

A large, semi-transparent Bitcoin logo watermark is positioned on the left side of the image, partially overlapping the central text box.

ⓑ BTC EXCHANGE

Encoding a platform where
you can trade bitcoins.

Requirements

- The platform must have an endpoint to manage user registration and access.
- Automatically assign 1 to 10 bitcoins to each user.
- Each user can post one or more sales or purchase orders of a certain amount.
- At the time of publication, if the purchase price of the order is equal to or greater than the selling price of any other user, match the transaction and mark both orders as filled.
- Provide an endpoint to get all active buy and sell orders.
- Provide an endpoint to calculate the total profit or loss from each user's trades.
- Assume that the platform in question is totally free for users and does not retain any type of commission on operations.

Useful links

Here are the various project links:

- [GitHub repo](#)
- [Website link](#)

Create a new user to access all features

Code

The site was written using **Django** (*Python*-based framework), with **MongoDB** as a *noSQL database*.

It was deployed with **Heroku** and **Microsoft Azure** was used as cloud for static and media files.




Django project structure

The Django project name is *Exchange* it consists of two applications:

- *app*
- *user*

Inside the project folder there are also:


- the *backend* folder containing the .py file that manages static and media files in the Microsoft Azure cloud.
- the *static* folder where there are all the static files, such as site images and the .css file.
- the *templates* folder where there are some .html files.



```
▼ Exchange
  > .env
  > .git
  > app
  > backend
  > Exchange
  > static
  > templates
  > user
  ◆ .gitignore
  🐍 manage.py
  📁 Procfile
  ⓘ README.md
  ≡ requirements.txt
  ≡ runtime.txt
```

app

In the *app* folder there is the *templates* folder with the .html files for the exchange page and there are the .py files.



```
▼ app
  > __pycache__
  > migrations
  > templates
  • __init__.py
  • admin.py
  • apps.py
  • decorators.py
  • forms.py
  • market.py
  • models.py
  • tests.py
  • urls.py
  • views.py
```


forms.py

```
from django import forms

# Limit orders generation form
class OrderForm(forms.Form):
    price = forms.FloatField(label='Price ($)')
    quantity = forms.FloatField(label='Quantity')

    def clean(self):
        cleaned_data = super().clean()
        price = self.cleaned_data.get('price')
        quantity = self.cleaned_data.get('quantity')
        if price < 0:
            raise forms.ValidationError('')
        if quantity < 0:
            raise forms.ValidationError('')
        return cleaned_data

# Market orders generation form
class MarketOrderForm(forms.Form):
    quantity = forms.FloatField(label='Quantity')

    def clean(self):
        cleaned_data = super().clean()
        quantity = self.cleaned_data.get('quantity')
        if quantity < 0:
            raise forms.ValidationError('')
        return cleaned_data
```

The *forms.py* file contains:

- the *OrderForm* class which represents the form for placing limit orders.
- the *MarketOrderForm* class which represents the form for placing market orders.

market.py

```
import requests

# Class to get info about BTC from the Coinmarketcap API
class Market:
    def __init__(self):
        self.url = 'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest'

        self.params = {
            'start': '1',
            'limit': '1',
            'convert': 'USD'
        }

        self.headers = {
            'Accepts': 'application/json',
            'X-CMC_PRO_API_KEY': '0535ae25-81e9-4e5d-8df1-8c6a20820c16'
        }

    def updated_data(self):
        database = requests.get(
            url=self.url, headers=self.headers, params=self.params).json()
        bitcoin_price = round(database['data'][0]['quote']['USD']['price'], 8)
        return bitcoin_price
```

The *market.py* file contains the *Market* class which it get info about BTC from the Coinmarketcap API.

models.py

```
# Class that attributes IP and login date
class IPAddress(models.Model):
    pubDate = models.DateTimeField('date published')
    ipAddress = models.GenericIPAddressField()

# Create user profile
class Profile(models.Model):
    _id = ObjectIdField()
    user = models.ForeignKey(User, on_delete=models.CASCADE)

# Create a wallet related to the user with the amount of BTC and USD
class Wallet(models.Model):
    _id = ObjectIdField()
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    btcWallet = models.FloatField(default=0)
    usdWallet = models.FloatField(default=0)
    startValue = models.FloatField(default=0)

    def __str__(self):
        text = f"Wallet n. {self._id}. User owner: {self.user}"
        return text

# Create order related to the user
class Order(models.Model):
    _id = ObjectIdField()
    profile = models.ForeignKey(User, on_delete=models.CASCADE)
    status = models.CharField(max_length=5)
    type = models.CharField(max_length=9)
    price = models.FloatField(default=None)
    quantity = models.FloatField()
    created = models.DateTimeField(auto_now_add=True)
    modified = models.DateTimeField(auto_now=True)

    def get_id(self):
        return self._id
```

The *models.py* file contains:

- the *IPAddress* class which creates an object in the db with the current IP and date.
- the *Wallet* class which create a wallet related to the user with the amount of BTC and USD.
- the *Order* class which create order related to the user.

view.py

Homepage view

```
# Homepage's view
@login_required(login_url='user:login')
@csrf_exempt
def homePageView(request):
    data = Market()
    currency = data.updated_data()

    # Stores the last IP that have logged in to the platform as admin, shows a warning
    # message when this is different from the previous one
    checkIp = None
    if request.user.is_staff:
        dbIp = IPAddress.objects.all().values().last()
        actualIp = getActualIP(request)

        if not dbIp:
            addIp(actualIp)
        else:
            if actualIp != dbIp['ipAddress']:
                addIp(actualIp)
                checkIp = True

    return render(request, "homepage.html", {"currency": currency, "checkIp": checkIp})
```

This view shows the homepage, i.e. the page that is shown after logging in.

in this view we check the IP: a warning message is shown if the IP of a logged in admin user is different from the IP of the previously logged in admin user.

The `login_required` decorator prevents non-logged-in users from accessing the page.

This view returns a page showing some information about BTC.

Exchange page view

[GitHub link](#)

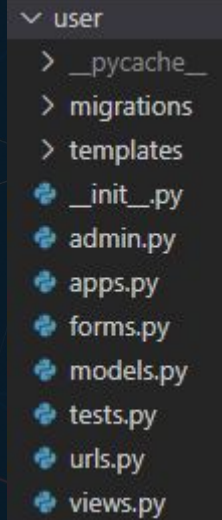
This view manages the creation of *limit orders* and matches them with the *market orders* that are sent.

In this view are also processed data useful for the creation of the DOM in the page of the site.

The view return a web page where on the left we have the order creation forms, in the center we have a BTC TradingView chart and on the right we have the DOM where all active limit orders are shown.

user Application

In the *user* folder there is the *templates* folder with the .html files for the login, register and account page, and there are the .py files.



forms.py

```
# Form for the user registration
class NewUserForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ["username", "email", "password1", "password2"]

    def save(self, commit=True):
        user = super(NewUserForm, self).save(commit=False)
        user.email = self.cleaned_data['email']
        if commit:
            user.save()
        return user
```

The *forms.py* file contains the *NewUserForm* class which represents the form for the user registration.

view.py

Register page view

```
# Register page's view
@unauthenticated_user
@csrf_exempt
def registerPageView(request):
    data = Market()
    currency = data.updated_data()

    form = NewUserForm()
    if request.method == 'POST':

        form = NewUserForm(request.POST)
        if form.is_valid():
            user = form.save()

            # Profile creation
            newUserProfile = Profile(
                user=user,
            )
            newUserProfile.save()

            #Wallet Creation
            newUserWallet=Wallet(
                user=user,
                btcWallet = round(random.uniform(1, 10), 8),
                usdWallet = round(random.uniform(50000, 150000), 2),
            )

            newUserWallet.startValue = newUserWallet.usdWallet + (newUserWallet.btcWallet*currency)
            newUserWallet.save()

            return redirect('user:login')
        else:
            messages.info(request, 'Check that you have filled in all the required fields correctly.')

    return render(request, 'user/register.html', {"form": form})
```

This view takes the POST requests from the registration page and creates a new user with the data entered in the form, it assigns him a random value between 1 and 10 BTC and between 50k and 100k\$.

The `unauthenticated_user` decorator ensures that if a logged in user tries to enter the registration page by typing the link into the browser's search bar, he will be redirected to the Homepage.

Login page view

```
# Login page's view
@unauthenticated_user
@csrf_exempt
def loginPageView(request):
    form = AuthenticationForm(request, data=request.POST)
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            return redirect('app:homepage')
        else:
            messages.info(request, 'Wrong username or password')

    return render(request, 'user/login.html', {"loginForm": form})
```

This view takes the POST requests of the login page and, if the credentials are correct, it redirects the user to the Homepage.

The `unauthenticated_user` decorator ensures that if a logged in user tries to enter the registration page by typing the link into the browser's search bar, he will be redirected to the Homepage.

Account page view

```
# Account page's view
@login_required(login_url='login')
@csrf_exempt
def accountPageView(request, id):
    data = Market()
    currency = data.updated_data()

    username = request.user.username

    userWallet = get_object_or_404(Wallet, user_id=id)

    btcBalance = userWallet.btcWallet
    usdBalance = userWallet.usdWallet

    actualValue = usdBalance + (btcBalance * currency)
    startValue = userWallet.startValue

    delta = actualValue - startValue

    return render(request, 'user/account.html', {
        'username': username,
        'delta': delta,
        'btcBalance': btcBalance,
        'usdBalance': usdBalance
    })
```

This view returns a page showing the total profit or total loss since the portfolio was created.

It also gives us information on the balance of the BTC and USD wallet.