**SERVER CLIENT PROGRAM**

**Introduction to django**

Introduction

Django is a web application framework written in Python programming language. It is based on MVT (Model View Template) design pattern. The Django is very demanding due to its rapid development feature. It takes less time to build application after collecting client requirement.

This framework uses a famous tag line: **The web framework for perfectionists with deadlines.**

By using Django, we can build web applications in very less time. Django is designed in such a manner that it handles much of configure things automatically, so we can focus on application development only.

Popularity

Django is widely accepted and used by various well-known sites such as:

* Instagram
* Mozilla
* Disqus
* Pinterest
* Bitbucket
* The Washington Times

Features of Django

* Rapid Development
* Secure
* Scalable
* Fully loaded
* Versatile
* Open Source
* Vast and Supported Community

**Django and python**

* 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.

**MVT: Model View and Template**

# Django MVT

The MVT (Model View Template) is a software design pattern. It is a collection of three important components Model View and Template. The Model helps to handle database. It is a data access layer which handles the data.

The Template is a presentation layer which handles User Interface part completely. The View is used to execute the business logic and interact with a model to carry data and renders a template.

Although Django follows MVC pattern but maintains it?s own conventions. So, control is handled by the framework itself.

There is no separate controller and complete application is based on Model View and Template. That?s why it is called MVT application.

See the following graph that shows the MVT based control flow.



Here, a user **requests** for a resource to the Django, Django works as a controller and check to the available resource in URL.

If URL maps, **a view is called** that interact with model and template, it renders a template.

Django responds back to the user and sends a template as a **response**.

**GETTING STARTED WITH DJANGO**

**Core files**

**Models.py**

In Django, a model is a class which is used to contain essential fields and methods. Each model class maps to a single table in the database.

Django Model is a subclass of **django.db.models.Model** and each field of the model class represents a database field (column).

Django provides us a database-abstraction API which allows us to create, retrieve, update and delete a record from the mapped table.

Model is defined in **Models.py** file. This file can contain multiple models.

# Django URL Mapping

Well, till here, we have learned to create a model, view, and template. Now, we will learn about the routing of application.

Since Django is a web application framework, it gets user requests by URL locater and responds back. To handle URL, **django.urls** module is used by the framework.

Let's open the file **urls.py** of the project and see the what it looks like:

**// urls.py**

**0**HTML Tutorial

1. from django.contrib **import** admin
2. from django.urls **import** path
3. from myapp **import** views
4. urlpatterns = [
5. path('admin/', admin.site.urls),
6. path('index/', views.index),
7. path('hello/',  views.hello),
8. ]

See, Django already has mentioned a URL here for the admin. The path function takes the first argument as a route of string or regex type.

The view argument is a view function which is used to return a response (template) to the user.

The **django.urls** module contains various functions, **path(route,view,kwargs,name)** is one of those which is used to map the URL and call the specified view.

Now, start the server and enter **localhost:8000/hello** to the browser. This URL will be mapped into the list of URLs and then call the corresponding function from the views file.

In this example, hello will be mapped and call hello function from the views file. It is called URL mapping.

**Setting Up Connections**

The **settings.py** file contains all the project settings along with database connection details. By default, Django works with **SQLite,** database and allows configuring for other databases as well.

Database connectivity requires all the connection details such as database name, user credentials, hostname drive name etc.

We need to provide all connection details in the settings file.

**How to fetch the data from database**

The database view is created by following the SQL and it can be injected into a customized data migration with the raw SQL execution command. The next step is to create a Django model which maps to this view so we can **use Django ORM to retrieve the data from the view**.

**Static files**

# Django Static Files Handling

In a web application, apart from business logic and data handling, we also need to handle and manage static resources like CSS, JavaScript, images etc.

It is important to manage these resources so that it does not affect our application performance.

Django deals with it very efficiently and provides a convenient manner to use resources.

The **django.contrib.staticfiles** module helps to manage them.

**Managing users and the django admin tool**

Django-admin-tools is a collection of extensions/tools for the default django administration interface, it includes:

* a full featured and customizable dashboard;
* a customizable menu bar;
* tools to make admin theming easier.

**DJANGO: URL PATTERNS**

# Django Views

A view is a place where we put our business logic of the application. The view is a python function which is used to perform some business logic and return a response to the user. This response can be the HTML contents of a Web page, or a redirect, or a 404 error.

All the view function are created inside the **views.py** file of the Django app.

### Django View Simple Example

**//views.py**

1. **import** datetime
2. # Create your views here.
3. from django.http **import** HttpResponse
4. def index(request):
5. now = datetime.datetime.now()
6. html = "<html><body><h3>Now time is %s.</h3></body></html>" % now
7. **return** HttpResponse(html)    # rendering the template in HttpResponse

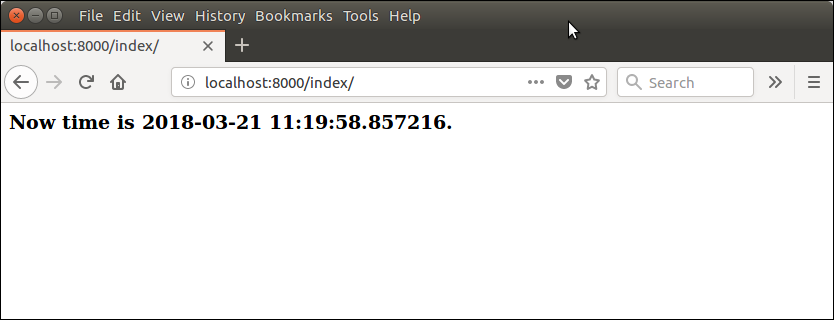
First, we will import DateTime library that provides a method to get current date and time and HttpResponse class.

Next, we define a view function index that takes HTTP request and respond back.

View calls when gets mapped with URL in **urls.py.** For example

1. path('index/', views.index),

**Output:**



**Forms**

# Django Forms

Django provides a Form class which is used to create HTML forms. It describes a form and how it works and appears.

It is similar to the **ModelForm** class that creates a form by using the Model, but it does not require the Model.

Each field of the form class map to the HTML form **<input>** element and each one is a class itself, it manages form data and performs validation while submitting the form.

**Classes**

Class-based views provide an alternative way to implement views as Python objects instead of functions. They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

* Organization of code related to specific HTTP methods (**GET**, **POST**, etc.) can be addressed by separate methods instead of conditional branching.
* Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.

**Validation**

Django provides built-in methods to validate form data automatically. Django forms submit only if it contains CSRF tokens. It uses uses a clean and easy approach to validate data.

The **is\_valid()** method is used to perform validation for each field of the form, it is defined in Django Form class. It returns True if data is valid and place all data into a cleaned\_data attribute.

**Authentication**

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions. This section of the documentation explains how the default implementation works out of the box, as well as how to [extend and customize](https://docs.djangoproject.com/en/4.0/topics/auth/customizing/) it to suit your project’s needs.

**UNIT TESTING WITH DJANGO**

**Test data bases**

Running the tests requires a Django settings module that defines the databases to use. To help you get started, Django provides and uses a sample settings module that uses the **SQLite database**.

**Doctests**

There are several common ways to use doctest: **To check that a module's docstrings are up-to-date by verifying that all interactive examples still work as documented**. To perform regression testing by verifying that interactive examples from a test file or a test object work as expected.

**Debugging**

The Django Debug Toolbar is **a configurable set of panels that display various debug information about the current request/response and when clicked, display more details about the panel's content**. In these panels you can check things like logging, requests to the database, values of variables, headers…

**Best Practices**

## Coding style

## Using virtual environment

## Requirements.txt File

## Correct model naming

## Using the correct related-name in model relationships

## Django templates

**DJANGO and REST API**

**Django REST Framework**

Django **representational state transfer** (REST) framework is a powerful and flexible toolkit for building Web application programming interface (APIs).

Some reasons you might want to use REST framework:

* The [Web browsable API](https://restframework.herokuapp.com/) is a huge usability win for your developers.
* [Authentication policies](https://www.django-rest-framework.org/api-guide/authentication/) including packages for [OAuth1a](https://www.django-rest-framework.org/api-guide/authentication/#django-rest-framework-oauth) and [OAuth2](https://www.django-rest-framework.org/api-guide/authentication/#django-oauth-toolkit).
* [Serialization](https://www.django-rest-framework.org/api-guide/serializers/) that supports both [ORM](https://www.django-rest-framework.org/api-guide/serializers#modelserializer) and [non-ORM](https://www.django-rest-framework.org/api-guide/serializers#serializers) data sources.
* Customizable all the way down - just use [regular function-based views](https://www.django-rest-framework.org/api-guide/views#function-based-views) if you don't need the [more](https://www.django-rest-framework.org/api-guide/generic-views/) [powerful](https://www.django-rest-framework.org/api-guide/viewsets/) [features](https://www.django-rest-framework.org/api-guide/routers/).
* Extensive documentation, and [great community support](https://groups.google.com/forum/?fromgroups#!forum/django-rest-framework).
* Used and trusted by internationally recognised companies including [Mozilla](https://www.mozilla.org/en-US/about/), [Red Hat](https://www.redhat.com/)