**Django Forms**

Django forms are an advanced set of HTML forms that can be created using python and support all features of HTML forms in a pythonic way. This post revolves around how to create a basic form using various Form Fields and attributes. Creating a form in Django is completely similar to creating a model, one needs to specify what fields would exist in the form and of what type. For example, to input, a registration form one might need First Name (CharField), Roll Number (IntegerField), and so on.

**Types of Forms**

* Raw HTML Form
* Django Form class
* Django ModelForm

**1.Raw HTML Form**

In Django, you can create HTML forms in a variety of ways, including using raw HTML in your templates or by using Django's form handling and rendering capabilities. To create a form using raw HTML in a Django template, follow these steps:

**Step 1: Create a Django App**

If you haven't already, create a Django app using the command line:

**python manage.py startapp myapp**

**Step 2: Register the application**

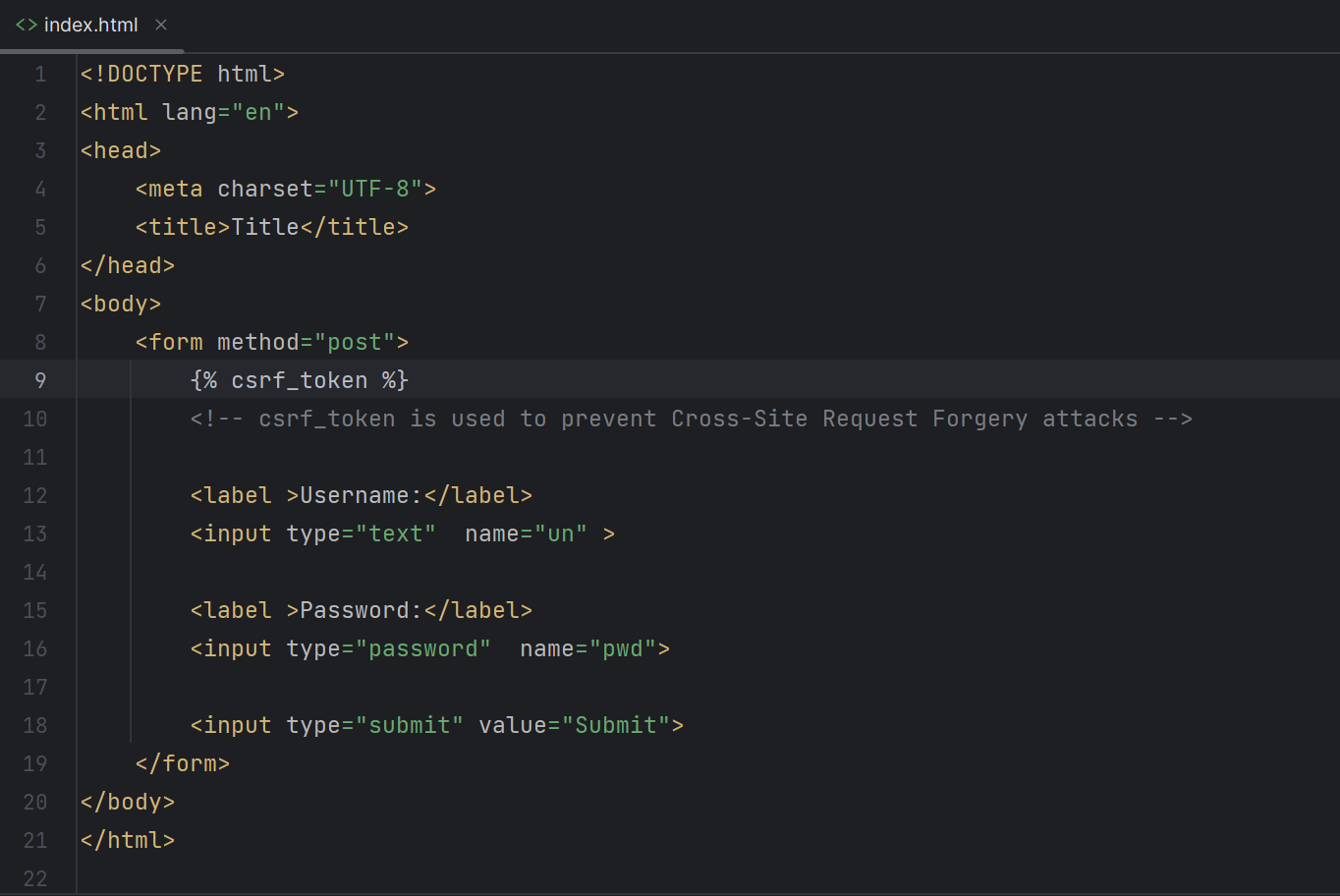
Register app into the INSTALLED\_APPS list inside settings.py file.

A screenshot of a computer

Description automatically generated

**Step 3:** **Create a Django Template:**

Create or open an existing Django template directory (.html file) where you want to include the raw HTML form.



**action:** This attribute specifies where the form data should be sent when submitted. Replace "/submit-form" with the URL or path where your form data should be processed.

**method:** This attribute specifies the HTTP method to be used when submitting the form. POST is commonly used to send data securely.

Inside the <form> tags, you can add various input fields using the <input> element:

Explanation:

* <label>: Provides a label for an input field. The for attribute links the label to its corresponding input field.
* <input>: Creates different types of input fields (text, email, textarea, submit, etc.). Each has different attributes like type, id, and name.
* <input type="submit">: Creates a submit button to send the form data.
* Closing the Form, Always ensure to properly close the form tag:</form>

Once the form is created, you can test it by opening the HTML file in a web browser. Fill in the form fields and submit it. The form data will be sent to the specified action URL using the specified method.

**Note:**

* Use appropriate id and name attributes for each input field.
* Always ensure proper security measures when handling form submissions to prevent exploits like SQL injection or cross-site scripting (XSS) attacks.
* Remember to include {% csrf\_token %} inside the <form> tag when using the POST method to prevent CSRF (Cross-Site Request Forgery) attacks.
* In a Django view, you can access this data using request.GET for GET requests or request.POST for POST requests to retrieve the values by their respective name attributes specified in the form.

**Methods:**

**GET Method:**

* Sends data in the URL's query string.
* Parameters are visible in the URL.
* Limited data can be sent (URL has a length limit).
* Can be bookmarked and cached.
* Generally used for retrieving data from the server.
* Not secure for sensitive information as data is visible in the URL.

Example:

<form method="GET">

<input type="text" name="username">

<input type="submit" value="Submit">

</form>

If submitted, the URL might look like: process.php?username=enteredName.

**POST Method:**

* Sends data in the body of the HTTP request.
* Parameters are not visible in the URL.
* Can send larger amounts of data compared to GET.
* Not bookmarked or cached (by default).
* More secure for sensitive information as data is not visible in the URL.

Example:

<form method="POST">

<input type="text" name="username">

<input type="submit" value="Submit">

</form>

**Step 4: Create a View**

In your app's views.py, create a view function to render the form:

A screen shot of a computer program

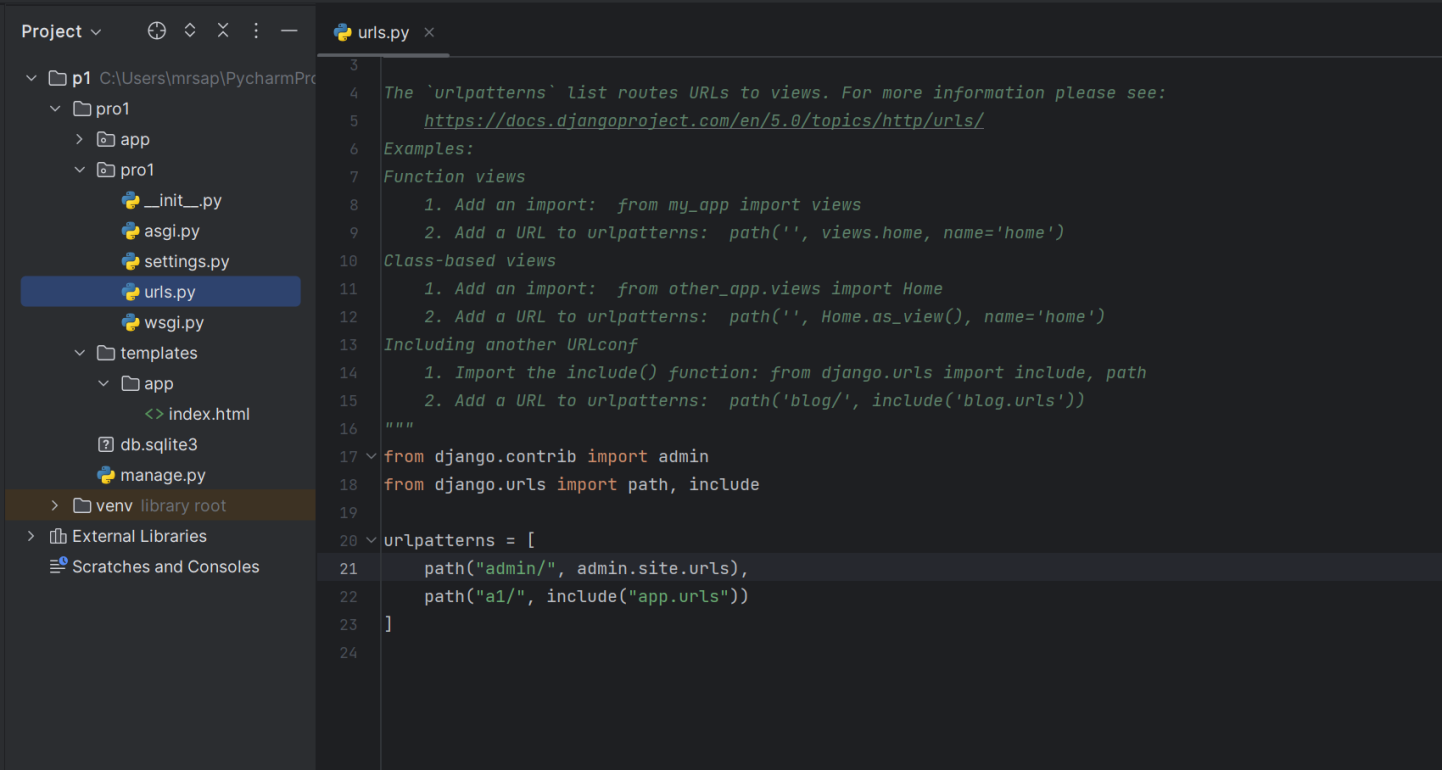
Description automatically generated

**Step 5: Configure URLs**

In your project's urls.py, map the URL to the view:

A screen shot of a computer

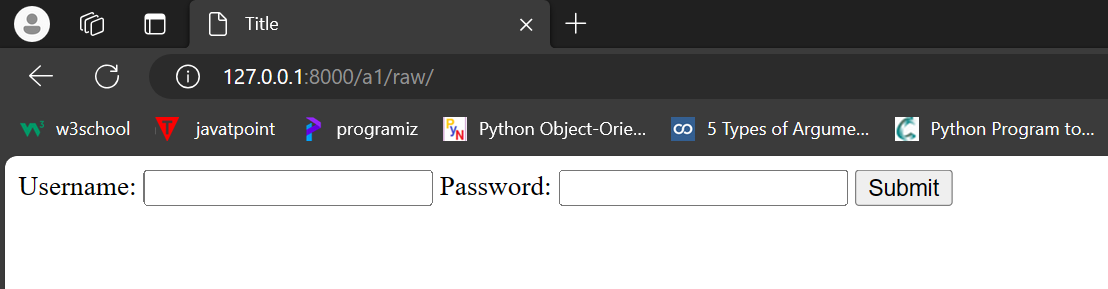
Description automatically generated



**Step 6: Run your Django development server:**

**python manage.py runserver**

**Visit http://127.0.0.1:8000/myform/ in your browser to access the form and test its functionality.**

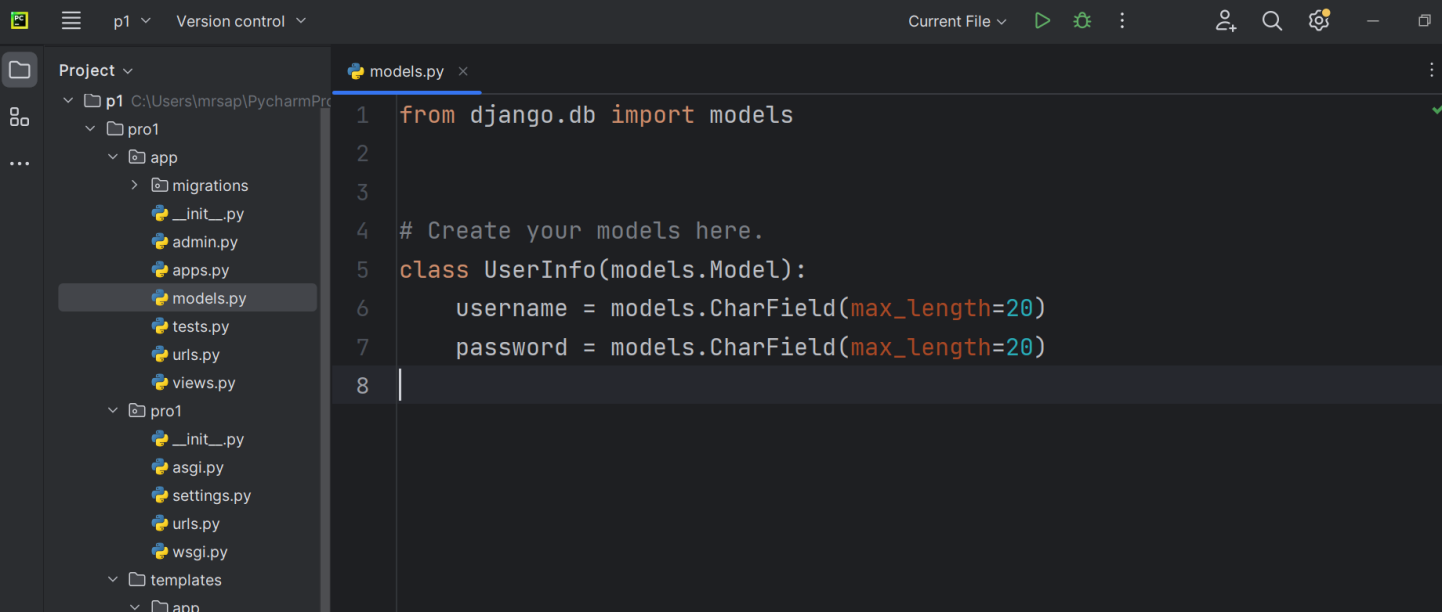


A screenshot of a computer

Description automatically generated

**Step 7: Data Store into DataBase**

* For that we should have model class defined in models.py Which has same fields as we are taking from user.



**Command:**

**Py manage.py makemigrations**

**Py manage.py migrate**

**Step 8: Insert ORM query in views.py file**

A screen shot of a computer

Description automatically generated

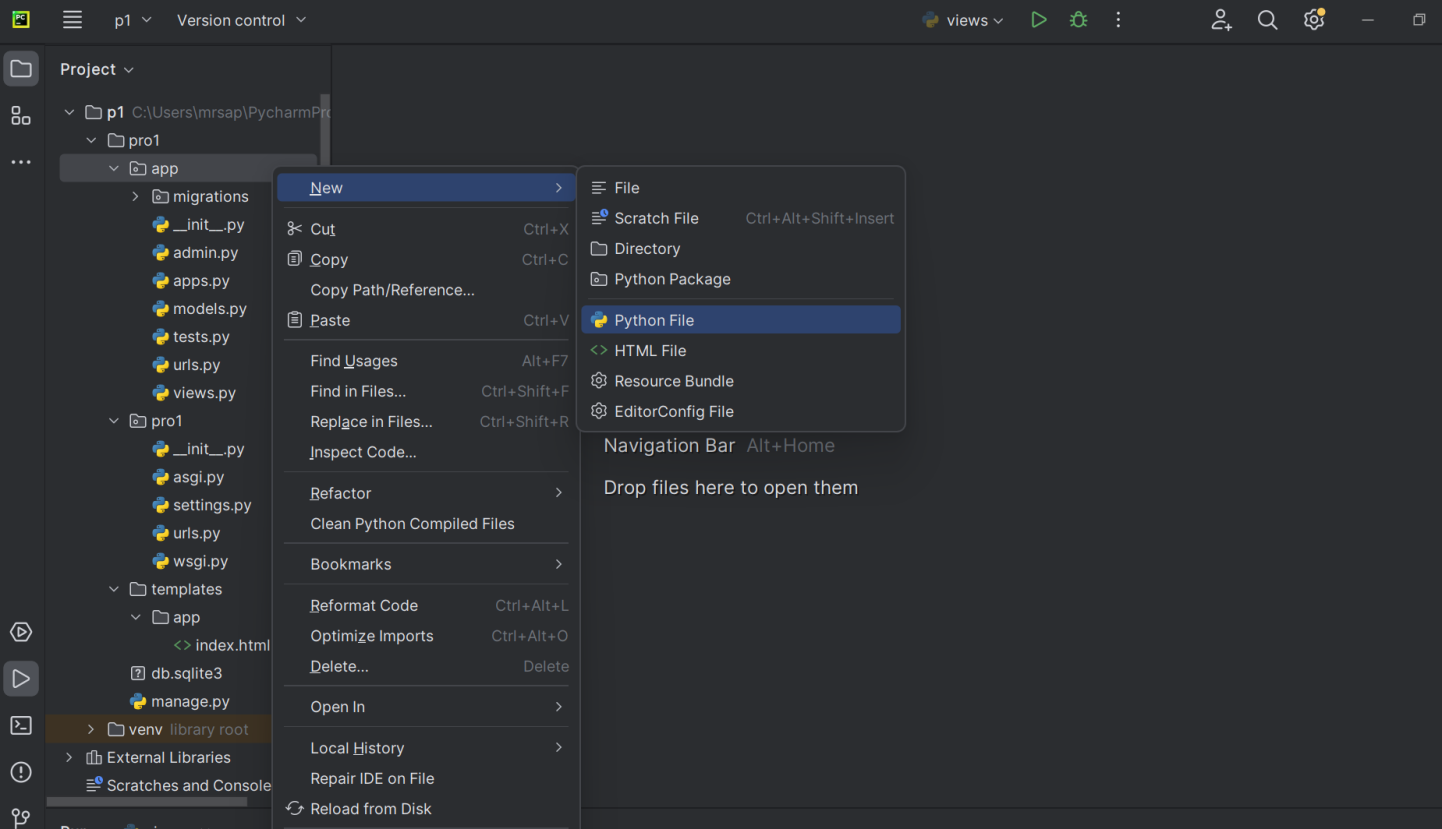
**Django Form Class:**

When one creates a Form class, the most important part is defining the fields of the form.

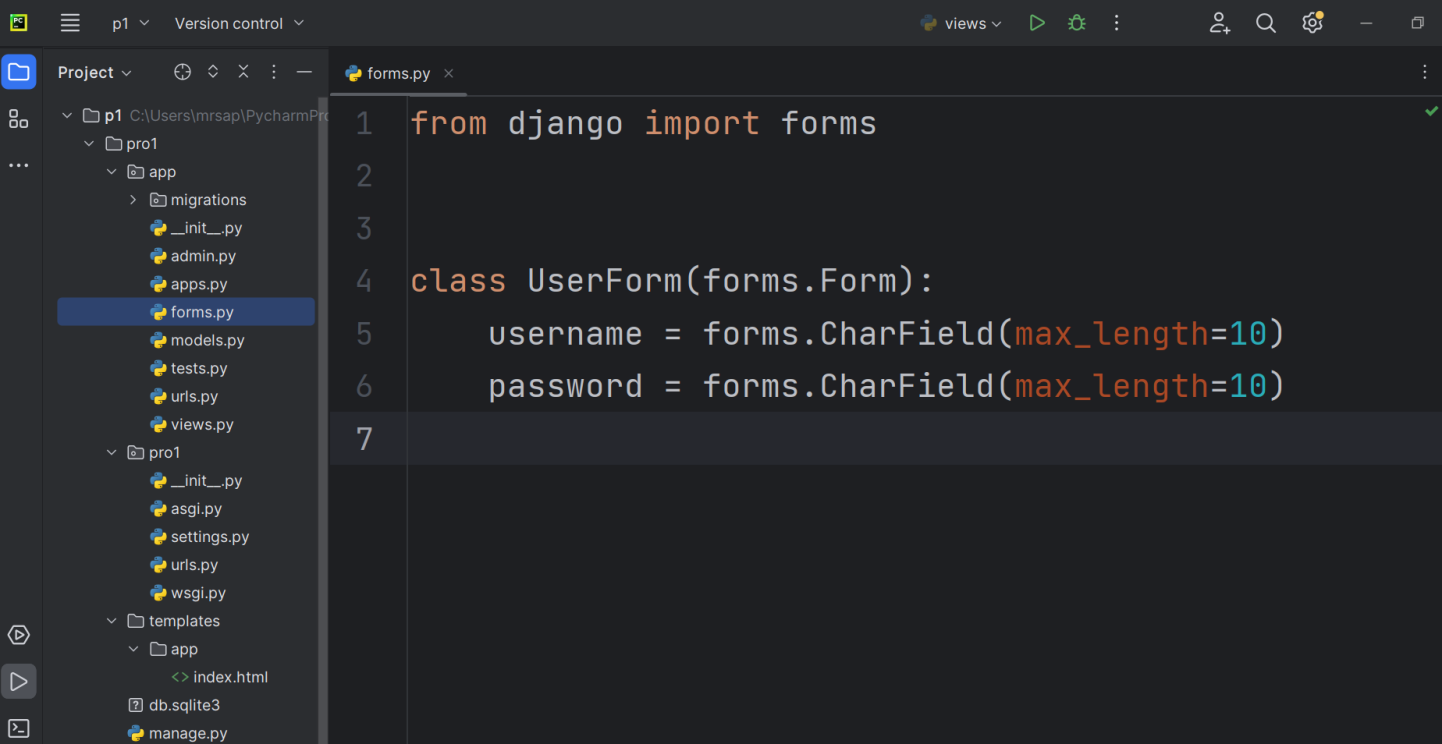
Forms are basically used for taking input from the user in some manner and using that information for logical operations on databases. For example, Registering a user by taking input as his name, email, password, etc.

Creating a form in Django is completely similar to creating a model, one needs to specify what fields would exist in the form and of what type. For example, to input, a registration form one might need First Name (CharField), Roll Number (IntegerField), and so on.

**Step 1:** **Create forms.py file in app directory.**



**Step 2: After creating forms.py file in our application folder with our required fields.**



* UserForm is a form class that inherits from Form class and imported from forms.py.
* Username, password are form fields defined as class attributes.( each field would be mapped as an input field in HTML)
* forms.CharField, forms.EmailField, and forms.Textarea are examples of field types provided by Django. They allow you to specify the type of data expected and provide validation rules.
* The arguments that are common to most fields are listed below (these have sensible default values):
  + **required**: If True, the field may not be left blank or given a None value. Fields are required by default, so you would set required=False to allow blank values in the form.
  + **label**: The label to use when rendering the field in HTML. If a label is not specified, Django will create one from the field name by capitalizing the first letter and replacing underscores with spaces (e.g. Renewal date).
  + **label\_suffix**: By default, a colon is displayed after the label (e.g. Renewal date:). This argument allows you to specify a different suffix containing other character(s).
  + **initial**: The initial value for the field when the form is displayed.
  + **widget**: The display widget to use.
  + **help\_text** (as seen in the example above): Additional text that can be displayed in forms to explain how to use the field.
  + **error\_messages:** A list of error messages for the field. You can override these with your own messages if needed.
  + **validators**: A list of functions that will be called on the field when it is validated.
  + **localize**: Enables the localization of form data input (see link for more information).
  + **disabled**: The field is displayed but its value cannot be edited if this is True. The default is False.

Once you've defined your form class, you can use it in your views to render the form in a template, process submitted data, validate it, and perform any necessary actions.

**Step 3: Define View**

A screen shot of a computer

Description automatically generated

* On a GET request, an instance of **MyForm** is created and passed to the template for rendering.
* On a POST request, the form is instantiated with the data from the request.
* 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 form.cleaned\_data attribute and perform further actions, such as saving to a database.
* Pass formclass object into the context.

{“form”: formclass\_nm}

**Step 4:** **Define App\_url**

**PROJECT\_NM🡪APP\_NM🡪urls.py**

A screenshot of a computer program

Description automatically generated

**Step 5: Render Django Forms using Template**

Django form fields have several built-in methods to ease the work of the developer but sometimes one needs to implement things manually for customizing User Interface(UI). A form comes with 3 in-built methods that can be used to render Django form fields.

* {{ form.as\_table }} will render them as table cells wrapped in <tr> tags
* {{ form.as\_p }} will render them wrapped in <p> tags
* {{ form.as\_ul }} will render them wrapped in <li> tags

A screen shot of a computer program

Description automatically generated

{% csrf\_token %} is used for security reasons to prevent Cross-Site Request Forgery (CSRF) attacks. {{ form.as\_p }} will render the form fields as paragraph elements (<p>).

**Step 6: Run development server**

**Py manage.py runserver**

A screenshot of a computer

Description automatically generated

**Django Model Form:**

It is a class which is used to create an HTML form by using the Model. It is an efficient way to create a form without writing HTML code.

Django automatically does it for us to reduce the application development time.

**Step 1:** **Create a model**

Create Model that contains fields name and other metadata. It can be used to create a table in database and dynamic HTML form.

**// model.py**

A screenshot of a computer program

Description automatically generated

This file contains a class that inherits ModelForm and mention the model name for which HTML form is created.

**Step 2: Migrate**

**Py manage.py makemigrations**

**Py manage.py migrate**

**Step 3: Create Forms.py file**

**// forms.py**

A screenshot of a computer

Description automatically generated

* UserModelForm is a form class is inherited from ModelForm.
* The Meta class is used to change the behavior of the ModelForm. Within it, specify the model your fields come from and the fields you want to use from that model.( to specify Model information and required fields.)
* field declarations if we are performing any custom validations.If we are not defining any custom validations then here we are not required to specify any field.

**Case 1:** All Fields

class Meta:

# we have to specify Model class name and requied fields

model=Student

fields='\_\_all\_\_'

* **Case 2:** Instead of all fields if we want only selected fields, then we have to specify as follows

class Meta:

model=Student

fileds=('field1','field2','field3') #In the form only 3 fields will be considered.If Model class contains huge number of fields and we required to consider very less number of fields in the form then we should use this approach.

* **Case 3:**Instead of all fields if we want to exclude certain fields, then we have to specify as follows:

class Meta:

model=Student

exclude=['field1', 'field2'] #In the form all fields will be considered except field1 and field2.If the Model class contains huge number of fields and if we want to exclude very few fields then we have to use this approach

**Step 4: Define views**

**//views.py**

A screen shot of a computer

Description automatically generated

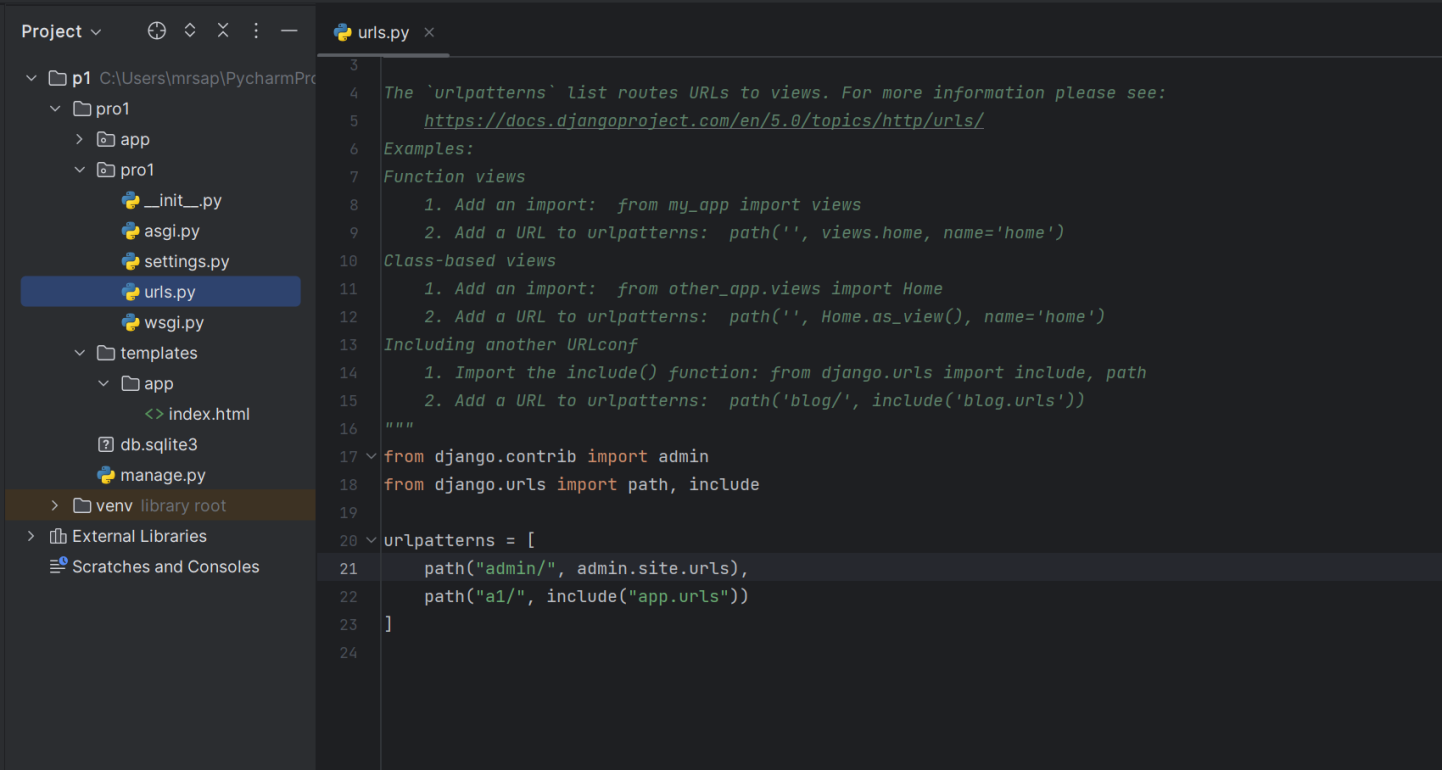
**Step 5: Define urls**

**App\_urls**

A screen shot of a computer

Description automatically generated

**Project\_urls**



**Step 6: Define Template**

And finally, create a **index.html** file that contains the following code.

A screen shot of a computer program

Description automatically generated

**Step 7: Run Server**

**python manage.py runserver**

After that access the template by **localhost:8000/index** URL, and it will display the following output to the browser.

A screenshot of a computer

Description automatically generated