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.

Django Model

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.

Let's see an example here, we are creating a model **Employee** which has two fields **first\_name** and **last\_name**.

1. from django.db **import** models
3. **class** Employee(models.Model):
4. first\_name = models.CharField(max\_length=30)
5. last\_name = models.CharField(max\_length=30)

The **first\_name** and **last\_name** fields are specified as class attributes and each attribute maps to a database column.

This model will create a table into the database that looks like below.

1. CREATE TABLE appname\_employee (
2. "id" INT NOT NULL PRIMARY KEY,
3. "first\_name" varchar(30) NOT NULL,
4. "last\_name" varchar(30) NOT NULL
5. );

The created table contains an auto-created **id field**. The name of the table is a combination of app name and model name that can be changed further.

Register / Use Model

After creating a model, register model into the **INSTALLED\_APPS** inside **settings.py.**

**For example,**

1. INSTALLED\_APPS = [
2. #...
3. 'appname',
4. #...
5. ]

Django Model Fields

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Class** | **Particular** |
| AutoField | class AutoField(\*\*options) | It An IntegerField that  automatically increments. |
| BigAutoField | class BigAutoField(\*\*options) | It is a 64-bit integer, much  like an AutoField except  that it is guaranteed to  fit numbers from  1 to 9223372036854775807. |
| BigIntegerField | class BigIntegerField(\*\*options) | It is a 64-bit integer, much  like an IntegerField except  that it is guaranteed to fit  numbers from -9223372036854775808  to 9223372036854775807. |
| BinaryField | class BinaryField(\*\*options) | A field to store raw binary data. |
| BooleanField | class BooleanField(\*\*options) | A true/false field. The  default form widget for  this field is a CheckboxInput. |
| CharField | class DateField(auto\_now=False, auto\_now\_add=False, \*\*options) | It is a date, represented in  Python by a datetime.date instance. |
| DateTimeField | class DateTimeField(auto\_now=False, auto\_now\_add=False, \*\*options) | It is a date, represented in  Python by a datetime.date instance. |
| DateTimeField | class DateTimeField(auto\_now=False, auto\_now\_add=False, \*\*options) | It is used for date and time, represented in Python  by a datetime.datetime instance. |
| DecimalField | class DecimalField(max\_digits=None, decimal\_places=None, \*\*options) | It is a fixed-precision  decimal number, represented  in Python by a Decimal instance. |
| DurationField | class DurationField(\*\*options) | A field for storing periods of time. |
| EmailField | class EmailField(max\_length=254, \*\*options) | It is a CharField that checks  that the value is a valid email address. |
| FileField | class FileField(upload\_to=None, max\_length=100, \*\*options) | It is a file-upload field. |
| FloatField | class FloatField(\*\*options) | It is a floating-point number represented in Python by  a float instance. |
| ImageField | class ImageField(upload\_to=None, height\_field=None, width\_field=None, max\_length=100, \*\*options) | It inherits all attributes and  methods from FileField,  but also validates that the  uploaded object is a valid image. |
| IntegerField | class IntegerField(\*\*options) | It is an integer field.  Values from -2147483648  to 2147483647  are safe in all databases  supported by Django. |
| NullBooleanField | class NullBooleanField(\*\*options) | Like a BooleanField,  but allows NULL as one  of the options. |
| PositiveIntegerField | class PositiveIntegerField(\*\*options) | Like an IntegerField, but  must be either positive or zero (0).  Values from 0 to 2147483647  are safe in all databases  supported by Django. |
| SmallIntegerField | class SmallIntegerField(\*\*options) | It is like an IntegerField,  but only allows values under  a certain (database-dependent) point. |
| TextField | class TextField(\*\*options) | A large text field.  The default form widget  for this field is a Textarea. |
| TimeField | class TimeField(auto\_now=False, auto\_now\_add=False, \*\*options) | A time, represented in  Python by a  datetime.time instance. |

Field Options

Each field requires some arguments that are used to set column attributes. For example, CharField requires mac\_length to specify varchar database.

Common arguments available to all field types. All are optional.

|  |  |
| --- | --- |
| **Field Options** | **Particulars** |
| Null | Django will store empty values as NULL in the database. |
| Blank | It is used to allowed field to be blank. |
| Choices | An iterable (e.g., a list or tuple) of 2-tuples to use as choices for this field. |
| Default | The default value for the field. This can be a value or a callable object. |
| help\_text | Extra "help" text to be displayed with the form widget. It's useful for  documentation even if your field isn't used on a form. |
| primary\_key | This field is the primary key for the model. |
| Unique | This field must be unique throughout the table. |

Django Model Example

We created a model Student that contains the following code in **models.py** file.

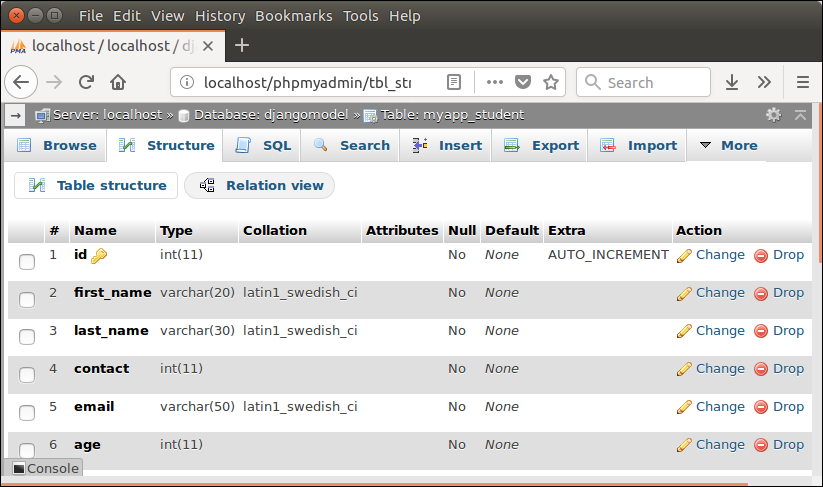
**//models.py**

1. **class** Student(models.Model):
2. first\_name = models.CharField(max\_length=20)
3. last\_name  = models.CharField(max\_length=30)
4. contact    = models.IntegerField()
5. email      = models.EmailField(max\_length=50)
6. age        = models.IntegerField()

After that apply migration by using the following command.

1. python manage.py makemigrations

It will create a table **myapp\_student**. The table structure looks like the below.



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.

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),

Django View HTTP Decorators

HTTP Decorators are used to restrict access to view based on the request method.

These decorators are listed in django.views.decorators.http and return a django.http.HttpResponseNotAllowed if the conditions are not met.

**Syntax**

require\_http\_methods(request\_method\_list)

Django Http Decorator Example

**//views.py**

1. from django.shortcuts **import** render
2. # Create your views here.
3. from django.http **import** HttpResponse, HttpResponseNotFound
4. from django.views.decorators.http **import** require\_http\_methods
5. @require\_http\_methods(["GET"])
6. def show(request):
7. **return** HttpResponse('<h1>This is Http GET request.</h1>')

This method will execute only if the request is an HTTP GET request.

Django Templates

Django provides a convenient way to generate dynamic HTML pages by using its template system.

A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Why Django Template?

In HTML file, we can't write python code because the code is only interpreted by python interpreter not the browser. We know that HTML is a static markup language, while Python is a dynamic programming language.

Django template engine is used to separate the design from the python code and allows us to build dynamic web pages.

Django Template Configuration

To configure the template system, we have to provide some entries in **settings.py** file.

1. TEMPLATES = [
2. {
3. 'BACKEND': 'django.template.backends.django.DjangoTemplates',
4. 'DIRS': [os.path.join(BASE\_DIR,'templates')],
5. 'APP\_DIRS': True,
6. 'OPTIONS': {
7. 'context\_processors': [
8. 'django.template.context\_processors.debug',
9. 'django.template.context\_processors.request',
10. 'django.contrib.auth.context\_processors.auth',
11. 'django.contrib.messages.context\_processors.messages',
12. ],
13. },
14. },
15. ]

Here, we mentioned that our template directory name is **templates**. By default, DjangoTemplates looks for a **templates** subdirectory in each of the INSTALLED\_APPS.

Django Model Form

Django provides a helper class which allows us to create a Form class from a Django model.

# Django Database Connectivity

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.

To connect with MySQL, **django.db.backends.mysql** driver is used to establishing a connection between application and database. Let's see an example.

We need to provide all connection details in the settings file. The settings.py file of our project contains the following code for the database.

1. DATABASES = {
2. 'default': {
3. 'ENGINE': 'django.db.backends.mysql',
4. 'NAME': 'djangoApp',
5. 'USER':'root',
6. 'PASSWORD':'mysql',
7. 'HOST':'localhost',
8. 'PORT':'3306'
9. }
10. }

After providing details, check the connection using the migrate command.

1. python manage.py migrate

This command will create tables for admin, auth, contenttypes, and sessions.