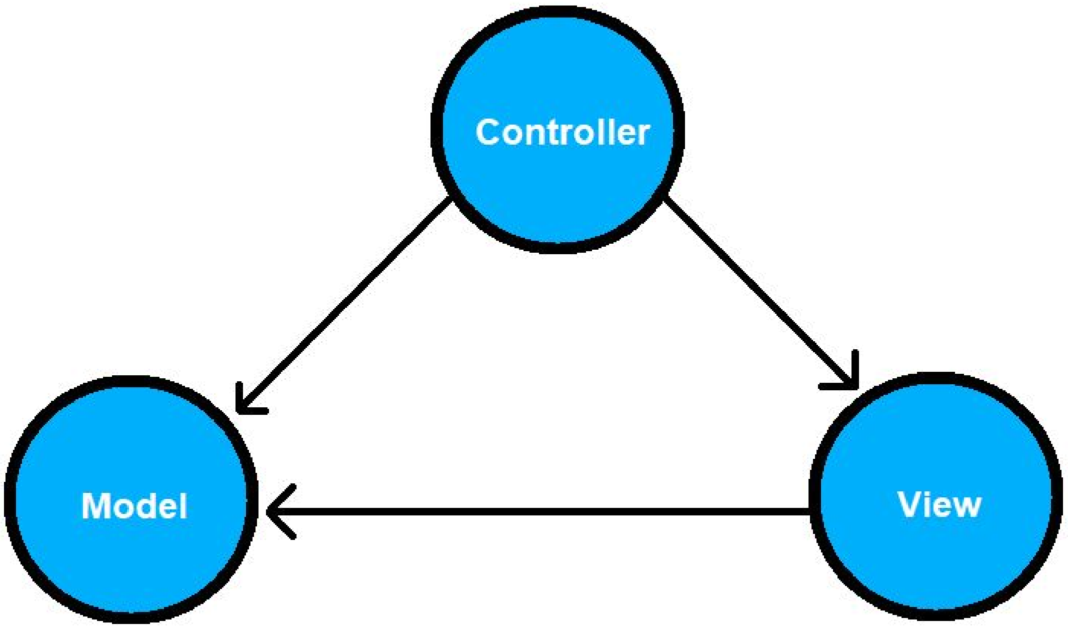
The Django Web Framework - Documentation

What is Django?

Django is a high-level Python web application framework that enables the rapid development of web applications. It is easy to use in comparison to other frameworks. It is a backend framework used to resolve problems of connectivity with databases, other server problems, etc so that a web developer need not write the same code for the similar modules for each website.

Django Architecture

Django is based on Model - View - Controller (MVC) architecture, which consists of three different parts:



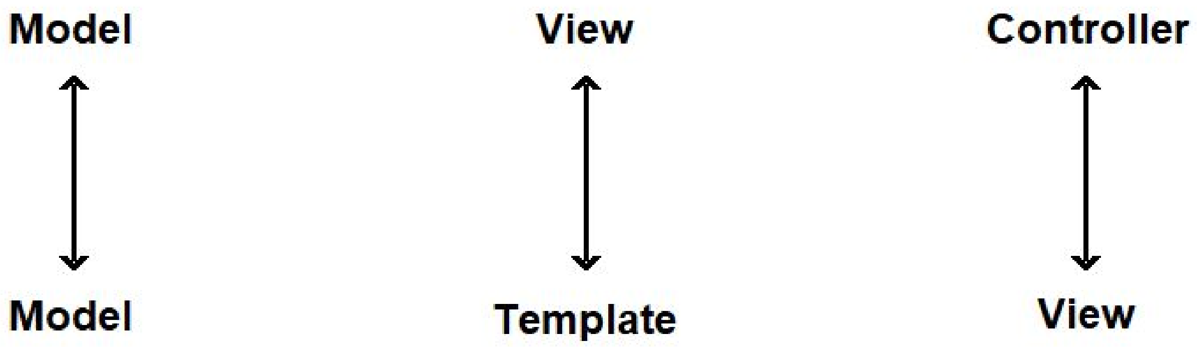
- **Model** - The Model is the logical data structure behind the complete application and represented by a database (generally relational database such as MySQL, Postgres).

- **View** - The View is the user interface that you see in the browser whenever you visit a website. They represent it through HTML/CSS/JavaScript files.
- **Controller** - The Controller is the middleman that connects the view and model together. It means the controller is the passing data from the model to the view.

For example -
A user will enter a URL in their browser, that request will go through the internet protocols (HTTP), to the server, which will call the Django. Django will then process the given URL path, and if the URL matches, it will call the Controller which will get an entry from your database and then render a View (HTML/CSS/JavaScript Web page).

MVT Pattern

Django is mainly an MVT (Model - Template - View) framework. It uses the terminology Templates for Views and Views for Controller. Thus Python code will be in views and models and HTML code will be in templates.



Why Learn Django?

- Django has Evolved Over Time
- Open-Source Technology
- One of the World's Best Software Community
- Django has Lots of Pre-Made Apps

Features

- Stability
- Excellent Documentation
- Highly Scalable
- Resolves Security Issues
- Utilities SEO

- Huge Library of Packages
- Allows Pragmatic and Robust Design

Installation

Virtual environment

Before installing Django, create a virtual environment (also called a virtualenv). Virtualenv will isolate Python/Django setup on a per-project basis. This means that any changes you make to one website won't affect any others you're developing.

```
pip install virtualenvwrapper-win
```

Make a virtualenv called project.

```
mkvirtualenv project
```

Installed and started virtualenv, now install Django.

```
pip install django
```

Create new project called Django

```
django-admin startproject Django
```

Create new app called calc in project Django

```
python manage.py startapp calc
```

Start the development server

```
python manage.py runserver
```

Example

Open the file calc/views.py and add the following Python code in it:

```
from django.shortcuts import render

def home (request):
    return render(request,'home.html',{'name':'Pooja'})
```

To call the view, we need to map it to a URL - and for this we need a URLconf.

To create a URLconf in the calc directory, create a file called urls.py.
In the calc/urls.py file add the following code:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
]
```

Now, point the root URLconf at the calc.urls module. In Django/urls.py, add an import for `django.urls.include` and insert `include()` an in the `urlpatterns` list.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('', include('calc.urls')),
    path('admin/', admin.site.urls),
]
```

For templates, create a new folder `template` in the root folder and create a html file like `home.html` in the template folder. In the template/home.html file add the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Django</title>
</head>
<body bgcolor="cyan">
      <h1>Hello {{name}} !</h1>
</body>
</html>
```

To point the template files, add path in Django/settings.py file. Add a path to the `DIRS`.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Start the server to verify it's working with the following command:

```
python manage.py runserver
```

Go to http://localhost:8000/ in your browser, and you can see the text "*Hello Pooja !*".

What is Django used for?

Django’s model makes use of a powerful ORM layer which simplifies dealing with the database and the data and accelerates the development process.

Without Object-Relational-Mapping, developers would have to create the tables themselves and define the queries or procedures which sometimes translates to the hefty amount of SQL that is prone to be complex and hard to track.

The ORM layer lets you write all the table definitions in simple python code, and it takes care of translating that to the appropriate query language chosen, and it also facilitates the CRUD operations.

Companies using Django

1. Instagram
2. Disqus
3. Spotify
4. YouTube

Advantages

1. **Batteries included**
   Django known as batteries-included framework i.e.it comes with lots of stuff out of the box, that you may or may not use depending on application. Instead of having to write your own code, just need to import the packages that you want to use.
   - Authentication with *auth package*
   - Admin interfacing with *admin package*
   - Session management with *Sessions package*

2. **Implemented in Python**
   The Django framework is implemented using the python language, so that gives it immense support in the backend. Python is quite powerful and is used in the implementation of scientific computations and high-level Artificial Intelligence.

3. **Community**
   Django’s community is one of the best things about it, they are helpful and working on making the framework more beginner-friendly and stabilizing the framework while adding new features.

4. **Scalable**
   Django allows you to take a lot of different actions regarding scalability, such as running separate servers for the database, the media and the application itself or even use clustering or load-balancing to distribute the application across multiple servers.

5. **Built-in Admin**
   Django offers an administrative interface.

Disadvantages

1. **Monolithic**
   Django has a certain set of files and pre-defined variables, and you need to learn about those before you create any project through Django. It has a certain way to define and perform tasks. It is a logical file structure and easy to learn. But, that makes it mandatory that you can't use your own file structure.

2. **Not for smaller projects**
   All the functionality of Django comes with lots of code. It takes server's processing and time, which poses some issues for low-end websites which can run on even very little bandwidth.

3. **Regular Expressions for URLs**
   Django uses RegEX to determine its URL routing models which makes the code bigger and makes twisted syntaxes. It uses a regular expression which is kind of a format of URL. These URLs look cleaner than the PHP, but they are more complex sometimes.

Comparison : **Django vs Flask**

1. Django is a Full-stack web framework that follows the batteries-included approach.
   Flask is a light-weight framework with minimalistic features.
2. In Django, Developers already have access to the most common features that make development faster.
   In Flask, developers can explore and keep control of the core of the application.
3. Django comes with a ready-to-use admin framework that can be customized. Flask doesn't have any such feature to handle administration tasks.
4. Django comes with a built-in template engine that saves a lot of development time. Flask's template engine Jinja2 is based on Django's template engine.
5. Django allows users to divide a single project into multiple small applications which makes them easy to develop and maintain.
   In Flask, Each project can be a single application and multiple models and views can be added to the single application.
6. The Django-admin tool is a built-in bootstrapping tool with which developers can build web applications without any external input. In flask, Admin features are not as prominent as in Django.
7. In Django, the built-in ORM system enables developers to use any database and perform common DB tasks without having to write long queries.
   With Flask, developers have to work with different databases by using ORM systems for Python and SQLAlchemy as the SQL toolkit. SQL queries have to be written for common tasks.