

Stock market trading app

→ 1 Technology used :

django.contrib.messages for message pop up

Django Framework

javascript for global search in table

Django REST Framework

Crispy forms

Bootstrap version 5.2 CDN for frontend

Django Template for frontend

Virtual environment

EMAIL_BACKEND = "django.core.mail.backends.console

For mongodb used django for connect with django project ORM

OS:Ubuntu

→ Requirement for setup django with versions

asgiref==3.5.2

beautifulsoup4==4.11.1

certifi==2022.9.14

charset-normalizer==2.1.1

configparser==5.3.0

Django==4.1.1

django-bootstrap4==22.2

django-crispy-forms==1.14.0

django-filter==22.1

django-rest-framework==3.13.1

django-rest-framework-simplejwt==5.2.0

django==1.3.6

dnspython==2.2.1

ez-setup==0.9

idna==3.4

Pyj WT==2.4.0

pymongo==3.12.3

pytz==2022.2.1

requests==2.28.1

soupsieve==2.3.2.post1

sqlparse==0.2.4

urllib3==1.26.12

→ Django REST Framework

1. API Created

→ Order Create view

→ order List view

→ order Retrieve view(Detail view)

→ order update view

→ order Delete view

→ Used generic view for api

→ url for api : <http://127.0.0.1:8000/api>

[al Help](#)

The image shows a VS Code editor window with a project structure at the top and a code editor below. The project structure includes files like `settings.py`, `requirements.txt`, `wsgi.py`, `asgi.py`, `urls.py`, `manage.py`, and `views.py`. The code editor displays the `settings.py` file, showing the `ALLOWED_HOSTS` and `INSTALLED_APPS` settings. The `INSTALLED_APPS` list includes `django.contrib.admin`, `django.contrib.auth`, `django.contrib.contenttypes`, `django.contrib.sessions`, `django.contrib.messages`, `django.contrib.staticfiles`, `trading_component`, `rest_framework`, `frontend`, and `crispy_forms`. The `MIDDLEWARE` list includes `django.middleware.security.SecurityMiddleware`. The terminal window at the bottom shows a log entry: `[21/Oct/2022 10:22:31] "GET /static/css/styles.css HTTP/1.1" 404 1801`.

```
19
20 ALLOWED_HOSTS = []
21
22
23 # Application definition
24
25 INSTALLED_APPS = [
26     'django.contrib.admin',
27     'django.contrib.auth',
28     'django.contrib.contenttypes',
29     'django.contrib.sessions',
30     'django.contrib.messages',
31     'django.contrib.staticfiles',
32     'trading_component',
33     'rest_framework',
34     'frontend',
35     'crispy_forms',
36 ]
37
38 MIDDLEWARE = [
39     'django.middleware.security.SecurityMiddleware',
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** JUPYTER SQL CONSOLE

[21/Oct/2022 10:22:31] "GET /static/css/styles.css HTTP/1.1" 404 1801

Setting of installed app

al Help

```
src > trading > settings.py > ...
65
66 WSGI_APPLICATION = 'trading.wsgi.application'
67
68
69 # Database
70 # https://docs.djangoproject.com/en/4.1/ref/settings/#databases
71
72 DATABASES = {
73     'default': {
74         'ENGINE': 'django',
75         'NAME': 'order_app',
76         'USER': 'root',
77     }
78 }
79
80
81 # Password validation
82 # https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators
83
84 AUTH_PASSWORD_VALIDATORS = [
85     {
```

Djongo package used for connect with mongo DB

```
al Help
[+] settings.py x [≡] requirements.txt [+] wsgi.py [+] asgi.py [+] urls.py .../trading [+] manage.py [+] views.py .../trading [▶] [□] [...]
src > trading > [+] settings.py > ...
111
112 # Static files (CSS, JavaScript, Images)
113 # https://docs.djangoproject.com/en/4.1/howto/static-files/
114
115
116 AUTH_USER_MODEL = 'trading_component.User'
117
118
119 STATIC_URL = 'static/'
120
121
122
123
124 EMAIL_BACKEND = "django.core.mail.backends.console.EmailBackend"
125 LOGIN_REDIRECT_URL="/home"
126 LOGIN_URL = "/login"
127 LOGOUT_REDIRECT_URL="/"
128
129
130
131
```

for Authuser used django auth User for Mail Django console emailbackend

```
al Help
[≡] requirements.txt [+] wsgi.py [+] asgi.py [+] urls.py .../trading [+] manage.py [+] views.py .../trading_component x [▶] [□] [...]
src > trading_component > [+] views.py > ...
1 from django.shortcuts import render,reverse,redirect
2 from .models import order
3 from .serializers import orderSerializer
4 from rest_framework.response import Response
5 from rest_framework import status, generics
6 import datetime
7
8
9
10
11
12
13
14 class orderView(generics.CreateAPIView):
15     serializer_class = orderSerializer
16     queryset = order.objects.all()
17
18     def post(self, request, format=None):
19         serializer = orderSerializer(data=request.data)
20         if serializer.is_valid():
21             serializer.save()
```

api GET and Post view

api GET and Post view

```
al Help
requirements.txt wsgi.py asgi.py urls.py .../trading manage.py views.py .../trading_component X
src > trading_component > views.py > ...
34
35
36 class OrderRetrieveView(generics.CreateAPIView):
37     serializer_class = orderSerializer
38     queryset = order.objects.all()
39     def create(self, request, *args, **kwargs):
40         queryset = order.objects.filter(pk=kwargs['pk']).first()
41         if queryset:
42             serializer = self.serializer_class(queryset)
43             return Response({"status": True, "data": serializer.data}, status=status.HTTP_200_OK)
44         return Response({"status": False, "data": "Data not found"}, status=status.HTTP_404_NOT_FOUND)
45
46
47 class OrderDetailUpdateView(generics.UpdateAPIView):
48     serializer_class = orderSerializer
49     def update(self, request, *args, **kwargs):
50         queryset = order.objects.filter(pk=kwargs['pk']).first()
51         if queryset:
52             serializer = self.serializer_class(queryset, data=request.data, partial=True)
53             if serializer.is_valid():
54                 serializer.save()
```

Detail api view

```
al Help
settings.py wsgi.py asgi.py urls.py .../trading manage.py views.py .../trading_component X
src > trading_component > views.py > ...
57     return Response({"status": False, "data": "Data not found"}, status=status.HTTP_404_NOT_FOUND)
58
59
60
61 class OrderDetailDeleteView(generics.DestroyAPIView):
62     serializer_class = orderSerializer
63     def delete(self, request, *args, **kwargs):
64         queryset = order.objects.filter(pk=kwargs['pk']).first()
65         if queryset:
66             serializer = self.serializer_class(queryset, data=request.data, partial=True)
67             if serializer.is_valid():
68                 queryset.delete()
69                 return Response({"status": True, "data": "Data deleted successfully"}, status=status.HTTP_200_OK)
70             return Response({"status": False, "data": serializer.errors}, status=status.HTTP_400_BAD_REQUEST)
71         return Response({"status": False, "data": "Data not found"}, status=status.HTTP_404_NOT_FOUND)
72
73
74
75
76
77
```

Delete api View

Username*

Password*

Login

[Don't have an account?](#) [Forgot password?](#)

Log in

[create Order](#)

select

Search

TYPE	ENTRY PRICE	QUANTITY
Sell	200	20

Called table through get API

[Go back to Home](#)

Add a new order

ENtry price

Quantity

Trade Type

Buy ▼

Submit

Add order through POST api

Username*

Required. 150 characters or fewer.
Letters, digits and @/./+/-/_ only.

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

Enter the same password as before, for verification.

Signup

Signup

Django frontend api call view through django template




The screenshot shows a code editor with several tabs: `views.py .../frontend`, `tests.py`, `settings.py`, `wsgi.py`, `asgi.py`, `urls.py .../trading`, and `manage.py`. The `views.py` tab is active, showing the following code:

```
30 }
31 return header
32
33
34
35
36
37
38
39 class SignupView(generic.CreateView):
40     template_name = "registration/signup.html"
41     form_class = CustomUserCreationForm
42
43     def get_success_url(self):
44         return reverse("login")
45
46
47 def login_page(request):
48     form_class=LoginForm
49
50
```

Below the code editor is a terminal window with the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER SQL CONSOLE
/home/kunal10x/Downloads/LMS-main/src/frontend/views.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
October 21, 2022 - 11:45:52
Django version 4.1.1, using settings 'trading.settings'
```



The screenshot shows the same code editor with the `views.py` tab active. The `login_page` function is now fully implemented:

```
46
47 def login_page(request):
48     form_class=LoginForm
49
50     message = ''
51     form=LoginForm
52     if request.method == 'POST':
53         form = form_class(request.POST)
54
55         if form.is_valid():
56             user = authenticate(
57                 username=form.cleaned_data['username'],
58                 password=form.cleaned_data['password'],
59             )
60             if user is not None:
61                 login(request, user)
62                 message = 'Hello {user.username}! You have been logged in'
63                 return redirect("/")
64             else:
65                 message = 'Login failed!'
66     return render(request, 'registration/login.html', context={ 'message': message, 'form':form})
67
68
```



```
views.py .../frontend X tests.py settings.py wsgi.py asgi.py urls.py .../trading manage.py ▶ ▾ □ ...
src > frontend > views.py > ...
70
71 @login_required
72 def index(request):
73     response=requests.get('http://127.0.0.1:8000/api', params=request.GET).json()
74     return render(request,'orders/order_list.html',{'response':response})
75
76
77
78 def home(request):
79     response=order.objects.all()
80
81     print("Response :",response)
82     return render(request,'books/book_list_student.html',{'response':response})
83
84
```

```
views.py .../frontend X tests.py settings.py wsgi.py asgi.py urls.py .../trading manage.py ▶ ▾ □ ...
src > frontend > views.py > orderview
79     return render(request, 'books/book_list_student.html',
80                   {'response': response})
81
82
83 def orderview(request):
84     if request.method == 'POST':
85         r = requests.post('http://127.0.0.1:8000/api', params=request.POST)
86     else:
87         r = requests.get('http://127.0.0.1:8000/api',
88                         params=request.GET).json()
89     return render(request, 'orders/order_list.html', {'response': r})
90
91
```

```
views.py .../frontend X tests.py settings.py wsgi.py asgi.py urls.py .../trading manage.py ▶ ▾ □ ...
src > frontend > views.py > ...
133
134
135 def delete_view(request, pk):
136     context = {}
137     order_data = order.objects.get(id=pk)
138
139     if request.method == "POST":
140
141         order_data.delete()
142
143         messages.info(request, "You have successfully deleted order")
144         return redirect("/")
145
146     return render(request, "orders/order_delete.html", context)
147
```

views.py .../frontend X tests.py settings.py wsgi.py asgi.py urls.py .../trading manage.py

src > frontend > views.py > orderview

```
92 def create_view(request):
93     r = ''
94     entry_price = request.POST.get('entry_price')
95     quantity = request.POST.get('quantity')
96     trade_type = request.POST.get('trade_type')
97     # entry_price = request.POST['entry_price']
98     payload = {
99         'entry_price': (entry_price),
100         'quantity': (quantity),
101         'trade_type': trade_type,
102         'user': request.user.id
103     }
104
105     if request.method == 'POST':
106
107         result = requests.post('http://127.0.0.1:8000/api',
108                                data=json.dumps(payload),
109                                headers=get_header())
110         return redirect("/")
111
112     return render(request, 'orders/order_create.html', {'response': r})
113
```

