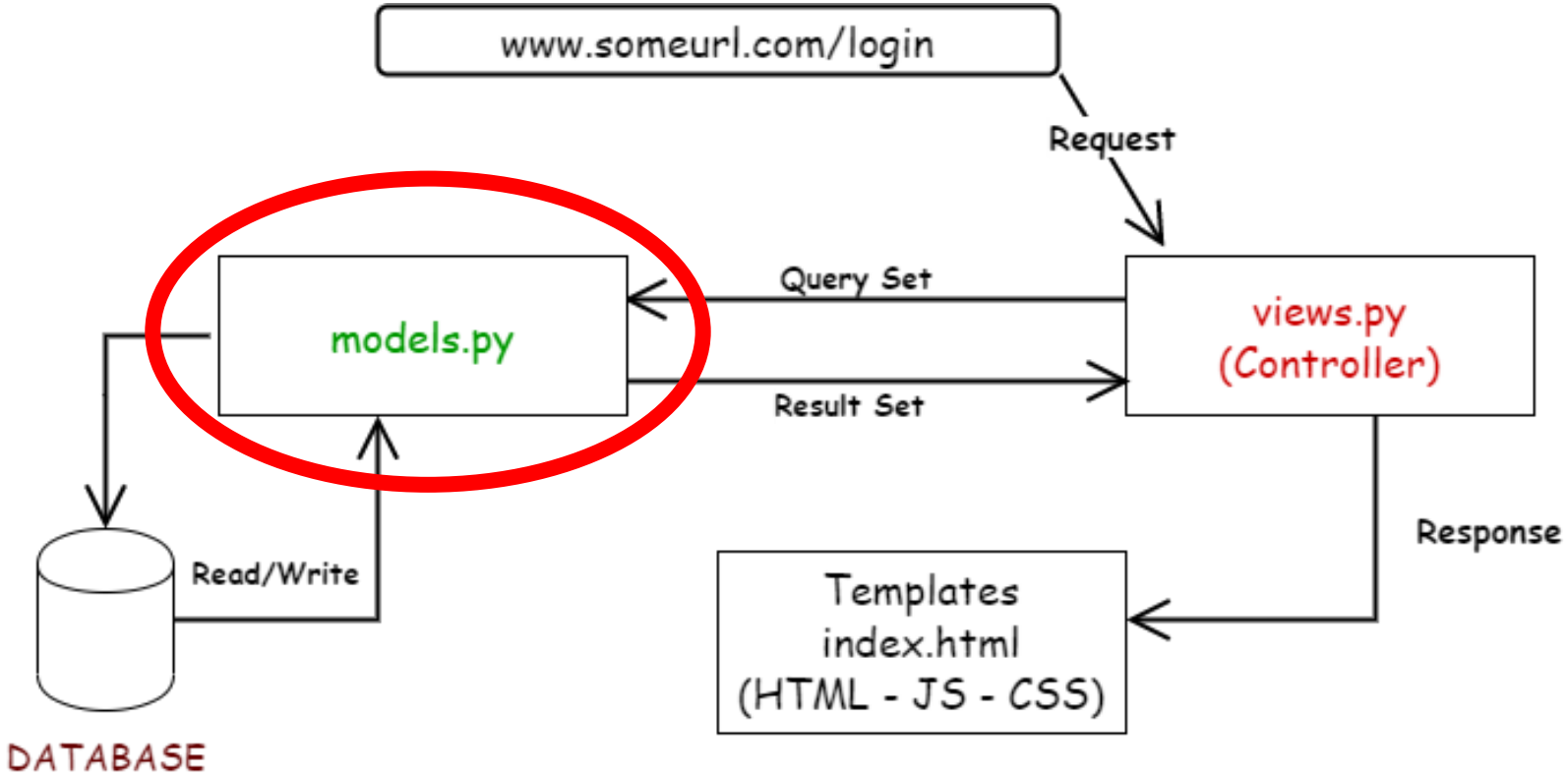
Django Framework

Merit Ehab

Models


www.someurl.com/login
Request
Query Set
models.py
views.py
(Controller)
Result Set
Response
Read/Write
Templates
index.html
(HTML - JS - CSS)
DATABASE

Models

- A model in Django is a special kind of object that is saved in the database.
- We store our models in <app_dir>/models.py => blog/models.py.
- Edit ***blog/models.py*** and write the following code.

Models

blog > models.py > ...

```python
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User



class Post(models.Model):
    author = models.ForeignKey(User, on_delete= models.CASCADE,related_name='blog_posts')
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(default=timezone.now)
    published_date = models.DateTimeField(blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()

    def __str__(self):
        return self.title
```

Models (Field Options)

null	Django will store empty values as **NULL** in the database. Default is **False**.
blank	If **True** the field is allowed to be blank, Default is **False**.
db_column	The name of the database column to use for this field.
default	The default value for the field.
peimary_key	If **True**, this field is the primary key for the model (null=False and unique=True).
unique	If **True**, this field must be unique throughout the table.
verbose_name	A human-readable name for the field. If it isn't given, Django will automatically create it using the field's attribute name, converting underscores to spaces

```
title = models.CharField(null=False,blank=False,default="Untitled",unique=True)
```

```
os = models.CharField(......,verbose_name=_('Operating System'))
```

Models (Field Options)

choices	A Tuple of choices that Field value can be (The first element in each tuple is the actual value to be set on the model, and the second element is the human-readable name)

```python
from django.db import models

class Student(models.Model):
    FRESHMAN = 'FR'
    SOPHOMORE = 'SO'
    JUNIOR = 'JR'
    SENIOR = 'SR'
    YEAR_IN_SCHOOL_CHOICES = [
        (FRESHMAN, 'Freshman'),
        (SOPHOMORE, 'Sophomore'),
        (JUNIOR, 'Junior'),
        (SENIOR, 'Senior'),
    ]
    year_in_school = models.CharField(
        max_length=2,
        choices=YEAR_IN_SCHOOL_CHOICES,
        default=FRESHMAN,
    )

    def is_upperclass(self):
        return self.year_in_school in (self.JUNIOR, self.SENIOR)
```

Models (Char & Text)

CharField	A string field, for small- to large-sized strings.
EmailField	A **CharField** that checks that the value is a valid email address.
URLField	A **CharField** accepts valid urls only.
max_length	The maximum length (in characters) of the field.
TextField	A large text field

Models (Numeric & Boolean)

IntegerField	An integer. Values from -2147483648 to 2147483647.
AutoField	An **IntegerField** that automatically increments according to available IDs.
DecimalField	A fixed-precision decimal number.
max_digits	The maximum number of digits allowed in the number.
decimal_places	The number of decimal places to store with the number.
BooleanField	A true/false field.

```
models.DecimalField(..., max_digits=5, decimal_places=2)
```

Models (Date & Time)

DateField	A date, represented in Python by a **datetime.date** instance.
TimeField	A time, represented in Python by a **datetime.time** instance.
DateTimeField	A date and time, represented in Python by a **datetime.datetime** instance.
auto_now	Automatically set the field to now every time the object is saved.
auto_now_add	Automatically set the field to now when the object is first created.

Models (Relationships Field)

ForeignKey	A many-to-one relationship.
ManyToManyField	A many-to-many relationship.
OneToOneField	A one-to-one relationship.
on_delete	
to_field	

Models (Relationships Field)

```
class artist(models.Model):
    name = models.CharField(max_length=200)

class movie(models.Model):
    title = models.CharField(max_length=100)
    artists = models.ManyToManyField(artist, related_name = 'actor', through='roles')

class role(models.Model):
    role_name = models.CharField(max_length=100)
    artist = models.ForeignKey(artist, on_delete=models.CASCADE)
    movie = models.ForeignKey(movie, on_delete=models.CASCADE)
```

Models

- To add the new model to our database, we have to let django know about our change.

  `python manage.py makemigrations blog`

```
$ python manage.py makemigrations blog
Migrations for 'blog':
  blog\migrations\0001_initial.py
    - Create model Post
```

- To apply the migration file you have just made to your database

  `python manage.py migrate blog`

```
$ python manage.py migrate blog
Operations to perform:
  Apply all migrations: blog
Running migrations:
  Applying blog.0001_initial... OK
```

Django ORM & QuerySets

- QuerySets is a list of objects of a given models.
- It allows you to read data from database, filter it and order it.
- Open your console and type

  ```
  python manage.py shell
  ```
- Try the following queries to get to know the model operations

Django ORM & QuerySets (Insert)

```
In [1]: from blog.models import Post

In [2]: from django.contrib.auth.models import User

In [3]: user = User.objects.get(username="admin")

In [4]: Post.objects.create(author=user, title="Demo", text="Some text")
Out[4]: <Post: Demo>
```

OR

```
In [5]: post = Post(author=user, title="Demo2")

In [6]: post.text = "This is demo2 post text"

In [7]: post.save()
```

Django ORM & QuerySets (Select...Where)

```
In [8]: Post.objects.create(author=user, title="Demo", text="Hello World!")
Out[8]: <Post: Demo>

In [9]: Post.objects.all()
Out[9]: <QuerySet [<Post: First post>, <Post: The post>, <Post: Django>, <Post: Random>, <Post: The Web>, <Post: Demo>, <Post: Demo2>, <Post: Demo>]>

In [10]: Post.objects.filter(title="Demo")
Out[10]: <QuerySet [<Post: Demo>, <Post: Demo>]>

In [11]: Post.objects.get(title="Demo2")
Out[11]: <Post: Demo2>
```

Django ORM & QuerySets (Delete)

```
In [13]: Post.objects.filter(title="Demo").delete()
Out[13]: (2, {'blog.Post': 2})

In [14]: Post.objects.all()
Out[14]: <QuerySet [<Post: First post>, <Post: The post>, <Post: Django>, <Post: Random>, <Post: The Web>, <Post: Demo2>]>
```

Django ORM & QuerySets (Update)

The first object with Django Title I added it earlier from the Django admin

```
In [14]: Post.objects.create(author=user, title="Django", text="Hello World!")
Out[14]: <Post: Django>

In [15]: Post.objects.all()
Out[15]: <QuerySet [<Post: First post>, <Post: The post>, <Post: Django>, <Post: Random>, <Post: The Web>, <Post: Demo2>, <Post: Django>]>

In [16]: Post.objects.filter(title="Django").update(text="My First Django Project")
Out[16]: 2

In [17]: Post.objects.filter(title="Django")[0].text
Out[17]: 'My First Django Project'

In [18]: Post.objects.filter(title="Django")[1].text
Out[18]: 'My First Django Project'
```

Django ORM & QuerySets

```
In [19]: Post.objects.filter(title__in=["Django","Demo2"])
Out[19]: <QuerySet [<Post: Django>, <Post: Demo2>, <Post: Django>]>

In [20]: Post.objects.create(author=user, title="demo2")
Out[20]: <Post: demo2>

In [21]: Post.objects.all()
Out[21]: <QuerySet [<Post: First post>, <Post: The post>, <Post: Django>, <Post: Random>, <Post: The Web>, <Post: Demo2>, <Post: Django>, <Post: demo2>]>

In [22]: Post.objects.filter(title__exact="demo2")
Out[22]: <QuerySet [<Post: demo2>]>

In [23]: Post.objects.filter(title__iexact="demo2")
Out[23]: <QuerySet [<Post: Demo2>, <Post: demo2>]>

In [24]: from datetime import datetime

In [25]: today = datetime.today()

In [26]: Post.objects.filter(created_date__gt=today)
C:\Users\merit.ehab.FLAIRSTECH\AppData\Local\Programs\Python\Python37\lib\site-packages\django\db\models\fields\__init__.py:1368: RuntimeWarning: DateTimeField Post.created_date received a naive datetime (2020-05-
12 12:02:29.119988) while time zone support is active.
  RuntimeWarning)
Out[26]: <QuerySet []>

In [27]: Post.objects.filter(created_date__gt=today.date())
C:\Users\merit.ehab.FLAIRSTECH\AppData\Local\Programs\Python\Python37\lib\site-packages\django\db\models\fields\__init__.py:1310: RuntimeWarning: DateTimeField Post.created_date received a naive datetime (2020-05-
12 00:00:00) while time zone support is active.
  RuntimeWarning)
Out[27]: <QuerySet [<Post: Demo2>, <Post: Django>, <Post: demo2>]>
```

Django ORM & QuerySets (F & Q)

```
In [28]: Post.objects.create(author=user, title="admin", text="Admin post")
Out[28]: <Post: admin>

In [29]: from django.db.models import F

In [30]: Post.objects.filter(author__username= F("title"))
Out[30]: <QuerySet [<Post: admin>]>
```

```
In [31]: from django.db.models import Q

In [32]: Post.objects.filter(Q(title__startswith="The") | Q(created_date__gt=today.date()))
C:\Users\merit.ehab.FLAIRSTECH\AppData\Local\Programs\Python\Python37\lib\site-packages\django\db\models\fields\__init__.py:
12 00:00:00) while time zone support is active.
  RuntimeWarning)
Out[32]: <QuerySet [<Post: The post>, <Post: The Web>, <Post: Demo2>, <Post: Django>, <Post: demo2>, <Post: admin>]>

In [33]: Post.objects.filter(Q(title__startswith="The") | Q(title__contains='R'))
Out[33]: <QuerySet [<Post: First post>, <Post: The post>, <Post: Random>, <Post: The Web>]>
```

Django ORM & QuerySets (Order)

```
In [34]: Post.objects.order_by('created_date')
Out[34]: <QuerySet [<Post: First post>, <Post: The post>, <Post: Django>, <Post: Random>, <Post: The Web>, <Post: Demo2>, <Post: Django>, <Post: demo2>, <Post: admin>]>

In [35]: Post.objects.order_by('-created_date')
Out[35]: <QuerySet [<Post: admin>, <Post: demo2>, <Post: Django>, <Post: Demo2>, <Post: The Web>, <Post: Random>, <Post: Django>, <Post: The post>, <Post: First post>]>
```

Django ORM & QuerySets (Raw SQL)

```
In [34]: Post.objects.raw('Select * from blog_post')
Out[34]: <RawQuerySet: Select * from blog_post>
```

Forms

- In your blog app create a new file **forms.py**.

```
└── blog
    └── forms.py
```

- In **blog/forms.py**.

blog > forms.py > ...

```python
from django import forms


class PostForm(forms.Form):
    title = forms.CharField(label='Title')
    text = forms.CharField(label="Text", widget=forms.Textarea)
```

Forms

- What does Form class provide to us?

is_bound()	Checks if form has populated data.
is_valid()	Checks if the form is valid or not.
errors	List of errors for all the form’s fields.
fields	List of form’s fields.

Forms (Fields & Widgets)

CharField	TextInput, Textarea
EmailField	EmailInput
IntegerField	NumberInput
BooleanField	CheckboxInput
ChoiceField	Select
DateField	DateInput

Forms (Field Options)

Field	required, label, initial, error_messages, disabled, widget
CharField	min_length, max_length
IntegerField	min_value, max_value
ChoiceField	choices

Forms

- In **blog/templates/blog/base.html**, inside **“page-header”** div add.

```
<a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
```

- Create a new file in **blog/templates/blog/** call it **post_edit.html**, and add this.

*Notice the **csrf_token** tag it it’s very important to add as it secures your form*

blog > templates > blog > post_edit.html > ...

```
1   {% extends 'blog/base.html' %}
2
3   {% block content %}
4       <h2>New post</h2>
5       <form method="POST" class="post-form">
6           {% csrf_token %}
7           {{ form.as_p }}
8           <button type="submit" class="save btn btn-default">Save</button>
9       </form>
10  {% endblock %}
```

Forms

- In **blog/urls.py** add a new url to the urlpatterns.

```
path('post/new/', views.post_new, name='post_new'),
```

- In **blog/views.py**.

```
5   from .forms import PostForm
6
7
8 > def index(request):···
11
12 > def post_detail(request, pk):···
15
16  def post_new(request):
17      form = PostForm()
18      return render(request, 'blog/post_edit.html', {'form': form})
19
```

Forms

*This would happen if you didn't add the **csrf_token** tag in your form*

403 Forbidden

127.0.0.1:8000/post/new/

Forbidden (403)

CSRF verification failed. Request aborted.

Help

Reason given for failure:

```
CSRF token missing or incorrect.
```

In general, this can occur when there is a genuine Cross Site Request Forgery, or when Django's CSRF mechanism has not been used correctly. For POST forms, you need to ensure:

- Your browser is accepting cookies.
- The view function passes a `request` to the template's `render` method.
- In the template, there is a `{% csrf_token %}` template tag inside each POST form that targets an internal URL.
- If you are not using `CsrfViewMiddleware`, then you must use `csrf_protect` on any views that use the `csrf_token` template tag, as well as those that accept the POST data.
- The form has a valid CSRF token. After logging in in another browser tab or hitting the back button after a login, you may need to reload the page with the form, because the token is rotated after a login.

You're seeing the help section of this page because you have `DEBUG = True` in your Django settings file. Change that to `False`, and only the initial error message will be displayed.

You can customize this page using the CSRF_FAILURE_VIEW setting.

Forms

- Change your form to a model form.
- In **blog/forms.py** replace your form with this.

blog > forms.py > ...

```python
from django import forms

from .models import Post


class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        fields = ('title', 'text')

```

Forms

- To save the form, write this in **blog/views.py**.

blog > views.py > ...

```python
from django.shortcuts import render, get_object_or_404, redirect
from django.http import HttpResponse
from django.utils import timezone
from .models import Post
from .forms import PostForm


def index(request):...

def post_detail(request, pk):...

def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})

```

Edit Form

- In **blog/templates/blog/post_detail.html**, add this line before `post.title` .

```
<a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon \glyphicon-
pencil"></span></a>
```

- In **blog/urls.py** add a new url to the urlpatterns.

```
path('post/<int:pk>/edit/', views.post_edit, name='post_edit'),
```

Edit Form

- In **blog/views.py** add post_edit function.

```
def post_edit(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})
```

Generic Views

Django provides some classes which can be used as views. These allow you to structure your views and reuse code by inheriting them.

- **TemplateView:** render a given template.
- **RedirectView:** redirect to a given URL.
- **DetailView:** show full details of an object.
- **ListView:** show a collection of an object.
- **CreateView:** render a form to create an object, provides validation and updates database.
- **UpdateView:** render a form to edit an object, provides validation and updates database.
- **DeleteView:** GET method show confirmation screen, POST method delete object from database.

List View

- In **blog/views.py** replace index function with this.

```
from django.views.generic import ListView


class PostList(ListView):
    template_name = 'blog/index.html'
    context_object_name = 'posts'
    queryset = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')

```

1

2

- In **blog/urls.py**, replace the index url with this line.

```
path('', views.PostList.as_view(), name='index'),
```

1: Template nam, could be replaced by renaming your template to the same class name in snake case convention **ex: index.html → post_list.html**.

2: If you didn't use context_object_name, the default is **object_list**.

Detail View

- In **blog/views.py** replace post_detail function with this.

```
from django.views.generic import ListView, DetailView


class PostList(ListView):...


class PostDetail(DetailView):
    model = Post
```

- In **blog/urls.py**, replace the post_detail url with this line.

```
path('post/<int:pk>/', views.PostDetail.as_view(), name='post_detail'),
```

- In the detail template you can access the post by either **post** or **object** .

Create View

- In **blog/views.py** import CreateView and replace post_new function with this.

```
class PostCreate(CreateView):
    model = Post
    fields = ['title', 'text']
    template_name = 'blog/post_edit.html'

    def form_valid(self, form):
        form.instance.created_by = self.request.user
        form.instance.author = self.request.user
        form.instance.published_date = timezone.now()
        return super().form_valid(form)

    def get_success_url(self):
        return reverse('post_detail', kwargs={'pk': self.object.pk})
```

- In **blog/urls.py**, replace the post_new url with this line.

```
path('post/new/', views.PostCreate.as_view(), name='post_new'),
```

Update View

- In **blog/views.py** import UpdateView and replace post_edit function with this.

```python
class PostUpdate(UpdateView):
    model = Post
    template_name_suffix = '_edit'
    form_class = PostForm

    def form_valid(self, form):
        form.instance.created_by = self.request.user
        form.instance.author = self.request.user
        form.instance.published_date = timezone.now()
        return super().form_valid(form)

    def get_success_url(self):
        return reverse('post_detail', kwargs={'pk': self.object.pk})
```

- In **blog/urls.py**, replace the index url with this line.

```python
path('post/<int:pk>/edit/', views.PostUpdate.as_view(), name='post_edit'),
```

Delete View

- In **blog/views.py** import **DeleteView** and add this class.

```
class PostDelete(DeleteView):
    model = Post
    success_url = "/"
```

- In **blog/urls.py**, add new url.

```
path('post/<pk>/delete/', views.PostDelete.as_view(), name='post_delete'),
```

- In **post_detail.html** add this line after the dit link.

```
<a class="btn btn-default" href="{% url 'post_delete' pk=post.pk %}"><span class="glyphicon glyphicon-remove"></span></a>
```

Delete View

- By default the delete view expects to redirect to a delete confirmation page for the object.

- Make a new file **post_confirm_delete**.html.

blog > templates > blog > post_confirm_delete.html > ...

```
{% extends 'blog/base.html' %}

{% block content %}
    <form method="post">
        {% csrf_token %}
        <h2>Are you sure you want to delete "{{ object }}"?</h2>
        <input type="submit" value="Confirm">
    </form>
{% endblock %}
```