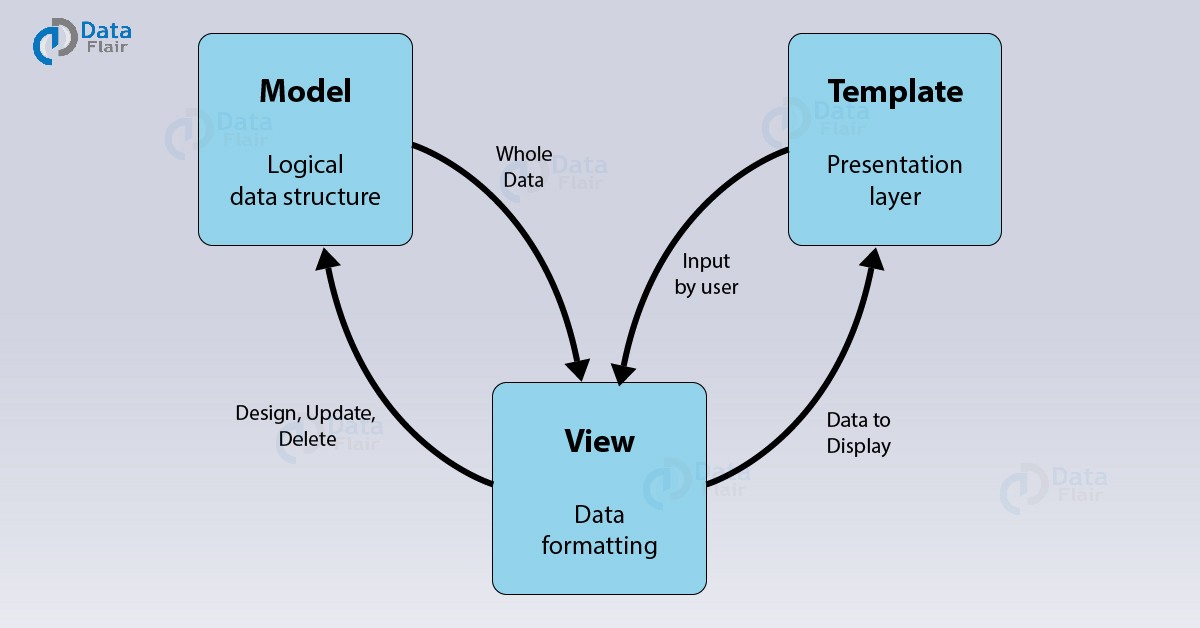
**Django REST Framework**

One of the main reasons why we picked Django to build the web service for our project was because we have coded all our backend in Python. As Django is built on Python, we figured it would be more efficient to use Django as the integration would be smoother. Also, Django has a lot to offer and has the ability to accommodate any modern web apps structures. Stacking Django and JavaScript web frameworks to build modern web apps is one of the best ways to stack backend and frontend frameworks. The web server developed by our team runs on the local host as of now and it facilitates the execution of all the project requirements right from the data collection phase, invoking the prediction algorithms, maintaining the admin and user databases and up until managing the user interface.

**Django MVT (Model, View, Template) Architecture**



The Django REST framework has three major modules, which are the Model, View and Template.

**Model:**

This module is a representation or an interface layer for the underlying data in the database. We can access the data and perform operations on this data in the database by using the Models. Each database table is represented by a single Model which is a Class object and the columns in the table are represented by the attributes of the Model class.

**Template:**

This module comprises all the static HTML and CSS files which will be displayed to users on the User Interface. When the user provides some inputs on the interface, it invokes the corresponding functionalities from the View template.

**View:**

This module is used to code various functionalities which will be used by the user on the User Interface. These functionalities will be invoked whenever a user hits an endpoint which is linked to the specific functionality and the output is rendered back to the HTML pages in the Template module.

**Interaction between the three modules:**

The View module takes as input the data from the Models and it performs various operations on the data like insert, modify or delete and updates the database accordingly using the Models. The View module also takes the user inputs from the Presentation layer through the Templates and performs various functionalities and invokes the backend code and renders the output back to the users through the Template module.

**Implementation**

Firstly, we created a virtual environment called ‘virtualstockenv’ for our application.

What is a Virtual Environment?

At its core, the main purpose of Python virtual environments is to create an isolated environment for Python projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has.

The command to create the Python virtual environment in Windows is:

$ python -m venv <name\_of\_virtual\_environment>

In our case it was,

$ python -m venv virtualstockenv

We then activated the virtual environment

$ cd virtualstockenv

$ cd Scripts

$ activate.bat

This command will activate the virtual environment and get it running.

We then installed the modules necessary for Django to function.

$ python -m pip install django djangorestframework

We then created a project called ‘virtualstockenv’ inside the virtual environment and an application called ‘stockapp’ inside this project.

$ django-admin startproject virtualstockenv

$ cd virtualstockenv

python manage.py startapp stockapp

We then made changes in the virtualstockenv/settings.py file as below by adding additional lines.

INSTALLED\_APPS = [

'virtualstockenv',

'stockapp.apps.StockAppConfig',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'users.apps.UsersConfig',

'rest\_framework',

'crispy\_forms',

]

The settings.py file is an important component of the server. It holds sensitive data used by the server. The information about the configurations necessary for the application is stored in this file. It also holds the information of the database like name of the database, username, password, host name and the port number that can be used by the web service we created to access our local database. This file also holds the information of the time zones.

We need to be able to use the stock data we have stored in our database during the Data Collection phase. For this, we first need to connect to the MySQL database management system. To facilitate this, we need to update the same settings.py file as below.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': "<database\_name>",

'USER': "<username>",

'PASSWORD': "<password>",

'HOST': "localhost",

'PORT': "3306",

}

}

Next step we followed was generating the models for our data. A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you’re storing. Django web applications access and manage data through Python objects referred to as models. Generally, each model maps to a single database table. Models define the structure of stored data, including the field types and possibly also their maximum size, default values, selection list options, help text for documentation, label text for forms, etc.

We can either create new models or already use the existing database schema to let Django to automatically generate the models for us using the existing schema. As we already had the database schema and the data set up, we used the inspectdb command of Django to get automatically generated models and saved these models into a file called models.py.

$ python manage.py inspectdb > models.py

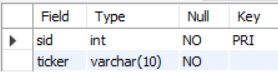
This command generated all the models into the models.py file. We then moved this file from /virtualstockenv to /virtualstockenv/stockapp to be able to use it for data access operations.

We then ran the migrate command to install any extra needed database records such as admin permissions and users and content types

$ python manage.py migrate

In this way, all the data we had in our database initially was then linked to the Django Framework such that we could access the data in all the database tables from the application through the Models generated for each table.

For example: The schema of our stocks table is as below.



The model generated by Django for the above table is as follows:

class Stocks(models.Model):

sid = models.IntegerField(primary\_key=True)

ticker = models.CharField(max\_length=10)

class Meta:

managed = False

db\_table = 'stocks'

The sid and ticker attributes of the Stocks Model correspond to the fields in the stocks table as the value of db\_table = ‘stocks’. These fields are mapped to IntegerField and CharField pertaining to the data types of these fields in the stocks table in the database.

We then create an admin account for our application by executing the below command. The admin is responsible for maintaining the application and he has extra privileges to the application and the data compared to the other application users. He can also add and delete user accounts from his account.

python manage.py createsuperuser

This command asks us to enter a username and password for the admin account. Upon entering these details, the admin account is created.

We then make use of the urls.py file in virtualstockenv/stockapp/ folder to register the URL endpoints for our application.

A sample endpoint registered in our urls.py file is:

path('Contact/', views.contact, name='contact'),

The first argument represents the endpoint which is localhost:<port\_number>/stockapp/Contact. The second argument represents the function which will be invoked when this endpoint is hit. This functionality will be a part of the views.py file in the virtualstockenv/stockapp/ folder. The third argument represents the name for this endpoint so that we refer to it in the other files on the service.

All the functionalities which are invoked when the application endpoints are hit are stored in the views.py file. Some of these functionalities also invoke the backend algorithms for predictions.

We have scheduled the data collection code and the code for thresholds to run at the specified intervals using apscheduler. To use Django for scheduling, we created a new folder called stockupdate in virtualstockenv folder. We placed the data\_collection.py file in this folder. We then created a new python file called updater.py in the same folder to write the code for scheduling. The scheduling code is written in this file under the start() method.

By using this scheduling code, whenever we start the server, Django will automatically start the scheduler and invokes the data fetch code and thresholds code at the specified intervals.

* We invoke the get\_real() function of data\_collection.py file every minute between 10:00 AM and 4:00 PM on stock market working days.
* We invoke the get\_hist() function of data\_collection.py file once at 5:00 PM on all stock market working days.
* We invoke the check\_thresholds() every minute similar to get\_real().

To render the output from the functionalities in the views.py file to the UI HTML pages, we create a table called templates inside stockapp folder and place all our HTML files in that location. We return the output from a function in the views.py to a HTML page in the following way:

return render(request, 'stockapp/<file\_name.html>', context)

The context is a python dictionary object which will contain the output which we want to display on the HTML file which we specify as the second parameter in the code above.

We also need to send data like user inputs from the HTML page to the functionalities in the views.py file in the form of POST requests so that we will be able to make use of this data and return the desired output to the user. Example: To get the stock name input by the user labelled with the name ‘stock’ on the UI, we use the following code:

stock\_name = request.POST.get('stock')

We then run the server by running the command from the virtualstockenv/ folder:

python manage.py runserver 8100

This will start the server on the localhost on the port 8100. We can then hit the URL <http://localhost:8100> and we will be redirected to the user landing page.

The major method signatures for our service are:

* **registermail(user\_email, email\_message)**

This method is used to send an email to the user once he registers on our website by creating an account.

Output: Email sent to user\_email with content = email\_message

* **get\_hist()**

Fetches the historical stock data for all the stocks in the database and updates the historical table with the data on every weekday after the stock market closes for the day.

* **get\_real()**

This method fetches the real time stock data for all the stocks in the database and updates the real-time table with data every minute on every weekday between the stock market timings.

* **long\_term\_nn(stock\_name, number\_of\_days)**

This method performs long term prediction using Neural Networks algorithm and takes as input the stock name and the number of days to find out the predictions for.

Output: List of predictions of length = number\_of\_days and Buy/Sell suggestion for stock\_name

* **short\_term\_pcf(stock\_name, number\_of\_minutes)**

This method performs the short term prediction using Polynomial Curve Fitting algorithm and takes as input the stock name and the number of minutes to find out the predictions for.

Output: List of predictions of length = number\_of\_minutes and Buy/Sell suggestion for stock\_name

* **calc\_ema(list\_of\_prices)**

This method calculates the Exponential Moving Average value for the input list of prices. These prices include the predicted prices as well.

* **calc\_rsi(list\_of\_prices)**

This method calculates the Relative Strength Index value for the input list of prices. These prices include the predicted prices as well.

* **calc\_macd(ema\_short, ema\_long)**

This method calculates the Moving Average Convergence/Divergence value by subtracting the ema\_long value from ema\_short.

* **thresholds()**

This method is used to notify users about any price changes for any of the stocks they are interested in trading.

Output: Email notifications sent to users’ email addresses in case of stock price reaching desired price for the user.

* **run\_queries(stock\_name)**

This method uses data for stock\_name and calculate all the values required for the queries like minimum and maximum price in 10 days, minimum, average and maximum price in 1 year and so on.

**References:**

[https://miro.medium.com/max/2400/0\*8ZFh-CsrMi7bQG0O.jpg](https://miro.medium.com/max/2400/0*8ZFh-CsrMi7bQG0O.jpg)