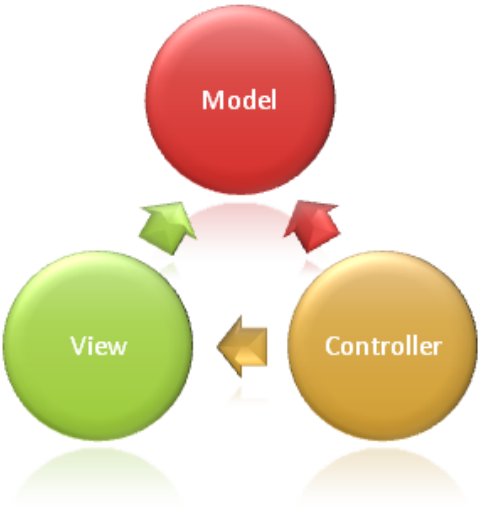
medium.com

Understanding the MVC pattern in Django - She Code Africa - Medium

Ada Nduka Oyom

4-5 minutes



In my previous post, i put up a tutorial for beginners on how to get started with Django, here. And in between i talked about how Django embraces the popular **M**odel — **V**iew — **C**ontroller pattern with a new twist and how it can be a bit difficult grasping it for first timers (Yes, it took a while for me to grasp it too)

So In this post I'm going to be giving a deeper insight as to how the MVC pattern works in general and how it can be related to Django in scope.

The MVC pattern is a software architecture pattern that separates data presentation from the logic of handling user interactions(in other words, saves you stress:), it has been around as a concept for a while, and has invariably seen an exponential growth in use since its inception. It has also been described as one of the best ways to create client-server applications, all of the best frameworks for web are all built around the MVC concept

To break it down, here's a general overview of the MVC Concept;

Model: This handles your data representation, it serves as an interface to the data stored in the database itself, and also allows you to interact with your data without having to get perturbed with all the complexities of the underlying database.

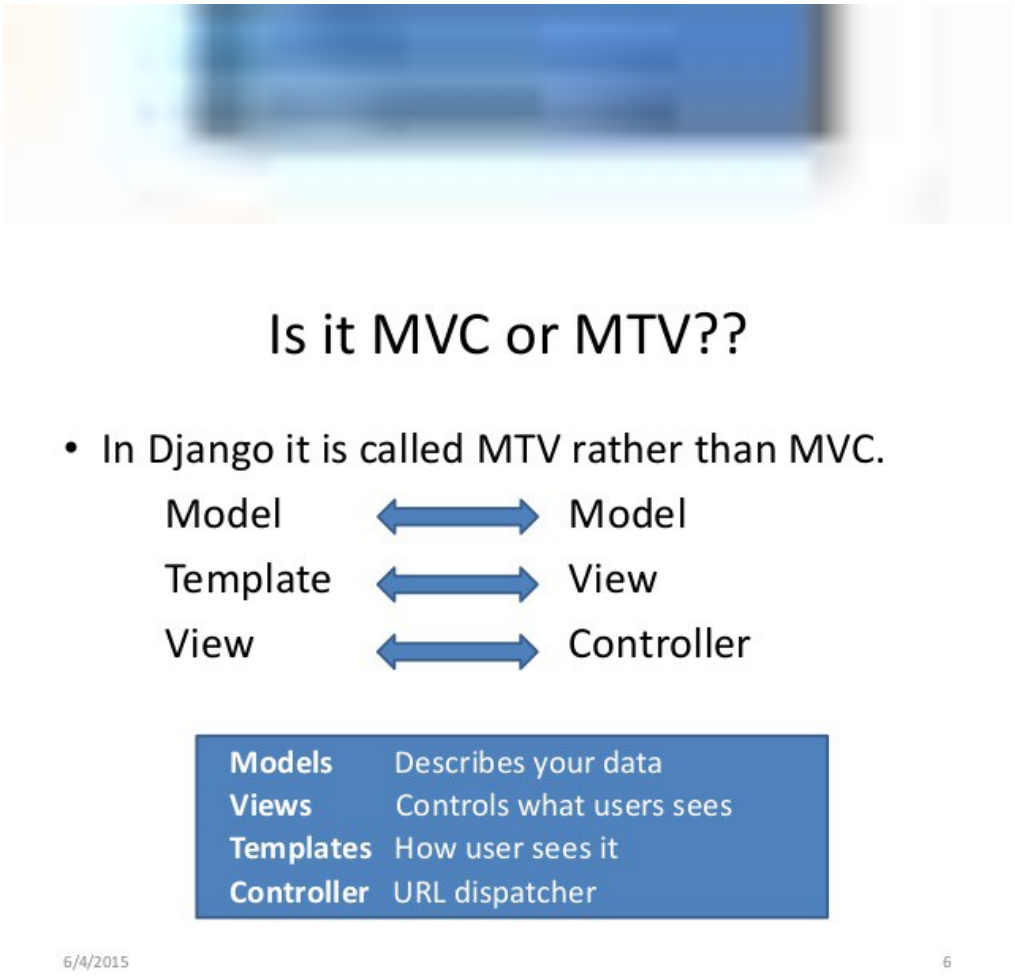
View: As the name implies, it represents what you see while on your browser for a web application or In the UI for a desktop application.

Controller: provides the logic to either handle presentation flow in the view or update the model's data i.e it uses programmed logic to figure out what is pulled from the database through the model and passed to the view,also gets information from the user through the view and implements the given logic by either changing the view or updating the data via the model , To make it more simpler, see it as the engine room.

Now that we understand the general concept of the MVC, understanding how it is implemented in different frameworks can be another task as some frameworks(Django inclusive) like to implement this same functionality in another way making it a bit difficult understanding what actually happens at each layer.

How do we relate it to Our scope in Django?

Even with Django following the MVC pattern, it prefers to use it's own logic in the implementation, the framework considers handling the Controller part of the MVC itself and letting most of the good stuff happen in the **Model-Template-View**, this is why Django is mostly reffered to as the **MTV** framework.



In the MTV pattern:

Model: Just like the Model explanation in the MVC pattern , this also takes the same position as the interface or relationship between the data and contains everything related to data access and validation.

Template: This relates to the View in the MVC pattern as it is the presentation layer that handles the presentation logic in the framework and basically controls what should be displayed and how it should be displayed to the user.

View: This part relates to the Controller in the MVC pattern and handles all the business logic that throws down back to the respective templates.It serves as the bridge between the model and the template

The tiny difference that can constitute as the most confusing part in all this, is how Django suggests that the View should include the business logic instead of the presentation logic alone as it is in the standard MVC pattern and the Template to take care of all of the presentation logic alone while the MVC pattern does not include a

Template component at all. As a result of this, when compared to the standard MVC pattern, Django's design is also referred to as the ***Model-Template-View + Controller*** where Controller is often times omitted because it's already part of the framework.

Although it is really helpful understanding this pattern before delving into development, what matters at the end of it all, is you being able to get the job done, and thankfully Django helps provide an ecosystem geared towards programming efficiency :).

Found this article helpful? Don't forget to recommend and share :)