UNIT 1

Django Overview

4.1 Django Introduction

Django is a back-end server side web framework.

Django is free, open source and written in Python.

Django makes it easier to build web pages using Python.

What is Django?

Django is a Python framework that makes it easier to create web sites using Python.

Django takes care of the difficult stuff so that you can concentrate on building your web applications.

Django emphasizes reusability of components, also referred to as **DRY (Don't Repeat Yourself)**, and comes with ready-to-use features like login system, database connection and CRUD operations (Create Read Update Delete).

Django is especially helpful for database driven websites. -

A database-driven website is a website that uses a database to generate pages in real time, instead of storing each page as a separate file. This is different from a static website, where each page is created and stored individually.

4.2 Installation in virtual Env.

To install Django, you must have Python installed, and a package manager like PIP.

PIP is included in Python from version 3.4.

Django Requires Python

To check if your system has Python installed, run this command in the command prompt:

```
python --version
```

If Python is installed, you will get a result with the version number, like this

```
Python 3.9.2
```

If you find that you do not have Python installed on your computer, then you can download it for free from the following website: https://www.python.org/

PIP

To install Django, you must use a package manager like PIP, which is included in Python from version 3.4.

To check if your system has PIP installed, run this command in the command prompt:

```
pip --version
```

If PIP is installed, you will get a result with the version number.

For me, on a windows machine, the result looks like this:

```
pip 20.2.3 from c:\python39\lib\site-packages\pip (python 3.9)
```

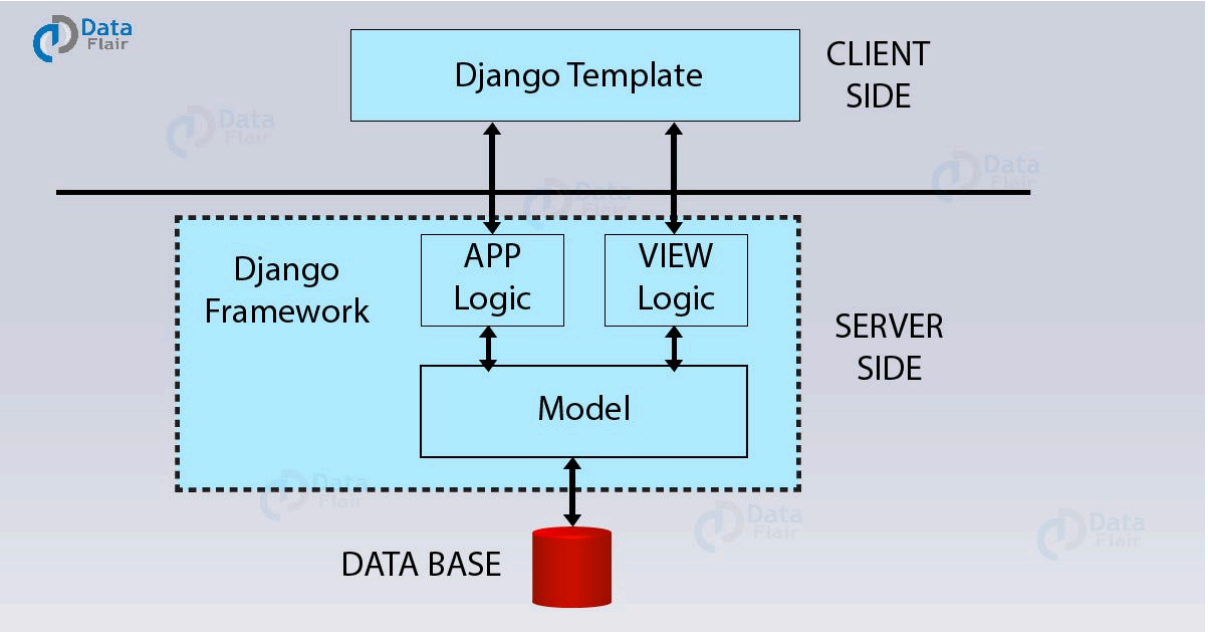
If you do not have PIP installed, you can download and install it from this page: https://pypi.org/project/pip/

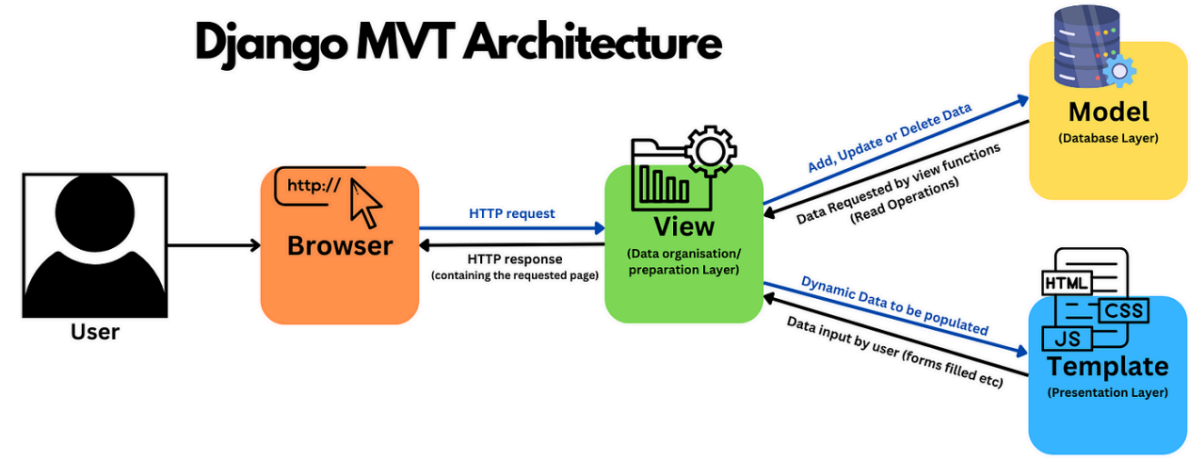
How does Django Work?

Django follows the MVT design pattern (Model View Template).

- Model - The data you want to present, usually data from a database.
- View - A request handler that returns the relevant template and content - based on the request from the user.

- Template - A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.





Model

The model provides data from the database.

In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases.

The most common way to extract data from a database is SQL. One problem with SQL is that you have to have a pretty good understanding of the database structure to be able to work with it.

Django, with ORM, makes it easier to communicate with the database, without having to write complex SQL statements.

The models are usually located in a file called models.py.

View

A view is a function or method that takes http requests as arguments, imports the relevant model(s), and finds out what data to send to the template, and returns the final result.

The views are usually located in a file called views.py.

Template

A template is a file where you describe how the result should be represented.

Templates are often .html files, with HTML code describing the layout of a web page, but it can also be in other file formats to present other results, but we will concentrate on .html files.

Django uses standard HTML to describe the layout, but uses Django tags to add logic:

```
<h1>My Homepage</h1>

<p>My name is {{ firstname }}.</p>
```

The templates of an application is located in a folder named templates.

URLs

Django also provides a way to navigate around the different pages in a website.

When a user requests a URL, Django decides which *view* it will send it to.

This is done in a file called urls.py.

So, What is Going On?

When you have installed Django and created your first Django web application, and the browser requests the URL, this is basically what happens:

1. Django receives the URL, checks the urls.py file, and calls the view that matches the URL.
2. The view, located in views.py, checks for relevant models.
3. The models are imported from the models.py file.
4. The view then sends the data to a specified template in the template folder.
5. The template contains HTML and Django tags, and with the data it returns finished HTML content back to the browser.

4.3 Creating project and apps, Creating superuser and other commands

CREATE PROJECT IN DJANGO

COMMANDS

1. pip freeze - which packages you have installed externally
2. Django-admin startproject myfirstproject

(name depends on your project)

Open cmd with the folder you have made for project

```
C:\django-project\silveroak>python manage.py runserver 4444

C:\django-project\silveroak>python manage.py runserver 8000
```

```
django-admin

Type 'django-admin help <subcommand>' for help on a specific subcommand.

Available subcommands:

[django]
    check
    compilemessages
    createcachetable
    dbshell
    diffsettings
    dumpdata
    flush
    inspectdb
    loaddata
    makemessages
    makemigrations
    migrate
    optimizemigration
    runserver
    sendtestemail
    shell
    showmigrations
    sqlflush
    sqlmigrate
    sqlsequencereset
```

```
squashmigrations
startapp
startproject
test
testserver
```

THIS WILL CREATE PROJECT :

Django-admin : fix command
startproject : fix keyword
Silveroak : name of the project

```
C:\django-project>django-admin startproject silveroak

C:\django-project>django-admin startproject socca
```

STARTING A DEVELOPMENT SERVER :

- **Until and unless you start a development server you won't be able to run the program**

```
C:\django-project\silveroak>python manage.py runserver
CTRL + C will stop the server
```

4.3.1 Creating an app

What is an App?

An app is a web application that has a specific meaning in your project, like a home page, a contact form, or a members database.

In this tutorial we will create an app that allows us to list and register members in a database.

But first, let's just create a simple Django app that displays "Hello World!".

Create App

I will name my app members.

Start by navigating to the selected location where you want to store the app, in my case the my_tennis_club folder, and run the command below.

If the server is still running, and you are not able to write commands, press [CTRL] [BREAK], or [CTRL] [C] to stop the server and you should be back in the virtual environment.

```
py manage.py startapp members
```

Django creates a folder named members in my project, with this content:

```
my_tennis_club
  manage.py
  my_tennis_club/
  members/
    migrations/
      __init__.py
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
```

4.4 Working with APIs and Developer Tools and SQL RESTful architecture, Working with APIs

API : APPLICATION PROGRAMMING INTERFACE

APPLICATION : ANY SOFTWARE THAT HAS SPECIFIC PURPOSE OR FUNCTION

PROGRAMMING :

INTERFACE : REFERS TO THE CONTRACT THAT DICTATES HOW TWO APPLICATIONS TALK TO EACH OTHER USING REQUEST AND RESPONSE

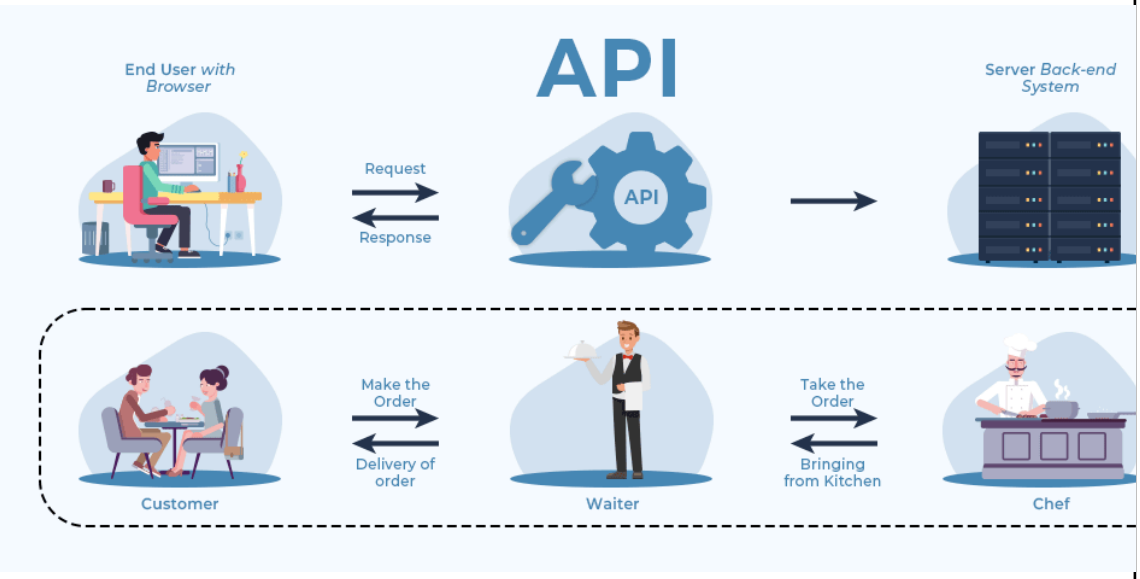
In simple language API means how two applications communicate with each other

Why do we need API ?

Call

Request

Response



REST API architecture refers to the design and structure of a web service that follows the principles of REST (Representational State Transfer). Here's an overview of the components and considerations involved in REST API architecture