**Class-Based Views**

View in Django:

1. The ‘view’ must be ‘callable’ (да може да се извиква със скоби – list\_article())
2. Returns ‘HttpResponse’ object

Дават възможнос да се концентрираме върху различните неща – какво да направим при GET/POST, а не дали са GET/POST

Работят с наследяване, лесен начин да правим crud операции

**БАЗОВИ КЛАСОВЕ ЗА НАСЛЕДЯВАНЕ:**

Могат да се наследяват:

* views.CreateView
* views.ListView
* views.UpdateView
* views.DeleteView
* views.DetailView

**BASE VIEW**

Най-базовия клас 🡪 View

path('', list\_articles, name='list articles'), закачане на FBV  
path('cbv/', ArticlesListView.as\_view(), name='list articles cbv'), закачане на CBV

.as\_view() – от класа да получим callable, функцията as\_view() връща функция, позволява ни да пишем FBV върху класове

class BaseView:  
 def get(self, request):  
 pass  
  
 def post(self, request):  
 pass  
  
 @classmethod  
 def as\_view(cls):  
 self = cls  
  
 def view(request): ТОВА Е ФУНКЦИЯ ОТ FBV  
 if request.method == 'GET':  
 return self.get(request)  
 else:  
 return self.post(request)  
  
 return view

**views.TemplateView**

Еднакви са:

class ArticlesListView(views.TemplateView):  
 template\_name = 'articles/list.html' ЗА ДА РАБОТИ ТРЯБВА ЗАДЛЖИТЕЛНО ДА ИМА ТОЗИ АТРИБУТ

# Static data  
 extra\_context = {  
 'articles': Article.objects.all()  
 }

и:

class ArticlesListView(views.TemplateView):  
 template\_name = 'articles/list.html'  
  
 # Dynamic data  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['articles'] = Article.objects.all()  
 return context

и това: (с тази разлика , че в template ключът на context dict се казва object\_list)

class ArticlesListView(views.ListView):  
 template\_name = 'articles/list.html'  
 model = Article

**reverse() vs reverse\_lazy()**

reverse() – изпълнява веднага

reverse\_lazy() – изпълнява само когато е необходимо

**views.RedirectView**

class RedirectToArticlesView(views.RedirectView):  
 url = reverse\_lazy('list articles cbv')ЗА ДА РАБОТИ ТРЯБВА ДА ИМА URL

paginate\_by – на страница по колко записа да показва

{% if page\_obj.has\_previous %}  
 <a href="?page={{ page\_obj.previous\_page\_number }}">&lt;</a>  
{% endif %}  
{% if page\_obj.has\_next %}  
 <a href="?page={{ page\_obj.next\_page\_number }}">&gt;</a>  
{% endif %}

.DetailView- показва детайлна информация

class ArticleDetailView(views.DetailView):  
 model = Article  
 template\_name = 'articles/details.html'

в темплейта 🡪

<h1>{{ article.title }}</h1>  
<p>  
 {{ article.content }}  
</p>

Forms, based on views – 4 ways

1. Auto created (default)
2. ‘form\_class’ – return class
3. ‘get\_form\_class’ – return class
4. ‘get-form’ – return instance

CRUD Views

.CreateView()

class ArticleCreateView(views.CreateView):  
 model = Article  
 template\_name = 'articles/create.html'  
 fields = ('title', 'content')  
 success\_url = reverse\_lazy('list articles cbv')

.UpdateView()

Същото като CreateView()

.DeleteView()

Dispatch() – метод от главното View; entry point към view-то

Getattr() – дава възможност от обект да вземем неговото пропърти по име

**Authorization and Authentication**

**Authentication** – Процес, с който един потребител доказва себе си (иденцифицира се/представя се); Who are you? How you prove it?

Auth flow:

Authentication

The user sends credentials to a system

- username & password, phone number + sms code, authentication app

Authorization

After authentication, the system authorizes the user; What you are allowed to do?

MiddleWare – когато браузърът поиска view-to, middleware-a е по средата

Django по дефоулт ни дава cookie based authentication, на база на cooki-то проверява дали даден потребител съществува (чрез сесия)

NEVER name your auth app ‘auth’

В session\_data се пазят данни за потребителя

В contenttype се пазят нещата, които навързват апликейшъна към съответния модел

Взимане на User:

ВМЕСТО ТОВА:

Doncho = User.objects.get(username=’doncho’)

Context = {

‘user’: doncho,

}

ТОВА:

UserModel = get\_user\_model()

**User and Password Management**

Абсолютен минимум, за да регистрираш потребители в ап-а

class RegisterUserView(views.CreateView):  
 template\_name = 'app\_auth/register.html'  
 form\_class = auth\_forms.UserCreationForm

Ръчно настройване на рег.форма:

class RegisterUserForm(auth\_forms.UserCreationForm):  
 content = forms.BooleanField()  
 password2 = forms.CharField(  
 label=\_("Repeat password"),  
 widget=forms.PasswordInput(attrs={"autocomplete": "new-password"}),  
 strip=False,  
 help\_text=\_("Repeat password, please"),  
 )  
  
  
class RegisterUserView(views.CreateView):  
 template\_name = 'app\_auth/register.html'  
 form\_class = RegisterUserForm ТУК СЕ ЗАДАВА НАШЕТО VIEW

Ако искаме да покажем нещо допълнително ОТ МОДЕЛА (а не нещо ново), трябва д аси презапишем class Meta

Abstract=True (Този клас ще се ползва за наследяване и няма да се създаде таблица от него)

Primary\_key = True (primary key-a на единия модел ни става и primary key на другия)