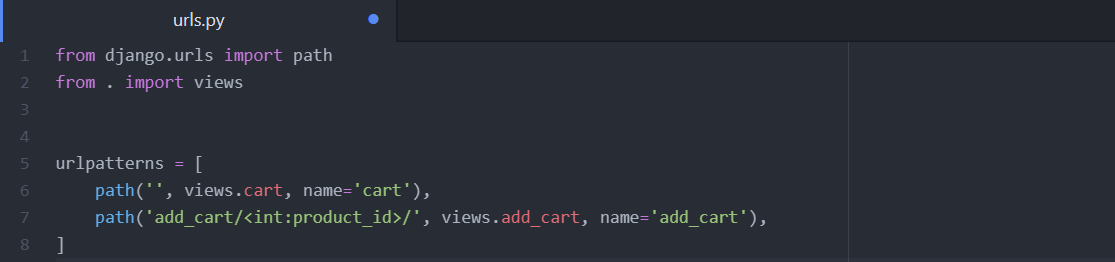
**Section - 7**

* **Add to Cart Functionality:**
* Whenever we click on the ‘Add to Cart’ button – then it should function to add that particular product to the customer cart & also redirect the customer to the cart 🡪 for these things we have a functionality called “session” in which we have “session keys” – we’re going to store the session key as a Cart ID.
* 1st we need to make ‘Add to Cart’ button functional – so go to ‘views.py’ file of the ‘carts’ folder and “create a new cart item there”
* We have a product & a cart – and now we need to put the product inside that cart – so that the product becomes the cart item. We’ll combine the product & the cart so that we’ll get the ‘cart item’

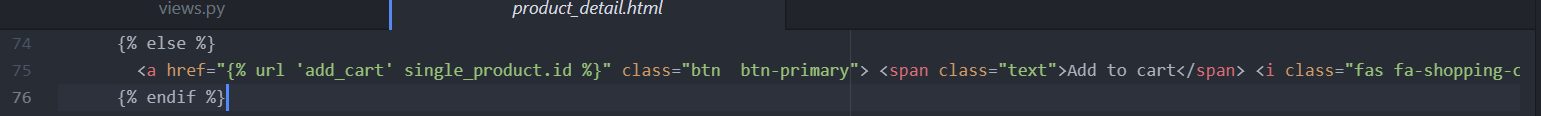
****

****

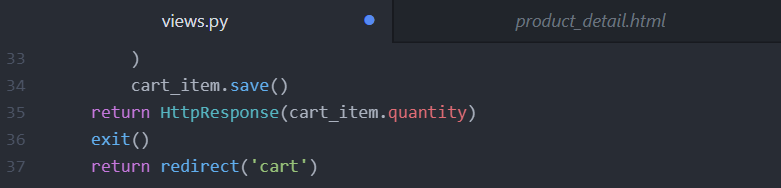
* Thereafter go to ‘urls.py’ file of the ‘cart’ folder

****

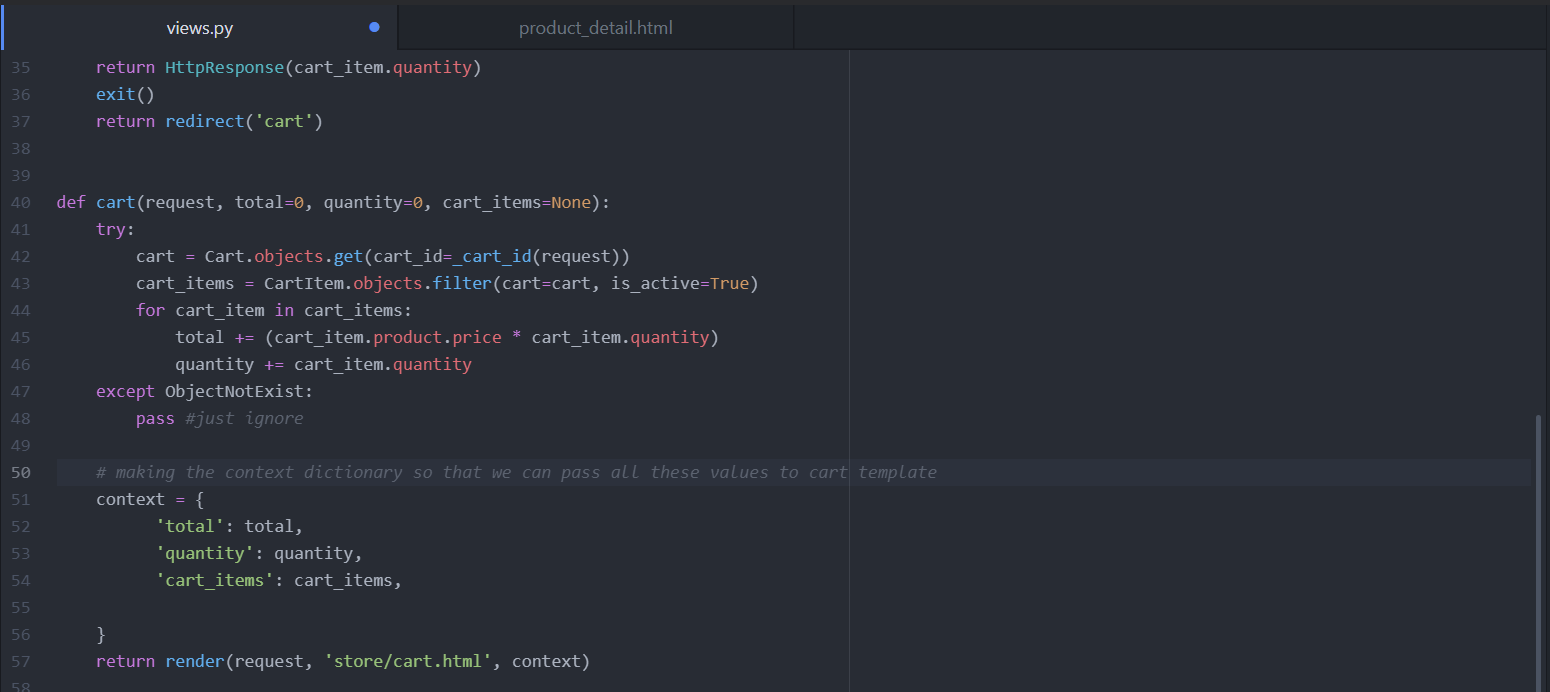
* Now to make the ‘Add to Cart’ button functional go to ‘product\_details.html’ file of the ‘store’ folder of the parent ‘templates’ folder.

****

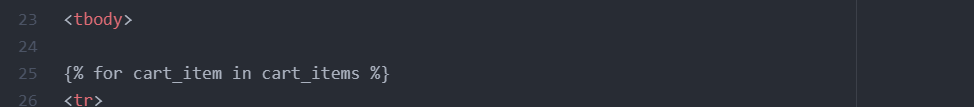
* When we click on ‘Add to cart’ button of a particular product – then the dynamic message like product id or name or quantity should display – for this go to ‘views.py’ file of ‘carts’

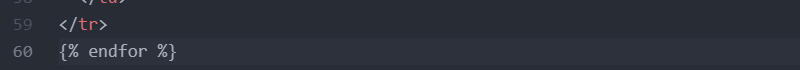
****

* **Get Total Quantity Cart Items:**
* Now we’re able to save the cart item & also able to redirect to the cart page. But the problem is we’re not taking anything to the cart page – we’re just going blank. Instead, we need the total, quantity, product information etc., - for this we need to modify the ‘cart function’ – so go to ‘views.py’ file of ‘carts’ folder

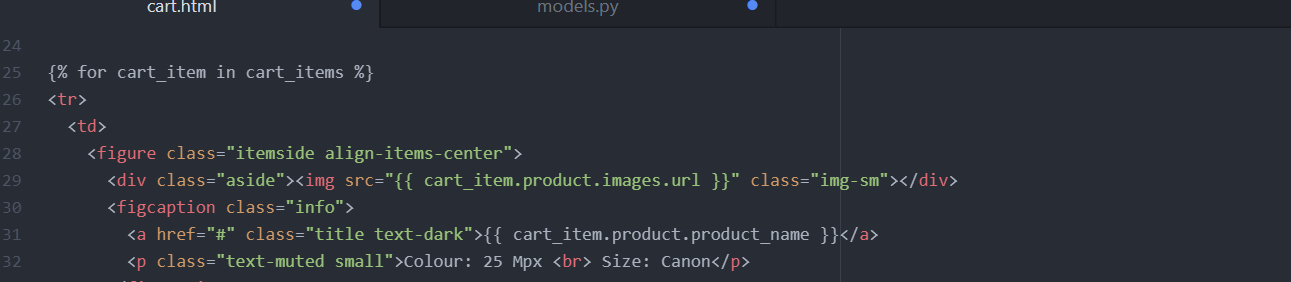


* **32 – Implement Data into Cart Page:**
* We have all the data, now we’ll go to the cart page now, as we have the text displaying in the add\_cart section we need to remove the return HttpResponse & exit () lines [i.e., 35 & 36] in the views.py file – save & refresh – then ‘cart’ will get display.
* Now the data we entered in ‘views.py’ will be displayed in the ‘cart’ page – for this 1st of all we will ‘loop through all the cart items, so that it brings all the cart items in the cart page – so then we can print all the details of the product there.
* So 1st go to ‘cart.html’ page inside ‘store’ folder of parent ‘templates’ folder 🡪 There at 1st we will loop through <tr> to </tr> [24 to 57]

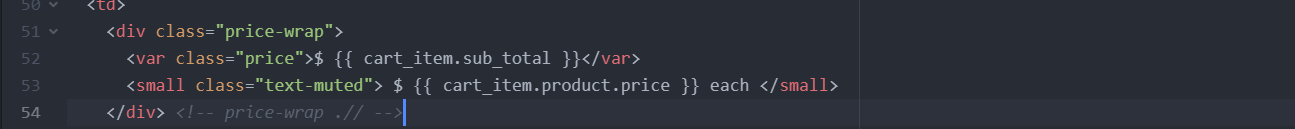
****

****

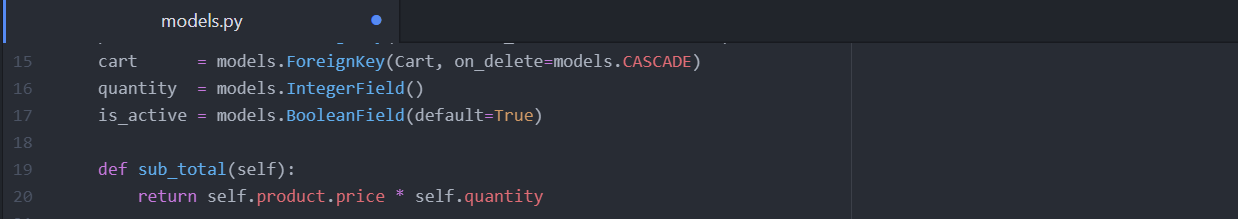
* Now if we save & refresh then multiple products in cart will appear, at next we need to change the names, images of those products in ‘cart.html’ itself

****

****

****

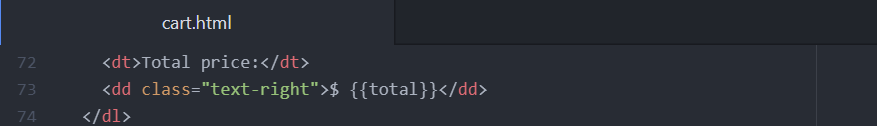
* As we don’t contain the sub\_total function, then we need to create one inside ‘models.py’ file of the ‘carts’ folder.



* Now when we click on the ‘+’ button in the quantity section, then the total quantity should increment – go to ‘cart.html’ and ‘plus’ command – there remove the ‘button’ tag as we need to insert <a> tag along with ‘href’ tag there



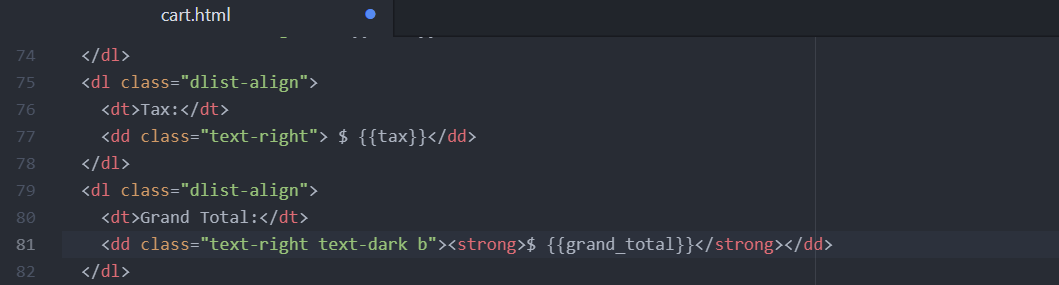
* Now we work on the 'total price’ feature



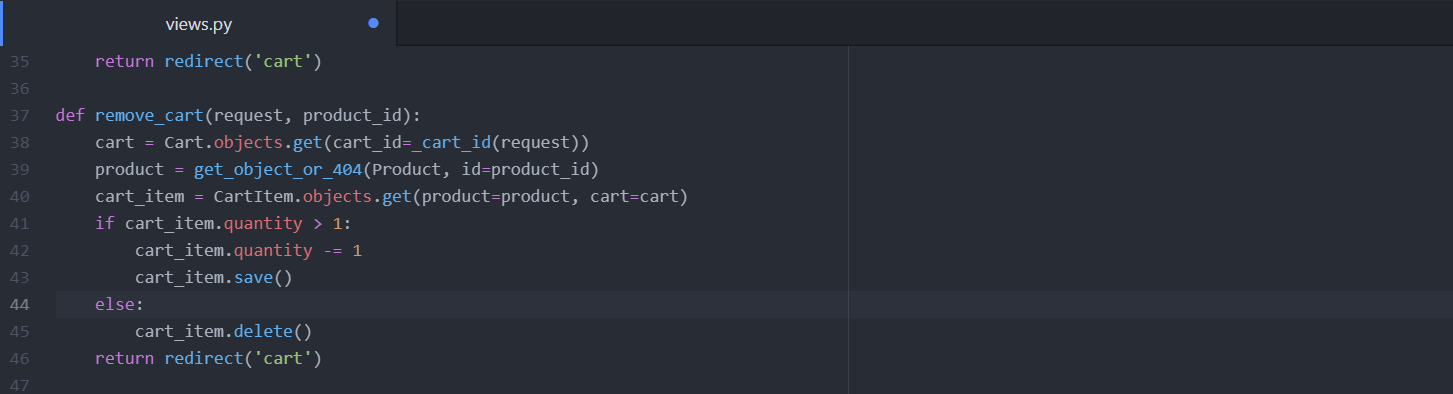
* **Calculate Tax Grand Total:**
* Here we need to calculate the ‘tax’ & the ‘grand total’ of the products in the cart – so 1st go to ‘views.py’ of ‘carts’ to include tax & grand total functions there.



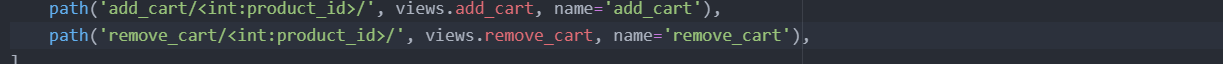
* Then we change the ‘tax’ & ‘grand total’ functions inside the ‘cart.html’ also



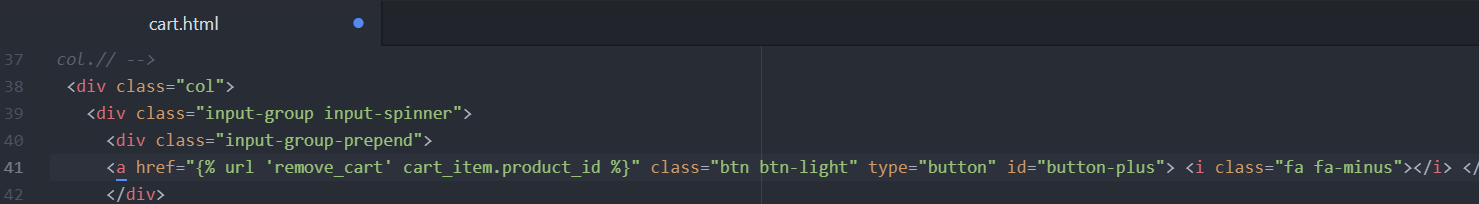
* **Cart Decrement and Remove:**
* Now here we’ll make the “-“ [minus] button – which should decrement the product quantity and “Remove” button – which removes the product from the cart, both functional.
* 1st we’ll work on the minus button – for this go to ‘views.py’ of ‘carts’



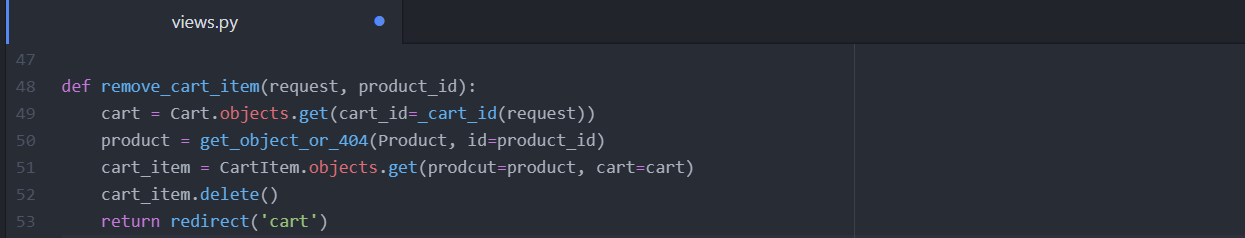
* Note: from django.shortcuts import get\_object\_or\_404
* Thereafter we need to define the path – so go to ‘urls.py’ of ‘carts’



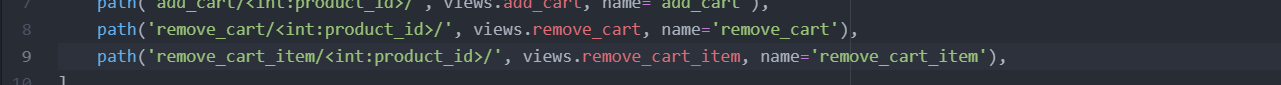
* Next go to ‘cart.html’ file & modify the ‘minus’ button



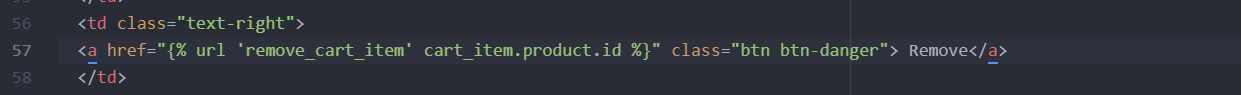
* With this the ‘minus’ button became functional
* Now we work on the ‘remove’ button – for this 1st go to ‘views.py’ of ‘carts’ & will define the remove function there.



* Now we’ll create the url path for the file in ‘urls.py’ file of ‘carts’

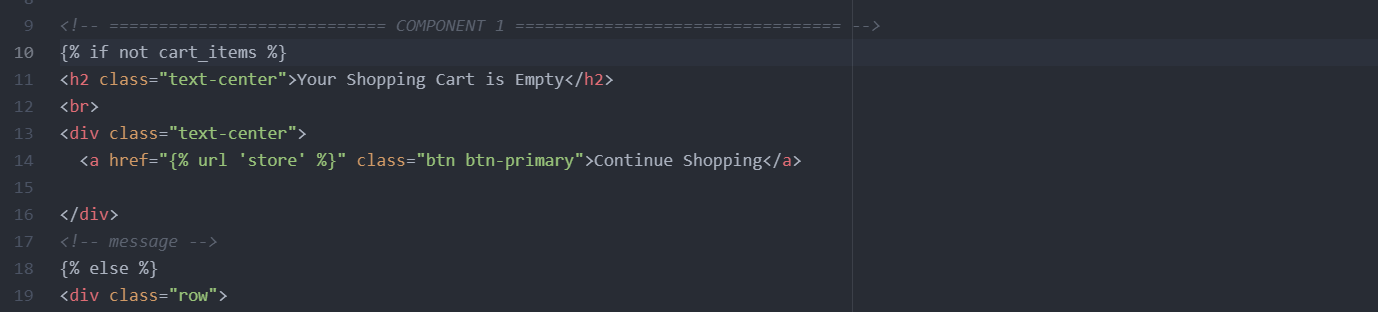


* Now configure those changes to remove button in ‘cart.html’

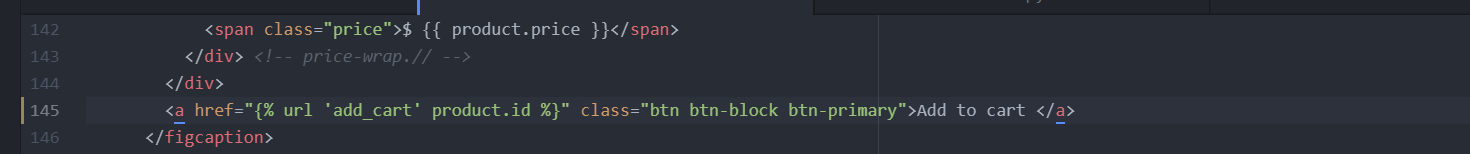


**SECTION-8**

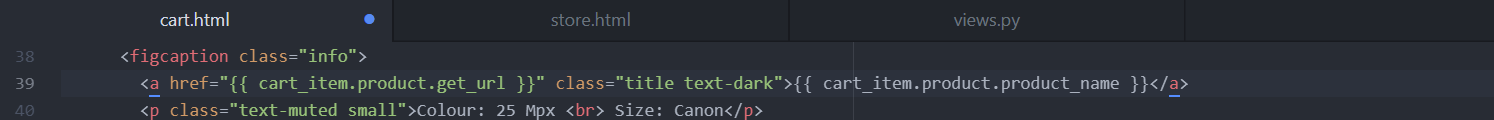
* **35 - Check For Empty Cart:**
* If there are products in the cart, then we should display a relevant message and also include ‘continue shopping’ button in it. So go to ‘cart.html’ & include ‘if-else’ condition & at the end of component-1 include {% endif %} condition.



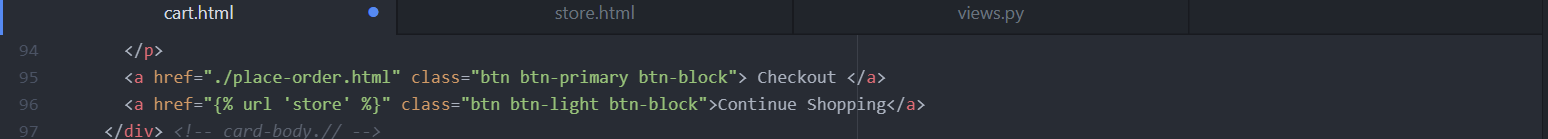
* **Fix Add to Cart Links:**
* Now we’ll make the ‘add to cart’ button functional in the store – go to ‘store.html’ of ‘store’ folder of parent ‘templates’ folder.



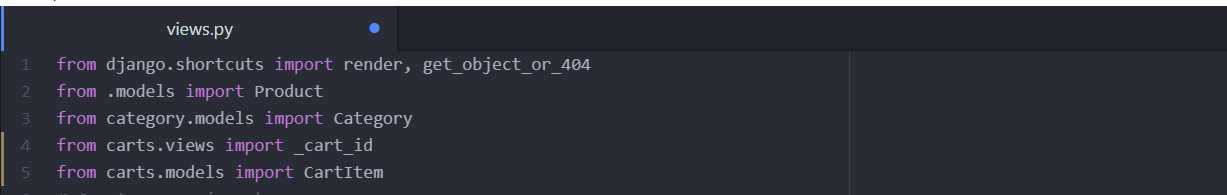
* Now we need to make the product name link active in the cart and also make ‘continue shopping’ button functional present below the payment’s method – go to ‘cart.html’

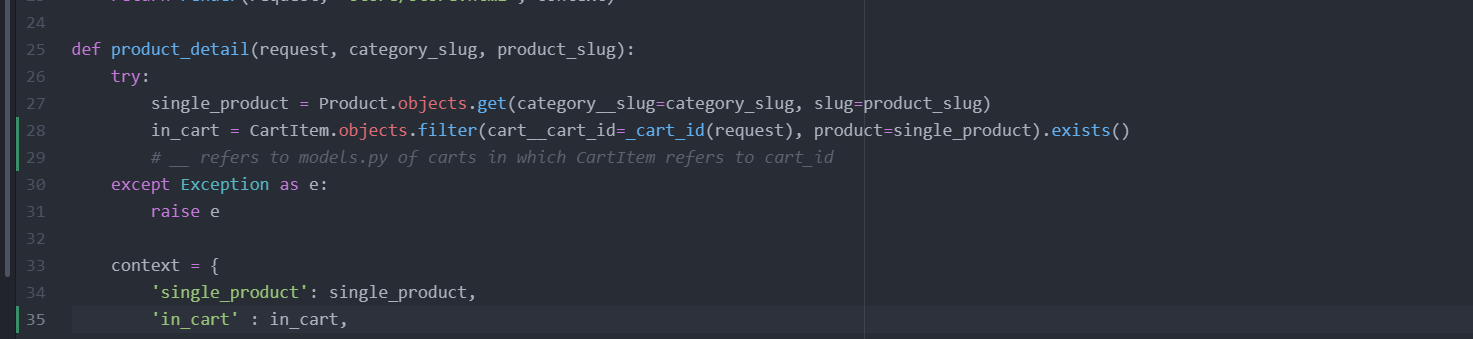


* And also ‘continue shopping’ button functional

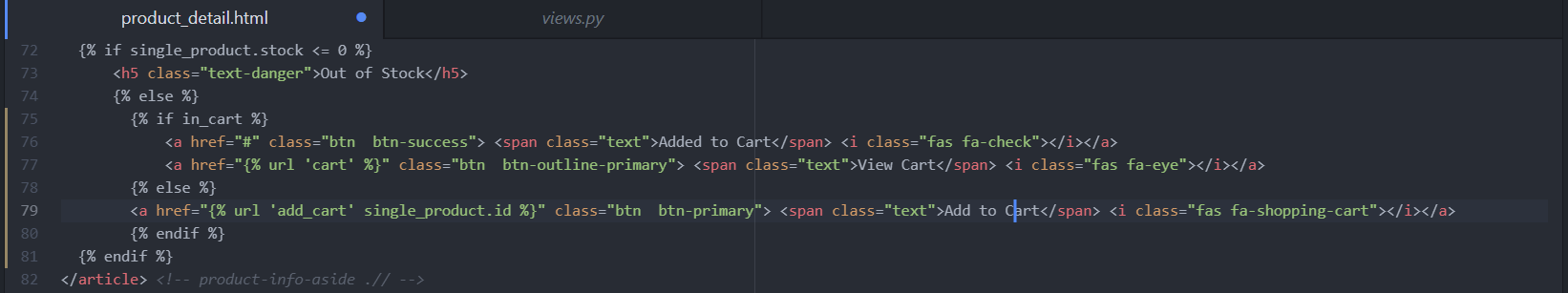


* **Check if the Product Added to the Cart:**
* Now we need to check whether a particular product is existed in the cart or not – for this go to ‘views.py’ of the ‘store’

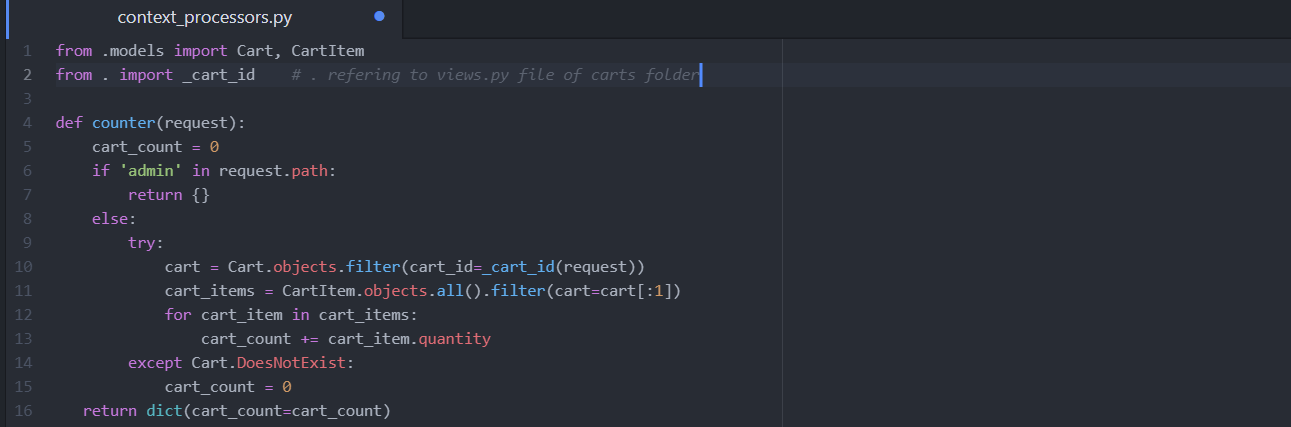




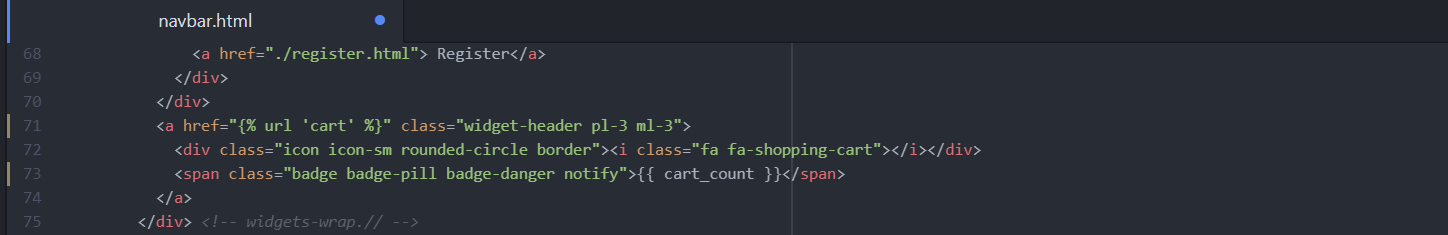
* Now we’ll go to ‘product\_detail.html’ and there we need to hide the ‘add to cart’ button for those products that are already added to cart.



* **38 – Counter For Cart Icon in Navbar:**
* Now we’ll make function for the Cart Item icon present on the right top in header section – For this we’ll go to ‘carts’ folder & there we create a new file called “context\_processors.py” – Here we’ll be working with two models – i.e., cart & cart\_item.

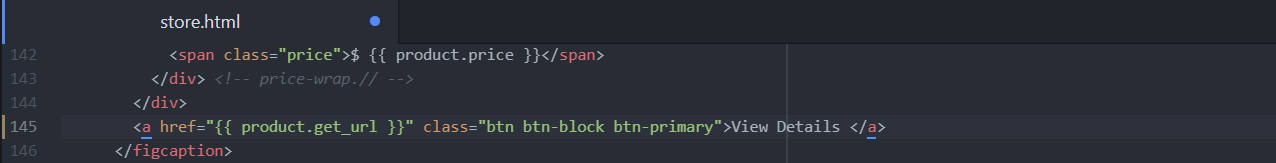


* **Note**: **from .views import \_cart\_id**
* Thereafter we go to ‘settings.py’ file and register the context\_processors inside ‘templates’
* **'carts.context\_processors.counter',**
* Now we go to ‘navbar.html’ and modify the count of products on the icon

****

**SECTION-09**

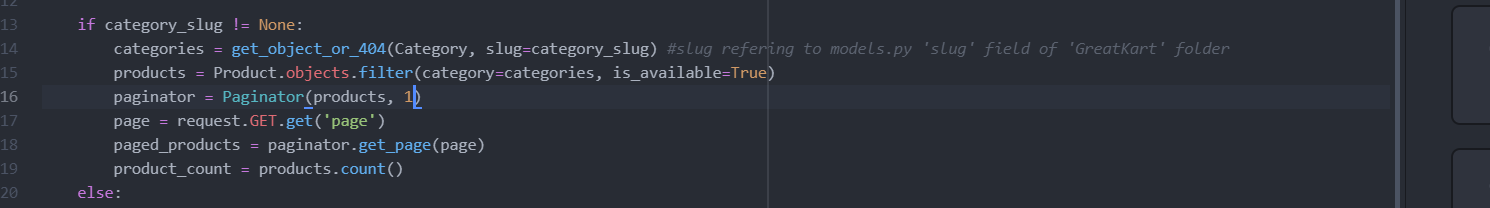
* **39 – View Details Button:**
* For the products that are ‘out of stock’ should also display the same (like in the product detail page) in the store page, but it is not & when we click on the ‘add to cart’ in the store page the item is getting added to cart – so we need to fix this issue.
* For this – 1st we go to ‘store.html’ page in the ‘store’ folder inside the ‘templates’ folder.

****

* **40 – Paginator Part 01:**
* Here we’ll fix all paginators (i.e., page 1,2,3…) on the home page of the website – For this we have a Django documentation available in the google, search as “Django Paginator” – the html code present there need to be customised
* Before that go to ‘store.html’ where we have ‘pagination’ block of code – to deal with that 1st go to ‘views.py’ of store



* **41-Paginator Part 02:**
* Now we’ll work on the products by category feature in the navbar which requires to modify the if condition in ‘views.py’ file of store



* Thereafter we need to work on the ‘previous’, ‘next’ & ‘paginator’ inside the template – for this go to ‘store.html’ file.
* 1st we need to show the paginator if only there are undisplay products available in the store page, and it don’t make sense if products are not available & still paginator is displayed.

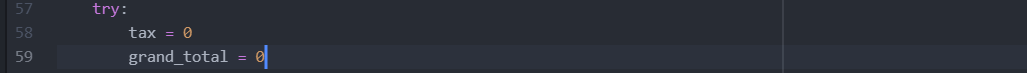


* **Products warning and Empty Cart Issue:**
* When we go to store & the git bash terminal in run server state then a warning is displayed in the terminal – warning: Unordered object\_list warning – which mean that whatever products we are bringing from the ‘views.py’ is not in a proper order [ It is not an Error]
* So 1st go to ‘views.py’ file of the ‘store’ app & include order\_by so that warning won’t display in the git terminal.



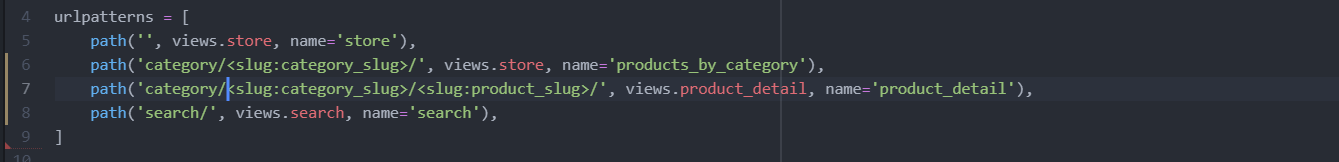
* The next thing is when we click on the ‘cart icon’ on the home page of the website we’re getting ‘Name Error’ so we going to deal it – so go to ‘views.py’ of the ‘carts’





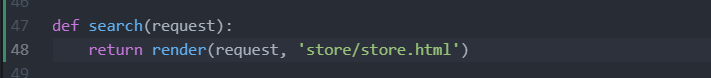


* **43 – Search Function:**
* Now we work on the ‘search functionality’ – for that we need to fix the url pattern – go to ‘urls.py’ of ‘store’ app. Here we also included “category/” for products\_by\_category & for product\_detail so that it adds more meaning to the category slug & also makes search feature functional (prev. getting page not found error)

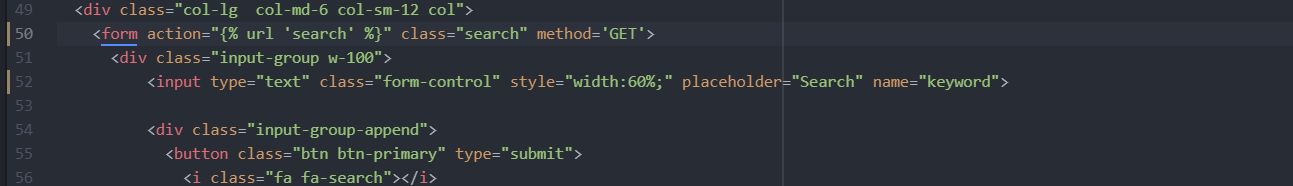


* Thereafter we create the view for the url in the ‘views.py’ of ‘store’



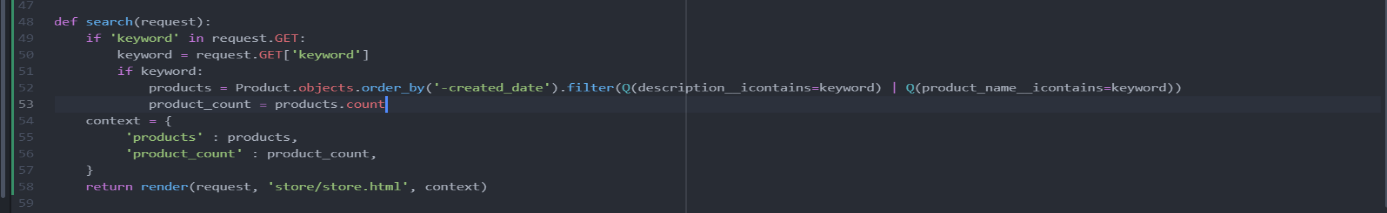


* Now we need to work on the ‘navbar.html’ because it is where the search bar is present.

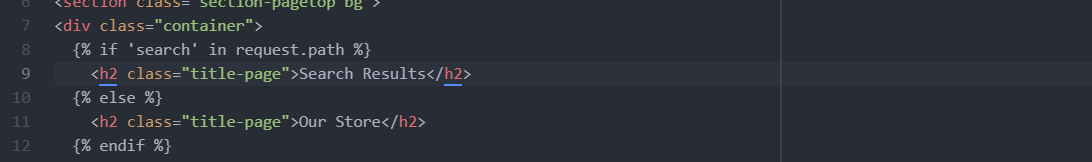


* Make sure the type= “submit”, because then only search function works
* Now we need to go to ‘views.py’ of store, because we need to receive what is coming from url

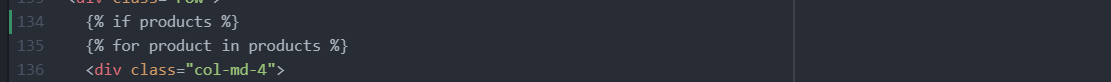


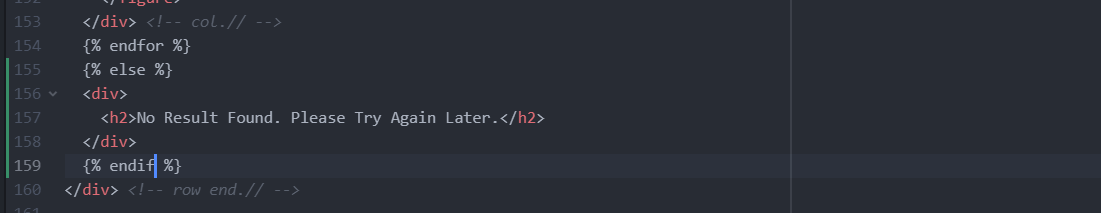


* Now we need to fix the display of ‘Our Store’ inside the search result – for this go to ‘store.html’



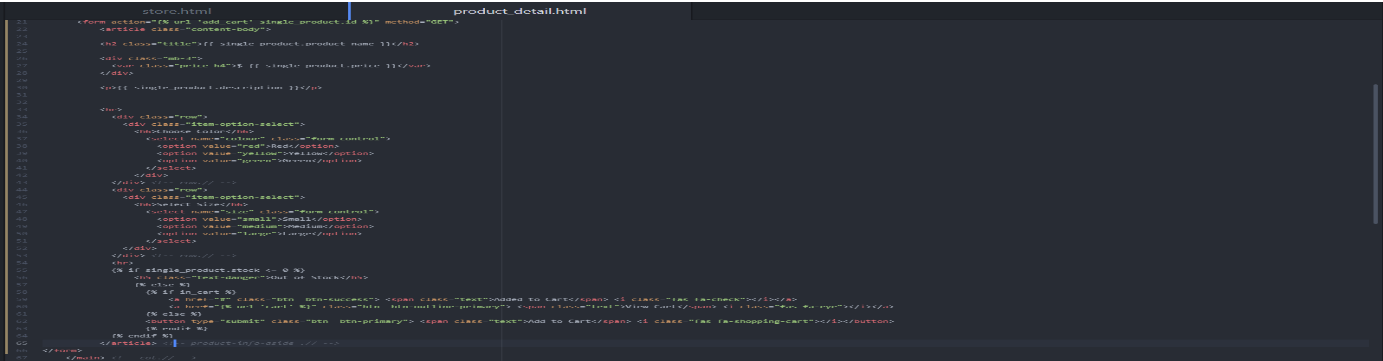
* Thereafter we need to fix the ‘No items to display’ message to the customer if there are no products available in that requirements.



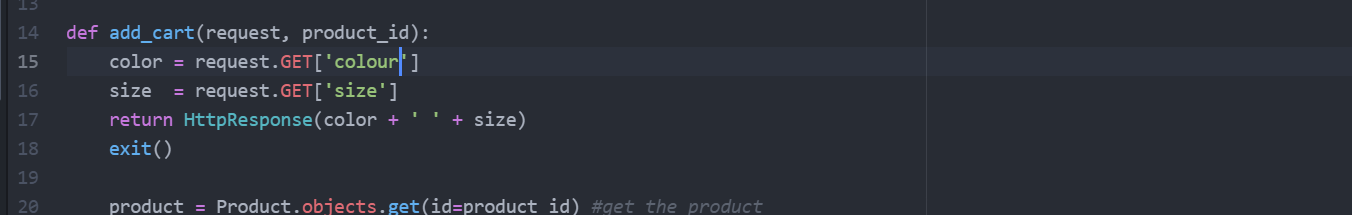


**Section – 10 [Starting Product Variations & Variation Manager]**

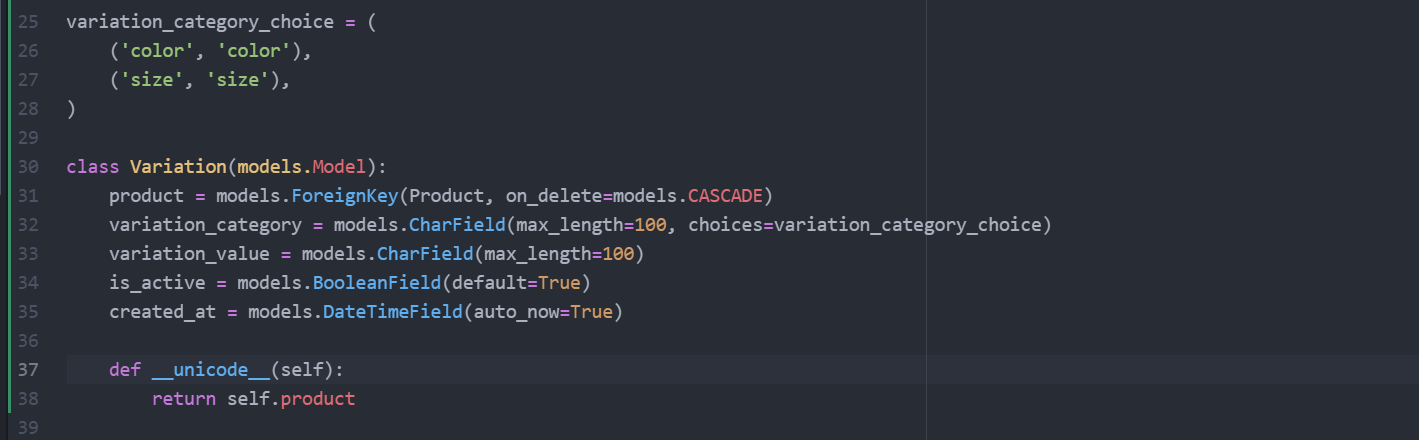
* **Product Variation Preparation:**
* Product variation functionality is usually very complex in nature & big giants like Amazon, Walmart etc., uses complex architectural technologies which contain large number of variations for their websites. Here we’re going to handle only 2 variations i.e., **colour and size** of the products. When we select a product – it should also contain the data like colour & size of the product which can be done through ‘Get request or Post request’ – which actually reminds the “form”.
* So go to **‘product\_detail’** page inside the ‘templates’ folder.

****

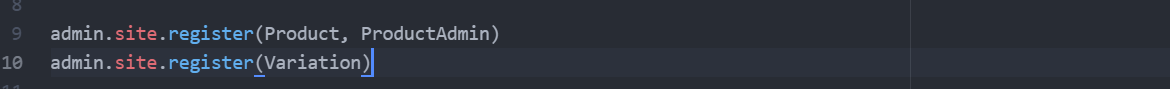
* Now when we click on the ‘Add to Cart’ button then the colour & size data should be sent to the url – so go to ‘**views.py’** of ‘carts’ & search for ‘add\_cart’ view.

****

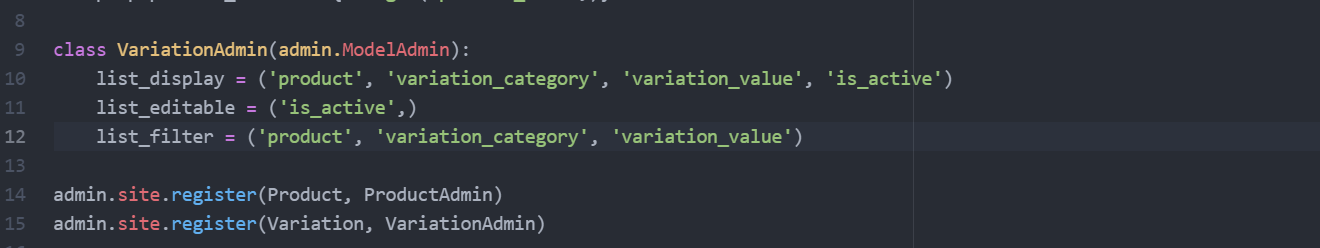
* **45 - Product Variation Model:**
* To create a variation model, we need few fields like add colour variations, variation category & product itself – so go to **‘models.py’** of ‘store.



* Now we want to display those variations done in models.py in the backend – so go to **‘admin.py’** of ‘store’



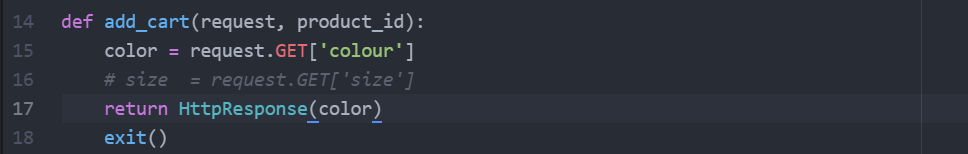
* Now in the admin site we can add the product variations – and when we enter the product variation, we don’t want to display the saved variations as variation 1,2… - so to deal with it we create a class in **admin.py** of store.



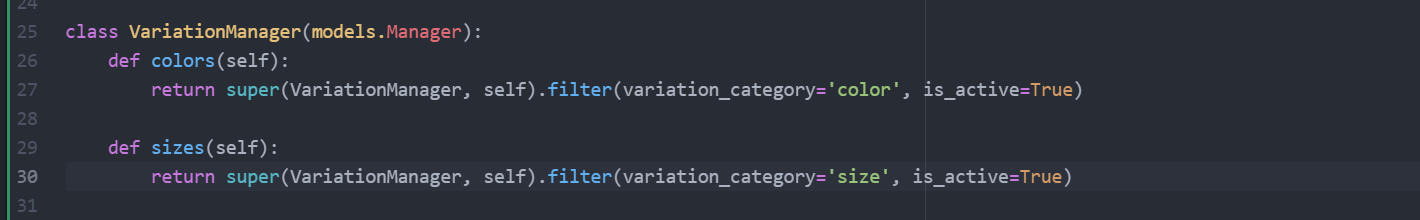
* **46 – Product Variable Fetch Dynamic Colour:**
* The colour and the size of the products are static values (i.e., Hard coded) – but we have the value for the colour in the database – so go to ‘**products\_detail.html’** file

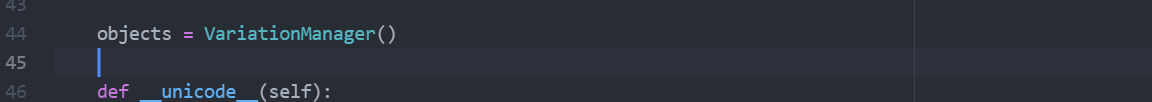
****

* When we selected the colour of a product and click ‘Add to cart’ then we’re getting ‘size error’ – so for time being we’ll comment the size commands in **‘views.py’** of ‘carts’

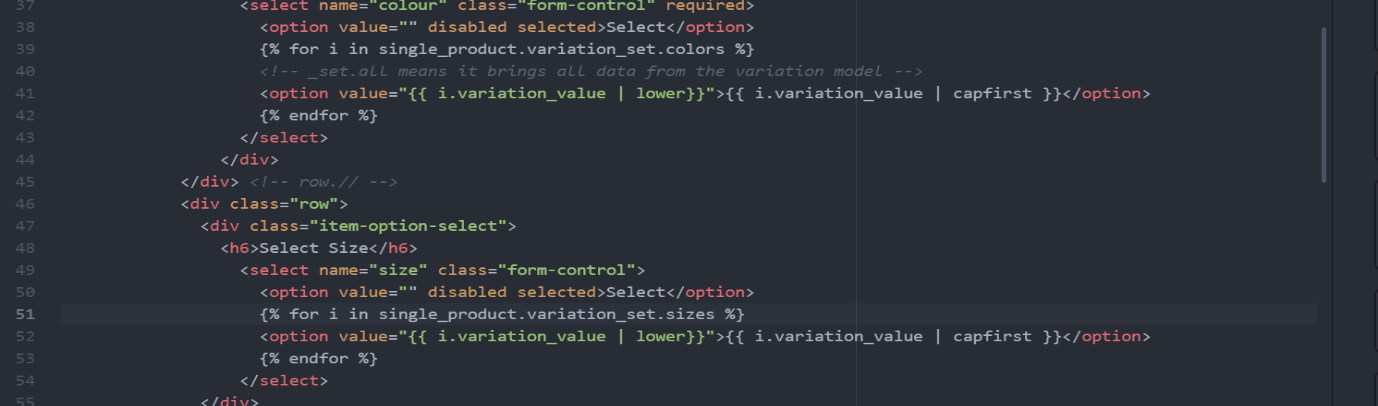


* **47 – Variation Manager and Post Request:**
* The ‘\_set.all’ command in the product\_detail is bringing both colour and size of the product – when we add a variation in the admin site with inclusion of size then the size option is getting displayed in the product details of the website – To deal with it go to ‘products\_detail.html’ and uncomment the commented section there.
* Next, we need to separate the colour & size to fix these sections inside the product detail page we need to create a variation manager in ‘models.py’ of ‘store’ folder.

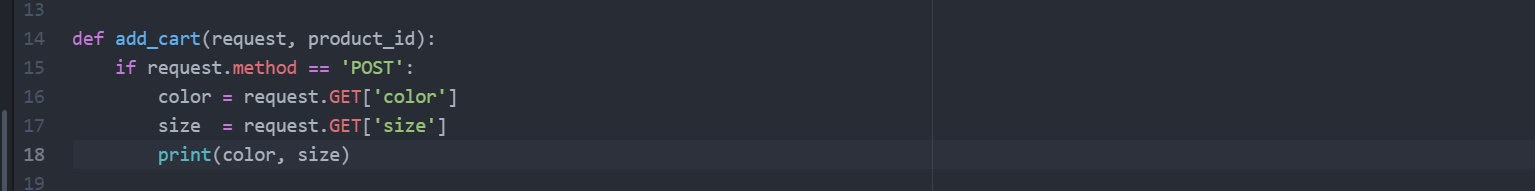




* Thereafter we’ll make the particular changes in the **‘products\_detail.html’** page inside the colour and size sections



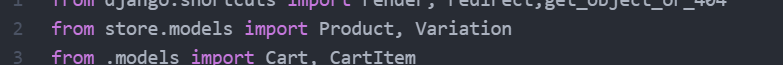
* We’ll also need to change the view of the colour & size of the page – in **‘models.py’** of ‘store’.

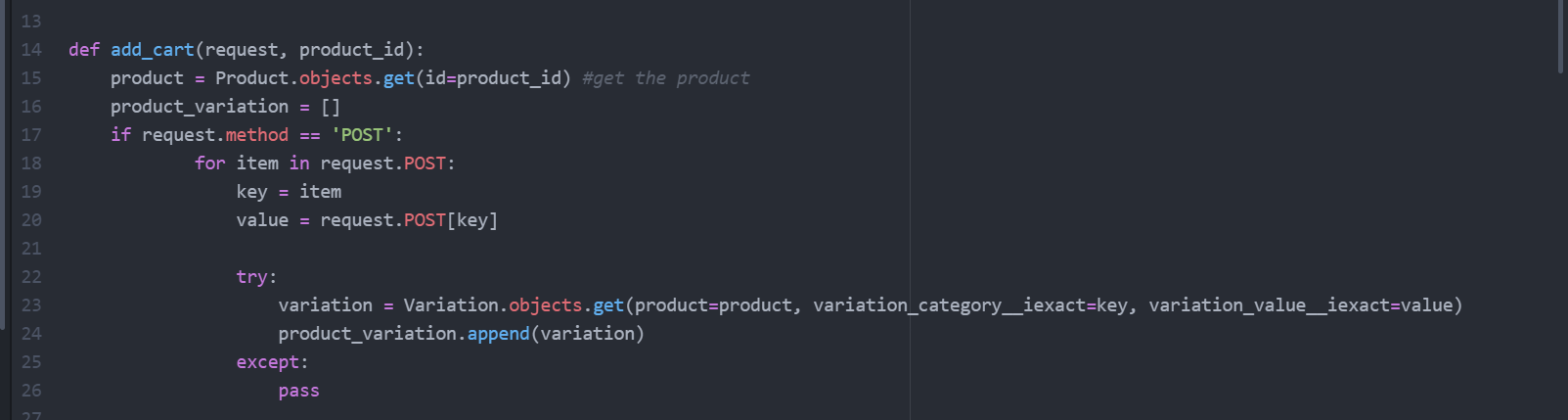


* Now we’re displaying the GET request in the url slug – but we actually needed POST request, so we change it in ‘**products\_detail.html’**

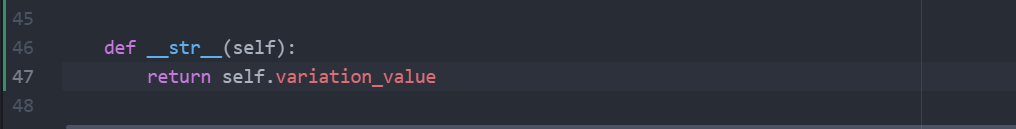


* With this the selected colour & size of a product are visible in the ‘git’ terminal when we stop the server, this is because we have included the ‘POST’ request.
* **48, 49 – Get the Instance of Variation Part-01 & 02:**
* So far we’re able to get the colour & size variations from request method ‘POST’ – But what if the brand comes in the place of colour & size – then we’ll get the error as we don’t have the brand – so we need to make the code little bit smarter to handle the situation – So go to ‘views.py’ file of the ‘carts’

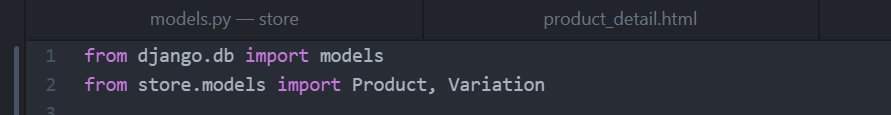


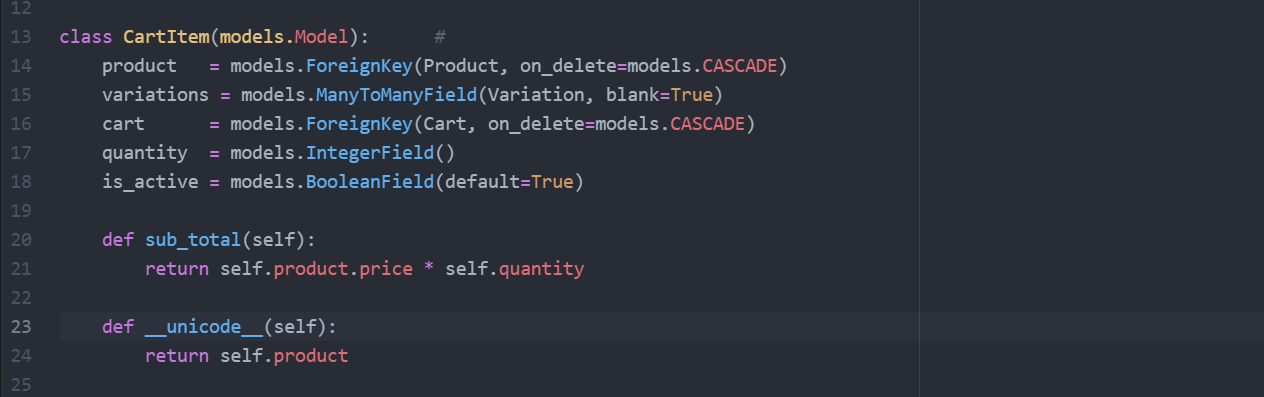


* Thereafter go to ‘models.py’ of the ‘store’ & modify the function with variation value.

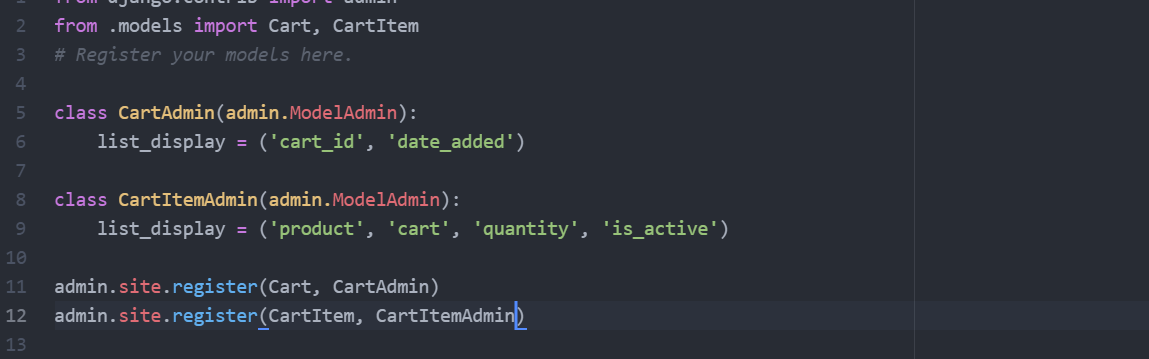


* With this the product variation values appear on ‘git’ terminal when server is stopped.
* Now go to CartItem model in the ‘models.py’ of the ‘carts’ folder



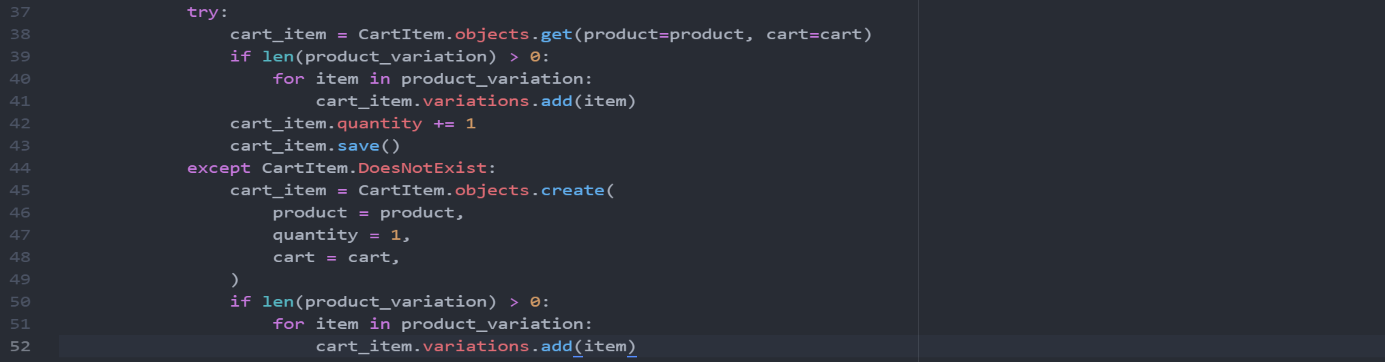


* Now the product variations are visible inside the Cart Items of the admin panel. Hereafter we need to modify some variations inside the ‘admin.py’ file of the ‘carts’ folder.



**Section-11 [Adding the Variation in Cart, Grouping Cart Item Variations]**

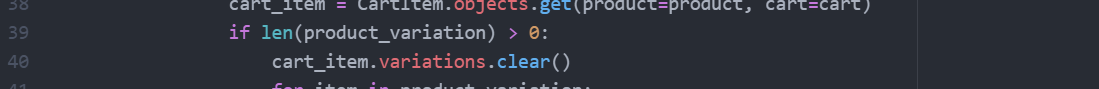
* **50 - Adding Variation in Cart Item:**
* As of till we have the product variation list, now we need to add the product variations inside the database – So go to add\_cart function inside the ‘views.py’ of the ‘carts’ folder.

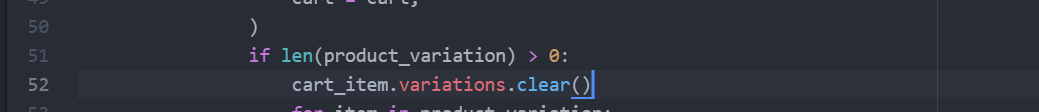


* Now the selected variations will be visible in the Cart Item of the particular product in the admin panel.
* Thereafter we need to change the ‘Added to cart’ option inside the product detail website, so that we can make any number of variations to the product – so go to ‘product\_detail.html’

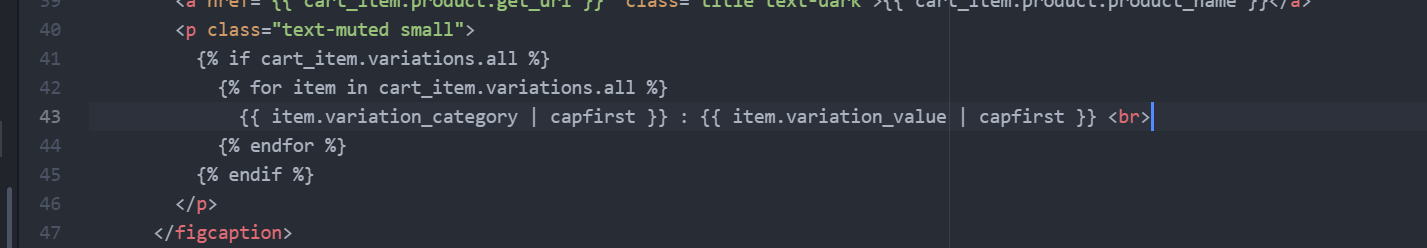


* Whenever we are adding any variations, then those are appearing in the shaded form with even the older variations still remaining same inside the Cart Items variations of the admin panel – so we’ll remove those variations & add, so that only newly added variations should only appear in shaded area– for this go to ‘views.py’ of ‘carts’.





* Now we need to make the variations display of a particular product inside the cart – go to ‘cart.html’ of templates.



* **51 – Grouping Cart Item Variations:**
* In the product cart page, whenever we’re adding a single product with multiple variations then only the latest variation is displaying in the cart with erasing the previous variations – So we need to create a new cart item whenever the same product with new variation is added – For this go to ‘views.py’ file of ‘carts’ folder. Here we’re also going to group the same kind of variations as a one product just by increasing the quantity of that particular product, and display the unsimilar product variations as separate cart item.