Django by Kamal Sir

Book 2 Index Page:

The book is to be used with the points discussed in the lectures for understanding Django with Kamal Sir.

Topic 9: Django Forms

Django Forms:

- ❑ Django provides Form class which is used to create HTML forms.
- ❑ Each field of the form class maps to the HTML form <input> element.
- ❑ To create Django form we have to create a new file in the application forms.py
- ❑ Syntax is:

```
from django import forms
class FormClassName(forms.Form):
        label_1 = forms.FieldType(args)
        label_2 = forms.FieldType(args)
```

- ❑ To display form to user we create object of Form class in views.py and then pass object to template file to be rendered to the user.

Form Rendering Options:

- ❑ **{{ form }}** will render them all
- ❑ **{{ form.as_table }}** will render as table cells wrapped in <tr> tags.
- ❑ **{{ form.as_p }}** will render them wrapped in <p> tags.
- ❑ **{{ form.as_ul }}** will render them as wrapped in <li> tags.
- ❑ Note: the ouput does not include
    - ❑ <table> and </table> tags
    - ❑ <form> and </form> tags
    - ❑ Or any buttons.

Generated HTML Analysis:

- ❑ The output uses HTML5 syntax.
- ❑ Each FieldType has a default HTML representation.
    - ❑ CharField is represented by <input type="text">
    - ❑ EmailField is represented by <input type="email">
- ❑ HTML text label for each field is generated from field name of FormClassName.
- ❑ Each text label is surrounded by HTML <label> tag.
- ❑ **HTML name** for each tag is taken directly from its **attribute name** from FormClassName.
- ❑ **HTML id** for each tag is prepended by **'id_'** to the field name.
- ❑ The output will be in the same order in which we define the field in form class.

FieldTypes and HTMLInput

FieldTypes	HTML Input
CharField	TextInput
IntegerField	NumberInput
FloatField	NumberInput
BooleanField	CheckBoxInput
EmailField	EmailInput
FileField	FileInput
URLField	URLInput

Arguments:

- **label** – it is used to specify the content of the label.
- **initial** – it is used to declare the initial values of form fields at runtime. It can also be given at field level.
    - fm = StudentForm(initial={'rno':10, 'name':'amit'})
- **required** – it takes value either True(default)/False.
- **help_text** = it will be displayed next to the field when it is rendered.
- **widget** = lets u specify Widget class to use when rendering the field.
- Eg:
- class StudentReg(forms.Form):
  name = forms.CharField(label='enter your name', initial='Amit', required=False,
  help_text='limit 30 chars')

Widget:

- It is Django's representation of HTML input element.
- **name** = forms.CharField(widget=**forms.TextInput**)
    - Will be rendered as <input type="text" ...>
- **Textarea** will be rendered as textarea
- **RadioSelect** will be rendered as
    - <input type="radio" ...>
- **PasswordInput** will be rendered as
    - <input type="password" ...>
- **URLInput** will be rendered as
    - <input type="url" ...>

Topic 10: Django Models

Django Models:

- ❑ Django's Models provide an ORM(Object Relational Mapping) which simplifies database programming by providing mapping between object and underlying database.
- ❑ Django model is the feature of Django to create tables, their fields and various constraints.
- ❑ Each model maps to a single database table.
- ❑ Model class is a class which will represent a table in database.

Model Class:

- ❑ Each model in Python is class that subclasses django.db.models.Model.
- ❑ Each attribute of the model represents a database field.
- ❑ Django also provides automatically generated database-access API to create, retrieve, update and delete objects.

Creating Model Class:

- ❑ To create our own model class we write code in models.py file which is inside the application folder.
- ❑ Syntax is:

```
class ClassName(models.Model):
        FieldName = models.FieldType(option)
```

Working

- ❑ This class will create a table with columns and their data types.
- ❑ Table name will be **ApplicationName_ClassName**.
- ❑ **FieldName** will become tables **ColumnName**.
- ❑ **FieldType** will becomes columns **DataType**.

FieldName:

- ❑ FieldName can be any combination of lowercase letters, uppercase letters, digits and _(underscore).
- ❑ FieldName cannot be Python reserved keyword.
- ❑ FieldName cannot contain more than one underscore and also cannot end with underscore.

FieldType:

- **IntegerField** – an integer with 4 byte range.
- **BigIntegerField** – an integer with 8 byte range.
- **FloatField** – floating point number
- **CharField** – string
- **TextField** – for large amount of text.
- **BooleanField** – true/false field.
- **EmailField** – CharField that checks that the value is a valid email address.
- **URLField** – CharField that checks that the value is a valid URL.

Option:

- **null -** which can be either True(Django will store empty values as NULL) or False(default).
    - null=True/False
- **default** – it defines the default value for the field.
    - default='Mumbai'
- **primary_key** – if True this field is the primary key for the model.
    - primary_key=True
- **unique** – if True this field must be unique else IntegrityError.

How to work with Models:

- Once the model is defined, then we need to run two commands:
    - makemigrations
    - migrate
- **makemigrations:**
- it is used to convert model class into sql statements(which could be in a file located in Applications migrations folder).
- **migrate:**
- it is used to execute sql statements generated by makemigrations.(after this command table will be created).
- Note: any change in model class we need to run the above two to get the changes in the application.

Database Insights:

- Each model in Python is class that subclasses django.db.models.Model.
- Each attribute of the model represents a database field.
- Django also provides automatically generated database-access API to create, retrieve, update and delete objects.

Topic 11: Django Admin

Django – Admin Interface:

- ❑ Django provides a ready to user interface for administrative activities.
- ❑ This interface lets us to administrate Users, perform CRUD operations on your models, etc
- ❑ This admin app is enabled by default and already added into the INSTALLED_APPS list present in the settings.py file.
- ❑ To access admin, on browser we need to write ‘localhost:8000/admin’.

---

superuser:

Create superuser:

- ❑ To access the admin we need to create superuser which can be created by using the following command:
  - ❑ **python manage.py createsuperuser**
- ❑ It will ask for username, email, password and repeat password.

Change superuser password:

- ❑ To change superuser password we can use following:
  - ❑ **python manage.py changepassword <username>**

Find superuser and change password:

- ❑ To get all superuser and change password we use following:

```
python manage.py shell
from django.contrib.auth.models import User
usrs = User.objects.filter(is_superuser=True)
#identify the user

your_user = usrs.filter(username="yourusername")[0]
#youruser = usrs.get(username="yourusername")
#then set the password
```

Topic 12: ModelForm

ModelForm:

- ❑ If we are building a database drive app, then we have will forms which maps closely to Django models.
- ❑ So instead of creating a redundant code to first create a form and then map it to the model , we can directly use ModelForm.
- ❑ In simple words, ModelForm is useful when we want to create forms directly from Database Fields.
- ❑ We can include and exclude fields from the Model as per our choice.
- ❑ With ModelForm it becomes easy to save data to the database.

How to create ModelForm:

- ❑ To create ModelForm, we create subclass forms.ModelForm
- ❑ In the above class we add an inner class Meta, which would contain:
  - ❑ model = ModelName
  - ❑ fields = '__all__' or ['fn1', 'fn2']

FieldTypes:

- ❑ The generated Form class will have a form field for every model field specified.

Model Field	Form Field
CharField	CharField
IntegerField	IntegerField
BooleanField	BooleanField
DateField	DateField
EmailField	EmailField

is_valid():

- ❑ is_valid() method is used to check for validation and returns a boolen value whether data was valid or invalid.

cleaned_data:

- ❑ Once the form is validated we can access clean data (i.e normalizing it to a consistent format) via cleaned_data attribute.
- ❑ cleaned_data contains a key for the field defined in the form.

save():

- ❑ save method on the instance of ModelForm is used to save the data to the database (SQLite3).
- ❑ Calling save() runs a validation check and if the form doesn't validate then we get ValueError.

Topic 13: Django CRUD

CRUD :

- ❑ CRUD stands for Create, Read, Update and Delete.
- ❑ These are the four basic operations which are executed on Database Models.
    - ❑ **Create:**
      it is the ability of the app to store data in the database.
    - ❑ **Read:**
      it is the ability of the app to read data from the database.
    - ❑ **Update:**
      it is the ability of the app to edit stored values in the database.
    - ❑ **Delete:**
      it is the ability of the app to delete the value in the database.

Create:

- ❑ To create an object instantiate it using keyword arguments to the model class and then call save() to save it to the database.
- ❑ **save()** performs an INSERT SQL Statement behind the scenes.
    - ❑ b = Blog(name='amit', ….)
    - ❑ b.save()

Update:

- ❑ To update we need to save changes to an object that's already in the databse using save().
- ❑ In this case **save()** will perform UPDATE SQL Statement behind the scenes.
    - ❑ b = Blog.objects.get(pk=5)
    - ❑ b.name = 'amit'
    - ❑ b.save()

Delete:

- ❑ To delete we can use **delete()** method which delete the object and returns the number of objects deleted.
    - ❑ b = Blog.objects.get(pk=1)
    - ❑ b.delete()

Retrieving objects:

- ❑ **all()** method returns a QuerySet for all the objects in the database table.
  - ❑ All_entries = Entry.objects.all()
- ❑ **get()** method returns the object directly.
  - ❑ one_entry = Entry.objects.get(pk1=1)
- ❑ **filter()** method returns QuerySet containing objects that match the given lookup parameters.
  - ❑ Q1 = Entry.objects.filter(name='Amit')
- ❑ Pythons slicing can be used to limit the number of results.
  - ❑ Entry.objects.al()[:5] is Limit 5
  - ❑ Entry.objects.all()[5:10] is offset 5 limit 5
- ❑ **order_by()** it is used for sorting records.
  - ❑ For ascending order: Entry.objects.order_by('name')
  - ❑ For descending ordere: Entry.objects.order_by('-name')

for Tag:

- ❑ for tag can be used to loop over each item in the given list.
- ❑ Anything enclosed between for tag would be repeated the number of times the loop is run.
- ❑ Syntax is: **{% for i in list %}**
  {{ i }}
  {% endfor %}

HTML Tables:

- ❑ HTML tables allow web developers to arrange data into rows and columns.
- ❑ The <table> tag defines an HTML table.
- ❑ Each table row is defined with a <tr> tag.
- ❑ Each table header is defined with a <th> tag.
- ❑ Each table data/cell is defined with a <td> tag.
- ❑ By default, the text in <th> elements are bold and centered.
- ❑ By default, the text in <td> elements are regular and left-aligned.

Table Customization:

border:

- To add a border to a table, use the CSS border property

border-collapse:

- Borders are collapsed into a single border when possible

padding:

- Cell padding specifies the space between the cell content and its borders.
- By default, table headings are bold and centered.

text-align:

- To left-align the table headings, use the CSS text-align property

border-spacing:

- Border spacing specifies the space between the cells.
- To set the border spacing for a table, use the CSS border-spacing property

caption:

- To add a caption to a table, use the <caption> tag

Topic 14: Template Inheritance

Template:

- ❑ Templates in Django is written in HTML, CSS and JS in .html file.
- ❑ Templates help to generate dynamic HTML pages which are visible to the user.
- ❑ Template directory can be created for entire project and also for each app we can create a different template directory.
- ❑ Project level is used for same layout for each of webpage and app level is for different layout for the webpage.

Template inheritance:

- ❑ Template inheritance allows us to build a base skeleton or template that contains all the common elements of out site and defines blocks that child templates can override.
- ❑ extends tag is used for inheritance of templates in django.
- ❑ **Syntax:**

  {% extends 'template_name.html' %}
- ❑ extends tag tells the Django Template Engine, that this template extends another template.
- ❑ So DTE will evaluate the parent template and replace block tags with the contents of the child template.

block tag:

- ❑ {% block %} – the block tag is used for overriding specific parts of a template.
- ❑ **Syntax :**

  {% block blockname %} … {% endblock %}
- ❑ **Example:**

  {% block title %} … {% endblock %}
  {% block content %} … {% endblock %}

Rules for using template inheritance:

- ❑ If we use {% extends %} in a template, it must be the first template tag in that template else template inheritance wont work.
- ❑ Child templates don't have to define all parent blocks.
- ❑ Multiple block tags cannot have same name in the same template.

Topic 15: Authentication Management

User objects:

- ❑ User objects are the core of the authentication system.
- ❑ They represent the people interacting with your site and enable associating content with creators.
- ❑ Primary attributes of user are:
  - ❑ username, password, email, firstname and lastname..

Creating users:

- ❑ **create_user()** is the function which helps to create users.

How to log in a user:

- ❑ **login()** is used to log a user in.
- ❑ login() saves the user's ID in the session, using Django's session framework.
- ❑ It takes an HttpRequest object and User object.

How to log out a user:

- ❑ **logout()** is used to log out a user.
- ❑ It takes an HttpRequest object and has no return value.
- ❑ Note logout() doesn't throw any errors if the user wasn't logged in.

Authenticating users:

- ❑ **authenticate()** is the function to verifiy a set of credentials.
- ❑ It takes as input keyword arguments username and password and checks in the backend.
- ❑ If credentials are valid it returns User object else returns None.

Checking if user is authenticated:

- ❑ Every request has an attribute request.user which represents the current user.
- ❑ If the user has logged in then it will return an instance of User.
- ❑ To check if user is authenticated we use the following:
  - ❑ **request.user.is_authenticated**

Limiting access to logged in users:

- ❑ To limit access to pages to logged in users we check **request.user.is_authenticated** and either:
  - ❑ Redirect to login page
  - ❑ Or
  - ❑ Display an error message.

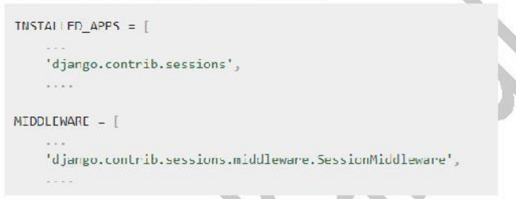
Topic 16 : Session Management

Session:

- ❑ All communication between web browsers and servers is via HTTP which is stateless.
- ❑ Stateless means client and server are completely independent of each other.
- ❑ Sessions are the mechanism used by Django for keeping track of the state between the site and a particular browser.
- ❑ Django uses a cookie containing a special session id to identify each browser and its associated session with the site. Note the session data is stored in the site database.

Enabling Session:

- ❑ Sessions are automatically enabled.
- ❑ It can be checked by inspecting the settings.py



```
INSTALLED_APPS = [
    ...
    'django.contrib.sessions',
    ....

MIDDLEWARE = [
    ...
    'django.contrib.sessions.middleware.SessionMiddleware',
    ....
```

Using Session:

- ❑ To use session we can access the **<u>session</u>** attribute in the view from the **<u>request</u>**. This attribute represents the specific connection to the browser.
- ❑ Set item:
  - ❑ **request.session['key'] = 'value'**
- ❑ Get item:
  - ❑ **returned_value = request.session['key']**
  - ❑ **returned_value = request.session.get('key', default=None)**
- ❑ Delete item:
  - ❑ **del request.session['key']**
- ❑ Contains:
  - ❑ **'key' in request.session**

Ending Session:

- ❑ **flush():**
- ❑ Deletes the current session data from the session and deletes the session cookie. This is used to ensure that the previous session data cant be accessed again from the users browser.
- ❑ **clear_expired:**
  - ❑ Removes expired sessions from the session store.
- ❑ **SESSION_COOKIE_AGE:**
  - ❑ The age of session cookies in seconds. Default age is 2 weeks ie 1209600.

Topic 17 Django Deployment

pythonanywhere:

- ❑ Pythonanywhere is a UK based web hosting service.
- ❑ It provides in-browser access to server based Python and Bash Command line interfaces.
- ❑ It was founded by Giles Thomas and Robert Smithson in 2012.

Steps to deploy:

1. Upload code to PythonAnywhere.
2. Setup virtualenv and install Django and any other requirement.
3. Setup the web app using manual config option.
4. Add any other setup for static files and databases.

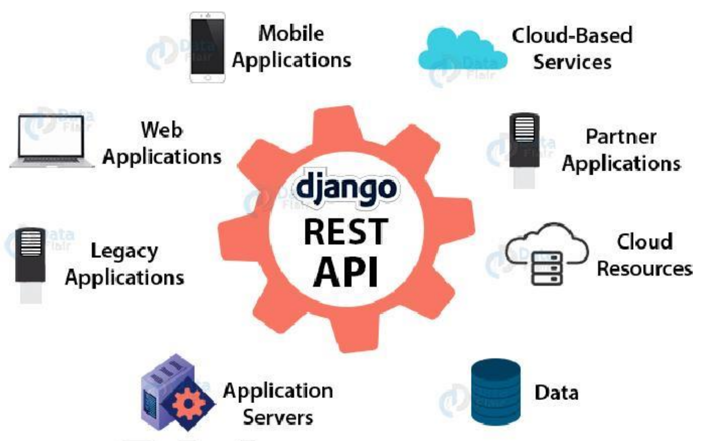
Topic 18 Django RESTful Framework

What is API:

- ❑ An **Application Programming Interface (API)** is used by two applications trying to communicate with each other over a network.

RESTful API:

- ❑ REST stands for Representational State Transfer.
- ❑ REST API is a web service API which uses URL's and HTTP protocol and JSON for data format.



RESTful Structure:

- ❑ In a RESTful API, endpoints (URLs) define the structure of the API and how end users access data from our application using HTTP methods: GET, POST, PUT and DELETE.
- ❑ End points should be logically organized around:
  - ❑ Collections (many elements) eg: /posts/
  - ❑ And
  - ❑ Elements (single element) et: /posts/<id>

Django REST Framework:

- ❑ Django REST Framework (DRF) is used to develop REST APIs for Django.
- ❑ DRF is not a separate Framework, it is build upon Django Framework.

ModelSerializer:

- ❑ ModelSerializer is used to transform our model instances into JSON (Java Script Object Notation).
- ❑ JSON is similar to Python Dictionary which is sent and received by the API to various applications.

request and Response:

- ❑ request.data is similar to request.POST but more useful for working with Web APIs.
- ❑ Response is used to return content to the client.

@api_view:

- ❑ @api_view is decorator for working with function based views.

status:

- ❑ REST Framework provides explicit identifiers for each status code, such as HTTP_400_BAD_REQUEST in the status module.

Authentication:

- ❑ Authentication is the mechanism of associating an incoming request with a set of identifying credentials, such as
  - ❑ the user the request came from,
  - ❑ or
  - ❑ the token that it was signed with.
  - ❑ Unauthenticated responses that are denied permission will result in an HTTP 401 Unauthorized.

Types of Authentication:

- ❑ **BasicAuthentication:**
  - ❑ This authentication scheme uses HTTP Basic Authentication, signed against a user's username and password.
  - ❑ Unauthenicated response would be:
    - ❑ <Response [401]>
    - ❑ {'detail': 'Invalid username/password.'}
- ❑ **TokenAuthentication:**
  - ❑ This authentication scheme uses a simple token-based HTTP Authentication scheme.
  - ❑ Unauthenticated response would be:
    - ❑ <Response [401]>
    - ❑ {'detail': 'Invalid token.'}