



# A CAMADA VIEW DO DJANGO (PYTHON)

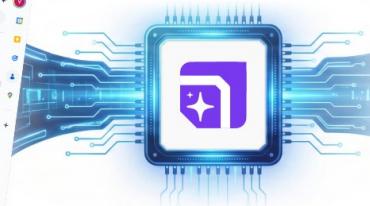
Nesse ebook, vamos tratar da camada \_View\_ do Django. Nela, nós desenvolvemos a lógica de negócio da nossa aplicação. Vamos ver sobre rotas, processamento de requisições e respostas, utilização de formulários e muito mais!

# Crie Ebooks técnicos incríveis em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



Deixe que nossa IA faça o trabalho pesado

 Syntax Highlight

 Adicione Banners Promocionais

 Edite em Markdown em Tempo Real

 Infográficos feitos por IA

**TESTE AGORA** 

Salve salve Pythonista! 🙋

Continuando a nossa série de *posts* sobre Django, no artigo de hoje vamos tratar sobre a camada *View* da sua arquitetura!

Se você ainda não leu os primeiros *posts* da série, você **COM CERTEZA** está perdendo informações importantes.

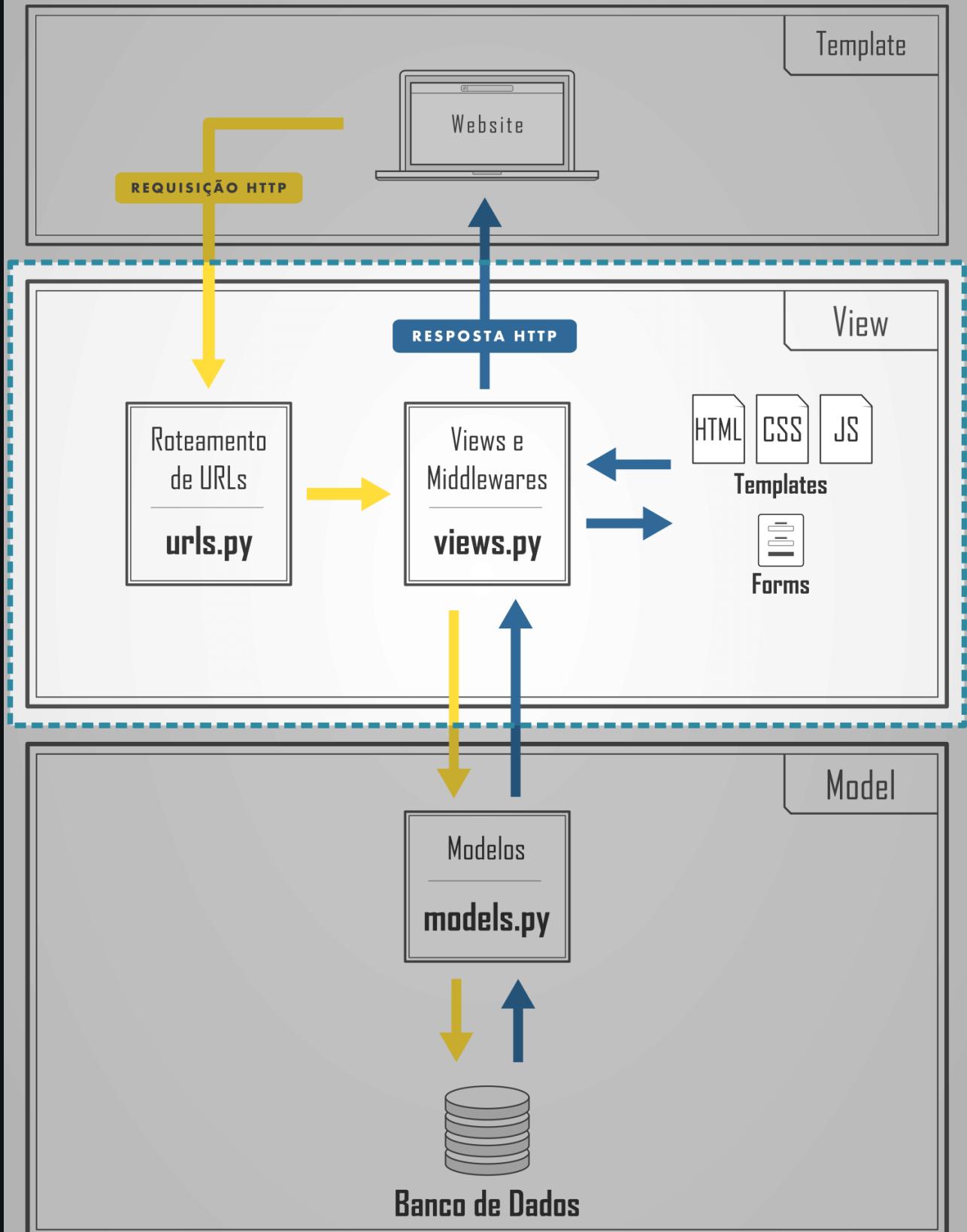
Se esse for o seu caso, então já [clica aqui pra ver o post introdutório](#) e/ou [aqui para ler o post sobre a camada Model](#) e **fique por dentro!**

Agora, faça uma **xícara de café**, ajuste sua cadeira e vamos nessa que o *post* de hoje tá **COMPLETÃO!!**

## Onde estamos...

Primeiramente, vamos nos situar:

# ARQUITETURA DO django



No [primeiro post](#), tratamos da parte introdutória do Django, uma visão geral das suas camadas, a instalação do *framework* e (**CLARO**) a criação do famoso **Hello World** *Django-based*.

Já no [segundo post](#), tratamos da camada *Model*, onde definimos as entidades do nosso sistema, a interface com o banco de dados e aprendemos como utilizar a API de acesso a dados provida pelo Django - que facilita muito nossa vida!

No *post* sobre a Camada *Model* desenvolvemos a base do nosso *HelloWorld*: um projeto de gerenciamento de funcionários.

Vamos continuar desenvolvendo-o nesse *post*, portanto, se você não tem o código desenvolvido, [baixe-o aqui][django-project-download] pois iremos utilizá-lo com o base!

Ao final do *post* se encontra o código final com o que foi passado aqui! 😊

Agora, vamos um pouco mais fundo no Django e vamos tratar da camada *View* da arquitetura **MTV** do Django (*Model Template View*).

É nessa camada que descrevemos a **lógica de negócio** da nossa aplicação!

Sem mais delongas, apresento-lhes: **A camada View!**

## Detalhes da Camada

Essa camada tem a responsabilidade de processar as **requisições** vindas dos usuários, formar uma **resposta** e enviá-la de volta ao usuário. É aqui que residem nossas **lógicas de negócio**!

Ou seja, essa camada deve: **recepçionar, processar e responder!**

Para isso, começamos pelo **roteamento de URLs**!

A partir da URL que o usuário quer acessar (`/funcionarios`, por exemplo), o Django irá rotear a requisição para quem irá tratá-la.

Mas primeiro, o Django precisa ser informado para **onde** mandar a requisição.

Fazemos isso no chamado **URLconf** e damos o nome à esse arquivo, por convenção, de `urls.py` !

Geralmente, temos um arquivo de rotas por *app* do Django. Portanto, crie um arquivo `urls.py` dentro da pasta `/helloworld` e outro na pasta `/website`.

Como o *app helloworld* é o núcleo da nossa aplicação, ele faz o papel de centralizador de rotas, isto é:

- Primeiro, a requisição cai no arquivo `/helloworld/urls.py` e é roteada para o *app* correspondente.
- Em seguida, o URLConf do *app* (`/website/urls.py`, no nosso caso) vai rotear a requisição para a *view* que irá processar dada requisição.

Dessa forma, o arquivo `helloworld/urls.py` deve conter:

```
from django.urls.conf import include
from django.contrib import admin
from django.urls import path

urlpatterns = [
    # URL padrão
    path('', include('website.urls', namespace='website')),

    # Interface administrativa
    path('admin/', admin.site.urls),
]
```

Assim, toda requisição sem o caminho (*path*) `/admin` vai ser roteado pela primeira regra: `website.urls` (URLConf do *app website*).

Em seguida, a requisição segue para o roteamento do *app* específico (`website`, no caso).

*Pode parecer complicado, mas ali embaixo, quando tratarmos mais sobre Views, vai fazer mais sentido (se não fizer, poste sua dúvida aqui embaixo)!*

A configuração do URLConf é bem simples! Basta definirmos qual função ou `View` irá processar requisições de **tal** URL.

Por exemplo, queremos que:

*Quando um usuário acesse a URL raíz `/`, o Django chame a função `index()` para processar tal requisição*

Vejamos como poderíamos configurar esse roteamento no nosso arquivo `urls.py`:

```
# Importamos a função index() definida no arquivo views.py
from . import views

app_name = 'website'

# urlpatterns a lista de roteamentos de URLs para funções/Views
urlpatterns = [
    # GET /
    url(r'^$', views.index, name='index'),
]
```

O atributo `app_name = 'website'` define o namespace do app **website** (lembre-se do décimo nono Zen do Python: **namespaces** são uma boa ideia! - [clique aqui para saber mais sobre o Zen do Python](#))

Django utilizava **Expressões Regulares** (RegEx - *Regular Expressions*) para o “cascimento” de URLs.

Já na versão 2.0 do Django, os desenvolvedores retiraram a obrigatoriedade de se utilizar RegEx!!! Agora, ao invés de fazermos isso:

```
url(r'^funcionarios/(?P<ano>[0-9]{4})/$', views.funcionarios_por_ano),
```

podemos fazer apenas:

```
path('funcionarios/<int:ano>', views.funcionarios_por_ano),
```

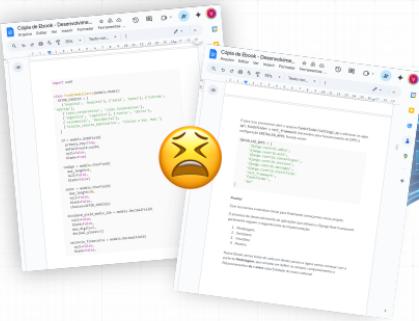
***Mais legível, né?!***



*Estou construindo o **DevBook**, uma plataforma que usa IA para criar e-books técnicos — com código formatado e exportação em PDF. Te convido a conhecê-lo!*

## Crie Ebooks técnicos incríveis em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



Deixe que nossa IA faça o trabalho pesado

Syntax Highlight

Adicione Banners Promocionais

Edite em Markdown em Tempo Real

Infográficos feitos por IA

**TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS**

## Funções vs *Class Based Views*

Com as URLs corretamente configuradas, o Django irá rotear a sua requisição para onde você definiu. No caso acima, sua requisição irá cair na função `views.funcionarios_por_ano()`.

Podemos tratar as requisições de duas formas: através de funções ou através de *Class Based Views* (**CBVs** - ou apenas *Views*).

Utilizando **funções**, você basicamente vai definir uma função que recebe como parâmetro uma requisição (`request`), realizar algum processamento e retornar alguma informação.

Já as *Views* são classes que herdam de `django.views.generic.base.View` e que agrupam diversas funcionalidades e facilitam a vida do desenvolvedor.

Nós podemos herdar e estender as funcionalidades das *Views* do Django para atender a lógica da nossa aplicação.

Por exemplo, suponha você quer criar uma tela com a **lista de todos os funcionários**.

Utilizando **funções**, você poderia fazer da seguinte forma:

```
def lista_funcionarios(request):
    # Primeiro, buscamos os funcionários
    funcionarios = Funcionario.objects.all()

    # Incluímos no contexto
    contexto = {
        'funcionarios': funcionarios
    }

    # Retornamos o template no qual os funcionários serão dispostos
    return render(request, "templates/funcionarios.html", contexto)
```

Algumas colocações:

- Toda função que vai processar requisições no Django recebe como primeiro parâmetro, um objeto `request` contendo os dados dessa requisição.
- Contexto é o conjunto de dados que estarão disponíveis no *template*.
- A função `django.shortcuts.render()` é um atalho (*shortcut*) do próprio Django que facilita a renderização de *templates*: ela recebe a própria requisição, o diretório do *template* e o contexto.

Já utilizando *Views*, podemos utilizar a *View* `django.views.generic.ListView` para listar os funcionários, da seguinte forma:

```
from django.views.generic import ListView

class ListaFuncionarios(ListView):
    template_name = "templates/funcionarios.html"
    model = Funcionario
    context_object_name = "funcionarios"
```

Perceba que você não precisou descrever a lógica para buscar a lista de funcionários?

É **exatamente isso** que as `Views` do Django proporcionam: elas descrevem o comportamento padrão para as funcionalidades mais simples (listagem, exclusão, busca simples, atualização).

O caso comum para uma listagem de objetos é buscar todo o conjunto de dados daquela entidade e mostrar no template, certo?! É exatamente **isso** que a `List-View` faz!

Com isso, um objeto `funcionarios` estará disponível, **magicamente**, no seu *template* para iteração.

Dessa forma, podemos então criar uma tabela no nosso *template* com os dados de todos os funcionários:

```
<table>
  <tbody>
    {% raw %}{% for funcionario in funcionarios %}
      <tr>
        <td>{{ funcionario.nome }}</td>
        <td>{{ funcionario.sobrenome }}</td>
        <td>{{ funcionario.remuneracao }}</td>
        <td>{{ funcionario.tempo_de_servico }}</td>
      </tr>
    {% endfor %}{% endraw %}
  </tbody>
</table>
```

*Se já quiser saber mais sobre **templates**, [acesse aqui o post sobre a Camada de Templates!](#)*

O Django tem uma diversidade enorme de *Views*, uma para cada finalidade, por exemplo:

- `CreateView` : Facilita a criação de objetos (*É o **Create** do **CRUD***)
- `DetailView` : Traz os detalhes de um objeto (*É o **Retrieve** do **CRUD***)
- `UpdateView` : Facilita a atualização de um objeto (*É o **Update** do **CRUD***)
- `DeleteView` : Facilita a deleção de objetos (*É o **Delete** do **CRUD***)

E várias outras muito úteis!

Agora vamos tratar detalhes do tratamento de requisições através de funções. Em seguida, trataremos mais sobre as *Class Based Views*.

# Funções (*Function Based Views*)

Utilizar funções é a maneira mais explícita para tratar requisições no Django (veremos que as *Class Based Views* podem ser um pouco mais complexas pois muita coisa acontece implicitamente).

Utilizando funções, geralmente tratamos primeiro o método HTTP da requisição: foi um `GET`? Foi um `POST`? Um `OPTION`?

A partir dessa informação, processamos a requisição da maneira desejada.

Vamos seguir o exemplo abaixo:

```
def cria_funcionario(request, pk):
    # Verificamos se o método POST
    if request.method == 'POST':
        form = FormularioDeCriacao(request.POST)

        if form.is_valid():
            form.save()
            return HttpResponseRedirect(reverse('list_view'))

    # Qualquer outro método: GET, OPTION, DELETE, etc...
    else:
        return render(request, templates/form.html, {'form': form})
```

O fluxo é bem simples de entender, vamos lá:

- Primeiro, conforme mencionei, verificamos o método HTTP da requisição no campo `method` do objeto `request` na **linha 3**.
- Depois instanciamos um `form` com os dados da requisição (no caso `POST`) com `FormularioDeCriacao(request.POST)` na **linha 4** (vamos falar mais sobre `Form` já já).
- Verificamos os campos do `form` com `form.is_valid()` na **linha 6..**

- Se tudo estiver **OK**, retornamos um *redirect* para uma *view* de listagem na **linha 8**.
- Se for qualquer outro método, apenas renderizamos a página novamente com `render(request, templates/form.html, {'form': form})` na **linha 12**.

Apesar de ser pouco código, foram introduzidos diversos conceitos novos. Então vamos por partes!

Vamos começar abrindo o objeto `request` para ver o que tem dentro e ver o que pode ser útil para nós.

**Observação:** Para saber mais sobre os campos do objeto `request`, dá uma olhada na classe `django.http.request.HttpRequest`!

Separarei aqui alguns atributos que provavelmente serão os mais utilizados por você:

- `request.scheme` : String representando o esquema (*HTTP* ou *HTTPS*).
- `request.path` : String com o caminho da página requisitada - exemplo: **/cursos/curso-de-python/detalhes**.
- `request.method` : Conforme citamos, contém o método *HTTP* da requisição (**GET**, **POST**, **UPDATE**, **OPTION**, etc).
- `request.content_type` : Representa o tipo MIME da requisição - `text/plain` para texto plano, `image/png` para arquivos .PNG, por exemplo - saiba mais [clicando aqui](#)
- `request.GET` : Um *dict* contendo os parâmetros GET da requisição.
- `request.POST` : Um *dict* contendo os parâmetros do corpo de uma requisição POST.

- `request.FILES`: Caso seja uma página de *upload*, contém os arquivos que foram enviados. Só contém dados se for uma requisição do tipo *POST* e o `<form>` da página *HTML* tenha o parâmetro `enctype="multipart/form-data"`.
- `request.COOKIES`: *Dict* contendo todos os *COOKIES* no formato de string.

Em 99.9% dos casos, estaremos processando dados desses campos!

**Dica:** Quer ver os campos da requisição em “tempo real” que chegaram no servidor? Então liga o debug da sua IDE (caso tenha) e coloque um breakpoint em alguma view, dispare alguma requisição e veja o que acontece! Por exemplo, utilizando o PyCharm:

```

1  def get_context_data(self, **kwargs):
2      # Get super() content
3      context = super().get_context_data()
4
5      # Get course questions
6      context['questions'] = CourseDiscussionQuestion.objects.filter(
7          course_id=context['course'].id
8      ).all()
9
10     # Put comment form on context
11     context['form'] = CommentForm()
12
13     # All lessons' progress
14     context['user_progress'] = UserCourseProgress.objects.filter(
15         user_id=self.request.user.id,
16         course=context['course'],
17     ).all()
18
19     # Last watched lesson
20     last_watched_id = UserCourseProgress.objects.filter(
21         user_id=self.request.user.id,
22         course=context['course'],
23     ).first()
24     accomplished=True
25     .aggregate(Max('lesson'))
26
27     if last_watched_id is None:
28         context['last_watched_lesson'] = None
29     else:
30         context['last_watched_lesson'] = Lesson.objects.filter(
31             id=last_watched_id.get('lesson__max')
32         ).first()
33
34     return context
35
36
37 # LESSON DETAIL VIEW
38
39
40 class LessonDetailView(DashboardMixin, DetailView):
41     template_name = 'dashboard/lesson/player.html'
42     model = Lesson
43     context_object_name = 'lesson'
44
45     def get_context_data(self, **kwargs):
46         context = super().get_context_data()
47
48         # Get course questions
49         context['questions'] = CourseDiscussionQuestion.objects.filter(
50             course_id=context['course'].id
51         ).all()
52
53         # Put comment form on context
54         context['form'] = CommentForm()
55
56         # All lessons' progress
57         context['user_progress'] = UserCourseProgress.objects.filter(
58             user_id=self.request.user.id,
59             course=context['course'],
60         ).all()
61
62         # Last watched lesson
63         last_watched_id = UserCourseProgress.objects.filter(
64             user_id=self.request.user.id,
65             course=context['course'],
66         ).first()
67         accomplished=True
68         .aggregate(Max('lesson'))
69
70         if last_watched_id is None:
71             context['last_watched_lesson'] = None
72         else:
73             context['last_watched_lesson'] = Lesson.objects.filter(
74                 id=last_watched_id.get('lesson__max')
75             ).first()
76
77         return context
78
79
80 # COURSE DETAIL VIEW
81
82 class CourseDetailView(DashboardMixin, DetailView):
83     template_name = 'dashboard/course/detail.html'
84     model = Course
85
86     def get_context_data(self, **kwargs):
87         # Get super() content
88         context = super().get_context_data()
89
90         # Get course questions
91         context['questions'] = CourseDiscussionQuestion.objects.filter(
92             course_id=context['course'].id
93         ).all()
94
95         # Put comment form on context
96         context['form'] = CommentForm()
97
98         # All lessons' progress
99         context['user_progress'] = UserCourseProgress.objects.filter(
100            user_id=self.request.user.id,
101            course=context['course'],
102        ).all()
103
104         # Last watched lesson
105         last_watched_id = UserCourseProgress.objects.filter(
106             user_id=self.request.user.id,
107             course=context['course'],
108         ).first()
109         accomplished=True
110         .aggregate(Max('lesson'))
111
112         if last_watched_id is None:
113             context['last_watched_lesson'] = None
114         else:
115             context['last_watched_lesson'] = Lesson.objects.filter(
116                 id=last_watched_id.get('lesson__max')
117             ).first()
118
119         return context
120
121
122 # COURSE LIST VIEW
123
124 class CourseListView(DashboardMixin, ListView):
125     template_name = 'dashboard/course/list.html'
126     model = Course
127
128     def get_context_data(self, **kwargs):
129         context = super().get_context_data()
130
131         # Get course questions
132         context['questions'] = CourseDiscussionQuestion.objects.filter(
133             course_id=context['course'].id
134         ).all()
135
136         # Put comment form on context
137         context['form'] = CommentForm()
138
139         # All lessons' progress
140         context['user_progress'] = UserCourseProgress.objects.filter(
141             user_id=self.request.user.id,
142             course=context['course'],
143         ).all()
144
145         # Last watched lesson
146         last_watched_id = UserCourseProgress.objects.filter(
147             user_id=self.request.user.id,
148             course=context['course'],
149         ).first()
150         accomplished=True
151         .aggregate(Max('lesson'))
152
153         if last_watched_id is None:
154             context['last_watched_lesson'] = None
155         else:
156             context['last_watched_lesson'] = Lesson.objects.filter(
157                 id=last_watched_id.get('lesson__max')
158             ).first()
159
160         return context
161
162
163 # LESSON LIST VIEW
164
165 class LessonListView(DashboardMixin, ListView):
166     template_name = 'dashboard/lesson/list.html'
167     model = Lesson
168
169     def get_context_data(self, **kwargs):
170         context = super().get_context_data()
171
172         # Get course questions
173         context['questions'] = CourseDiscussionQuestion.objects.filter(
174             course_id=context['course'].id
175         ).all()
176
177         # Put comment form on context
178         context['form'] = CommentForm()
179
180         # All lessons' progress
181         context['user_progress'] = UserCourseProgress.objects.filter(
182             user_id=self.request.user.id,
183             course=context['course'],
184         ).all()
185
186         # Last watched lesson
187         last_watched_id = UserCourseProgress.objects.filter(
188             user_id=self.request.user.id,
189             course=context['course'],
190         ).first()
191         accomplished=True
192         .aggregate(Max('lesson'))
193
194         if last_watched_id is None:
195             context['last_watched_lesson'] = None
196         else:
197             context['last_watched_lesson'] = Lesson.objects.filter(
198                 id=last_watched_id.get('lesson__max')
199             ).first()
200
201         return context
202
203
204 # COURSE INDEX VIEW
205
206 class CourseIndexView(DashboardMixin, ListView):
207     template_name = 'dashboard/course/index.html'
208     model = Course
209
210     def get_context_data(self, **kwargs):
211         context = super().get_context_data()
212
213         # Get course questions
214         context['questions'] = CourseDiscussionQuestion.objects.filter(
215             course_id=context['course'].id
216         ).all()
217
218         # Put comment form on context
219         context['form'] = CommentForm()
220
221         # All lessons' progress
222         context['user_progress'] = UserCourseProgress.objects.filter(
223             user_id=self.request.user.id,
224             course=context['course'],
225         ).all()
226
227         # Last watched lesson
228         last_watched_id = UserCourseProgress.objects.filter(
229             user_id=self.request.user.id,
230             course=context['course'],
231         ).first()
232         accomplished=True
233         .aggregate(Max('lesson'))
234
235         if last_watched_id is None:
236             context['last_watched_lesson'] = None
237         else:
238             context['last_watched_lesson'] = Lesson.objects.filter(
239                 id=last_watched_id.get('lesson__max')
240             ).first()
241
242         return context
243
244
245 # LESSON INDEX VIEW
246
247 class LessonIndexView(DashboardMixin, ListView):
248     template_name = 'dashboard/lesson/index.html'
249     model = Lesson
250
251     def get_context_data(self, **kwargs):
252         context = super().get_context_data()
253
254         # Get course questions
255         context['questions'] = CourseDiscussionQuestion.objects.filter(
256             course_id=context['course'].id
257         ).all()
258
259         # Put comment form on context
260         context['form'] = CommentForm()
261
262         # All lessons' progress
263         context['user_progress'] = UserCourseProgress.objects.filter(
264             user_id=self.request.user.id,
265             course=context['course'],
266         ).all()
267
268         # Last watched lesson
269         last_watched_id = UserCourseProgress.objects.filter(
270             user_id=self.request.user.id,
271             course=context['course'],
272         ).first()
273         accomplished=True
274         .aggregate(Max('lesson'))
275
276         if last_watched_id is None:
277             context['last_watched_lesson'] = None
278         else:
279             context['last_watched_lesson'] = Lesson.objects.filter(
280                 id=last_watched_id.get('lesson__max')
281             ).first()
282
283         return context
284
285
286 # COURSE SEARCH VIEW
287
288 class CourseSearchView(DashboardMixin, ListView):
289     template_name = 'dashboard/course/search.html'
290     model = Course
291
292     def get_context_data(self, **kwargs):
293         context = super().get_context_data()
294
295         # Get course questions
296         context['questions'] = CourseDiscussionQuestion.objects.filter(
297             course_id=context['course'].id
298         ).all()
299
300         # Put comment form on context
301         context['form'] = CommentForm()
302
303         # All lessons' progress
304         context['user_progress'] = UserCourseProgress.objects.filter(
305             user_id=self.request.user.id,
306             course=context['course'],
307         ).all()
308
309         # Last watched lesson
310         last_watched_id = UserCourseProgress.objects.filter(
311             user_id=self.request.user.id,
312             course=context['course'],
313         ).first()
314         accomplished=True
315         .aggregate(Max('lesson'))
316
317         if last_watched_id is None:
318             context['last_watched_lesson'] = None
319         else:
320             context['last_watched_lesson'] = Lesson.objects.filter(
321                 id=last_watched_id.get('lesson__max')
322             ).first()
323
324         return context
325
326
327 # LESSON SEARCH VIEW
328
329 class LessonSearchView(DashboardMixin, ListView):
330     template_name = 'dashboard/lesson/search.html'
331     model = Lesson
332
333     def get_context_data(self, **kwargs):
334         context = super().get_context_data()
335
336         # Get course questions
337         context['questions'] = CourseDiscussionQuestion.objects.filter(
338             course_id=context['course'].id
339         ).all()
340
341         # Put comment form on context
342         context['form'] = CommentForm()
343
344         # All lessons' progress
345         context['user_progress'] = UserCourseProgress.objects.filter(
346             user_id=self.request.user.id,
347             course=context['course'],
348         ).all()
349
350         # Last watched lesson
351         last_watched_id = UserCourseProgress.objects.filter(
352             user_id=self.request.user.id,
353             course=context['course'],
354         ).first()
355         accomplished=True
356         .aggregate(Max('lesson'))
357
358         if last_watched_id is None:
359             context['last_watched_lesson'] = None
360         else:
361             context['last_watched_lesson'] = Lesson.objects.filter(
362                 id=last_watched_id.get('lesson__max')
363             ).first()
364
365         return context
366
367
368 # COURSE RANKING VIEW
369
370 class CourseRankingView(DashboardMixin, ListView):
371     template_name = 'dashboard/course/ranking.html'
372     model = Course
373
374     def get_context_data(self, **kwargs):
375         context = super().get_context_data()
376
377         # Get course questions
378         context['questions'] = CourseDiscussionQuestion.objects.filter(
379             course_id=context['course'].id
380         ).all()
381
382         # Put comment form on context
383         context['form'] = CommentForm()
384
385         # All lessons' progress
386         context['user_progress'] = UserCourseProgress.objects.filter(
387             user_id=self.request.user.id,
388             course=context['course'],
389         ).all()
390
391         # Last watched lesson
392         last_watched_id = UserCourseProgress.objects.filter(
393             user_id=self.request.user.id,
394             course=context['course'],
395         ).first()
396         accomplished=True
397         .aggregate(Max('lesson'))
398
399         if last_watched_id is None:
400             context['last_watched_lesson'] = None
401         else:
402             context['last_watched_lesson'] = Lesson.objects.filter(
403                 id=last_watched_id.get('lesson__max')
404             ).first()
405
406         return context
407
408
409 # LESSON RANKING VIEW
410
411 class LessonRankingView(DashboardMixin, ListView):
412     template_name = 'dashboard/lesson/ranking.html'
413     model = Lesson
414
415     def get_context_data(self, **kwargs):
416         context = super().get_context_data()
417
418         # Get course questions
419         context['questions'] = CourseDiscussionQuestion.objects.filter(
420             course_id=context['course'].id
421         ).all()
422
423         # Put comment form on context
424         context['form'] = CommentForm()
425
426         # All lessons' progress
427         context['user_progress'] = UserCourseProgress.objects.filter(
428             user_id=self.request.user.id,
429             course=context['course'],
430         ).all()
431
432         # Last watched lesson
433         last_watched_id = UserCourseProgress.objects.filter(
434             user_id=self.request.user.id,
435             course=context['course'],
436         ).first()
437         accomplished=True
438         .aggregate(Max('lesson'))
439
440         if last_watched_id is None:
441             context['last_watched_lesson'] = None
442         else:
443             context['last_watched_lesson'] = Lesson.objects.filter(
444                 id=last_watched_id.get('lesson__max')
445             ).first()
446
447         return context
448
449
450 # COURSE RANKING INDEX VIEW
451
452 class CourseRankingIndexView(DashboardMixin, ListView):
453     template_name = 'dashboard/course/ranking/index.html'
454     model = Course
455
456     def get_context_data(self, **kwargs):
457         context = super().get_context_data()
458
459         # Get course questions
460         context['questions'] = CourseDiscussionQuestion.objects.filter(
461             course_id=context['course'].id
462         ).all()
463
464         # Put comment form on context
465         context['form'] = CommentForm()
466
467         # All lessons' progress
468         context['user_progress'] = UserCourseProgress.objects.filter(
469             user_id=self.request.user.id,
470             course=context['course'],
471         ).all()
472
473         # Last watched lesson
474         last_watched_id = UserCourseProgress.objects.filter(
475             user_id=self.request.user.id,
476             course=context['course'],
477         ).first()
478         accomplished=True
479         .aggregate(Max('lesson'))
480
481         if last_watched_id is None:
482             context['last_watched_lesson'] = None
483         else:
484             context['last_watched_lesson'] = Lesson.objects.filter(
485                 id=last_watched_id.get('lesson__max')
486             ).first()
487
488         return context
489
490
491 # LESSON RANKING INDEX VIEW
492
493 class LessonRankingIndexView(DashboardMixin, ListView):
494     template_name = 'dashboard/lesson/ranking/index.html'
495     model = Lesson
496
497     def get_context_data(self, **kwargs):
498         context = super().get_context_data()
499
500         # Get course questions
501         context['questions'] = CourseDiscussionQuestion.objects.filter(
502             course_id=context['course'].id
503         ).all()
504
505         # Put comment form on context
506         context['form'] = CommentForm()
507
508         # All lessons' progress
509         context['user_progress'] = UserCourseProgress.objects.filter(
510             user_id=self.request.user.id,
511             course=context['course'],
512         ).all()
513
514         # Last watched lesson
515         last_watched_id = UserCourseProgress.objects.filter(
516             user_id=self.request.user.id,
517             course=context['course'],
518         ).first()
519         accomplished=True
520         .aggregate(Max('lesson'))
521
522         if last_watched_id is None:
523             context['last_watched_lesson'] = None
524         else:
525             context['last_watched_lesson'] = Lesson.objects.filter(
526                 id=last_watched_id.get('lesson__max')
527             ).first()
528
529         return context
530
531
532 # COURSE RANKING DETAIL VIEW
533
534 class CourseRankingDetailView(DashboardMixin, DetailView):
535     template_name = 'dashboard/course/ranking/detail.html'
536     model = Course
537
538     def get_context_data(self, **kwargs):
539         context = super().get_context_data()
540
541         # Get course questions
542         context['questions'] = CourseDiscussionQuestion.objects.filter(
543             course_id=context['course'].id
544         ).all()
545
546         # Put comment form on context
547         context['form'] = CommentForm()
548
549         # All lessons' progress
550         context['user_progress'] = UserCourseProgress.objects.filter(
551             user_id=self.request.user.id,
552             course=context['course'],
553         ).all()
554
555         # Last watched lesson
556         last_watched_id = UserCourseProgress.objects.filter(
557             user_id=self.request.user.id,
558             course=context['course'],
559         ).first()
560         accomplished=True
561         .aggregate(Max('lesson'))
562
563         if last_watched_id is None:
564             context['last_watched_lesson'] = None
565         else:
566             context['last_watched_lesson'] = Lesson.objects.filter(
567                 id=last_watched_id.get('lesson__max')
568             ).first()
569
570         return context
571
572
573 # LESSON RANKING DETAIL VIEW
574
575 class LessonRankingDetailView(DashboardMixin, DetailView):
576     template_name = 'dashboard/lesson/ranking/detail.html'
577     model = Lesson
578
579     def get_context_data(self, **kwargs):
580         context = super().get_context_data()
581
582         # Get course questions
583         context['questions'] = CourseDiscussionQuestion.objects.filter(
584             course_id=context['course'].id
585         ).all()
586
587         # Put comment form on context
588         context['form'] = CommentForm()
589
590         # All lessons' progress
591         context['user_progress'] = UserCourseProgress.objects.filter(
592             user_id=self.request.user.id,
593             course=context['course'],
594         ).all()
595
596         # Last watched lesson
597         last_watched_id = UserCourseProgress.objects.filter(
598             user_id=self.request.user.id,
599             course=context['course'],
600         ).first()
601         accomplished=True
602         .aggregate(Max('lesson'))
603
604         if last_watched_id is None:
605             context['last_watched_lesson'] = None
606         else:
607             context['last_watched_lesson'] = Lesson.objects.filter(
608                 id=last_watched_id.get('lesson__max')
609             ).first()
610
611         return context
612
613
614 # COURSE RANKING LIST VIEW
615
616 class CourseRankingListView(DashboardMixin, ListView):
617     template_name = 'dashboard/course/ranking/list.html'
618     model = Course
619
620     def get_context_data(self, **kwargs):
621         context = super().get_context_data()
622
623         # Get course questions
624         context['questions'] = CourseDiscussionQuestion.objects.filter(
625             course_id=context['course'].id
626         ).all()
627
628         # Put comment form on context
629         context['form'] = CommentForm()
630
631         # All lessons' progress
632         context['user_progress'] = UserCourseProgress.objects.filter(
633             user_id=self.request.user.id,
634             course=context['course'],
635         ).all()
636
637         # Last watched lesson
638         last_watched_id = UserCourseProgress.objects.filter(
639             user_id=self.request.user.id,
640             course=context['course'],
641         ).first()
642         accomplished=True
643         .aggregate(Max('lesson'))
644
645         if last_watched_id is None:
646             context['last_watched_lesson'] = None
647         else:
648             context['last_watched_lesson'] = Lesson.objects.filter(
649                 id=last_watched_id.get('lesson__max')
650             ).first()
651
652         return context
653
654
655 # LESSON RANKING LIST VIEW
656
657 class LessonRankingListView(DashboardMixin, ListView):
658     template_name = 'dashboard/lesson/ranking/list.html'
659     model = Lesson
660
661     def get_context_data(self, **kwargs):
662         context = super().get_context_data()
663
664         # Get course questions
665         context['questions'] = CourseDiscussionQuestion.objects.filter(
666             course_id=context['course'].id
667         ).all()
668
669         # Put comment form on context
670         context['form'] = CommentForm()
671
672         # All lessons' progress
673         context['user_progress'] = UserCourseProgress.objects.filter(
674             user_id=self.request.user.id,
675             course=context['course'],
676         ).all()
677
678         # Last watched lesson
679         last_watched_id = UserCourseProgress.objects.filter(
680             user_id=self.request.user.id,
681             course=context['course'],
682         ).first()
683         accomplished=True
684         .aggregate(Max('lesson'))
685
686         if last_watched_id is None:
687             context['last_watched_lesson'] = None
688         else:
689             context['last_watched_lesson'] = Lesson.objects.filter(
690                 id=last_watched_id.get('lesson__max')
691             ).first()
692
693         return context
694
695
696 # COURSE RANKING INDEX DETAIL VIEW
697
698 class CourseRankingIndexDetailView(DashboardMixin, DetailView):
699     template_name = 'dashboard/course/ranking/index/detail.html'
700     model = Course
701
702     def get_context_data(self, **kwargs):
703         context = super().get_context_data()
704
705         # Get course questions
706         context['questions'] = CourseDiscussionQuestion.objects.filter(
707             course_id=context['course'].id
708         ).all()
709
710         # Put comment form on context
711         context['form'] = CommentForm()
712
713         # All lessons' progress
714         context['user_progress'] = UserCourseProgress.objects.filter(
715             user_id=self.request.user.id,
716             course=context['course'],
717         ).all()
718
719         # Last watched lesson
720         last_watched_id = UserCourseProgress.objects.filter(
721             user_id=self.request.user.id,
722             course=context['course'],
723         ).first()
724         accomplished=True
725         .aggregate(Max('lesson'))
726
727         if last_watched_id is None:
728             context['last_watched_lesson'] = None
729         else:
730             context['last_watched_lesson'] = Lesson.objects.filter(
731                 id=last_watched_id.get('lesson__max')
732             ).first()
733
734         return context
735
736
737 # LESSON RANKING INDEX DETAIL VIEW
738
739 class LessonRankingIndexDetailView(DashboardMixin, DetailView):
740     template_name = 'dashboard/lesson/ranking/index/detail.html'
741     model = Lesson
742
743     def get_context_data(self, **kwargs):
744         context = super().get_context_data()
745
746         # Get course questions
747         context['questions'] = CourseDiscussionQuestion.objects.filter(
748             course_id=context['course'].id
749         ).all()
750
751         # Put comment form on context
752         context['form'] = CommentForm()
753
754         # All lessons' progress
755         context['user_progress'] = UserCourseProgress.objects.filter(
756             user_id=self.request.user.id,
757             course=context['course'],
758         ).all()
759
760         # Last watched lesson
761         last_watched_id = UserCourseProgress.objects.filter(
762             user_id=self.request.user.id,
763             course=context['course'],
764         ).first()
765         accomplished=True
766         .aggregate(Max('lesson'))
767
768         if last_watched_id is None:
769             context['last_watched_lesson'] = None
770         else:
771             context['last_watched_lesson'] = Lesson.objects.filter(
772                 id=last_watched_id.get('lesson__max')
773             ).first()
774
775         return context
776
777
778 # COURSE RANKING INDEX INDEX VIEW
779
780 class CourseRankingIndexIndexView(DashboardMixin, ListView):
781     template_name = 'dashboard/course/ranking/index/index.html'
782     model = Course
783
784     def get_context_data(self, **kwargs):
785         context = super().get_context_data()
786
787         # Get course questions
788         context['questions'] = CourseDiscussionQuestion.objects.filter(
789             course_id=context['course'].id
790         ).all()
791
792         # Put comment form on context
793         context['form'] = CommentForm()
794
795         # All lessons' progress
796         context['user_progress'] = UserCourseProgress.objects.filter(
797             user_id=self.request.user.id,
798             course=context['course'],
799         ).all()
800
801         # Last watched lesson
802         last_watched_id = UserCourseProgress.objects.filter(
803             user_id=self.request.user.id,
804             course=context['course'],
805         ).first()
806         accomplished=True
807         .aggregate(Max('lesson'))
808
809         if last_watched_id is None:
810             context['last_watched_lesson'] = None
811         else:
812             context['last_watched_lesson'] = Lesson.objects.filter(
813                 id=last_watched_id.get('lesson__max')
814             ).first()
815
816         return context
817
818
819 # LESSON RANKING INDEX INDEX VIEW
820
821 class LessonRankingIndexIndexView(DashboardMixin, ListView):
822     template_name = 'dashboard/lesson/ranking/index/index.html'
823     model = Lesson
824
825     def get_context_data(self, **kwargs):
826         context = super().get_context_data()
827
828         # Get course questions
829         context['questions'] = CourseDiscussionQuestion.objects.filter(
830             course_id=context['course'].id
831         ).all()
832
833         # Put comment form on context
834         context['form'] = CommentForm()
835
836         # All lessons' progress
837         context['user_progress'] = UserCourseProgress.objects.filter(
838             user_id=self.request.user.id,
839             course=context['course'],
840         ).all()
841
842         # Last watched lesson
843         last_watched_id = UserCourseProgress.objects.filter(
844             user_id=self.request.user.id,
845             course=context['course'],
846         ).first()
847         accomplished=True
848         .aggregate(Max('lesson'))
849
850         if last_watched_id is None:
851             context['last_watched_lesson'] = None
852         else:
853             context['last_watched_lesson'] = Lesson.objects.filter(
854                 id=last_watched_id.get('lesson__max')
855             ).first()
856
857         return context
858
859
860 # COURSE RANKING INDEX INDEX DETAIL VIEW
861
862 class CourseRankingIndexIndexDetailView(DashboardMixin, DetailView):
863     template_name = 'dashboard/course/ranking/index/index/detail.html'
864     model = Course
865
866     def get_context_data(self, **kwargs):
867         context = super().get_context_data()
868
869         # Get course questions
870         context['questions'] = CourseDiscussionQuestion.objects.filter(
871             course_id=context['course'].id
872         ).all()
873
874         # Put comment form on context
875         context['form'] = CommentForm()
876
877         # All lessons' progress
878         context['user_progress'] = UserCourseProgress.objects.filter(
879             user_id=self.request.user.id,
880             course=context['course'],
881         ).all()
882
883         # Last watched lesson
884         last_watched_id = UserCourseProgress.objects.filter(
885             user_id=self.request.user.id,
886             course=context['course'],
887         ).first()
888         accomplished=True
889         .aggregate(Max('lesson'))
890
891         if last_watched_id is None:
892             context['last_watched_lesson'] = None
893         else:
894             context['last_watched_lesson'] = Lesson.objects.filter(
895                 id=last_watched_id.get('lesson__max')
896             ).first()
897
898         return context
899
900
901 # LESSON RANKING INDEX INDEX DETAIL VIEW
902
903 class LessonRankingIndexIndexDetailView(DashboardMixin, DetailView):
904     template_name = 'dashboard/lesson/ranking/index/index/detail.html'
905     model = Lesson
906
907     def get_context_data(self, **kwargs):
908         context = super().get_context_data()
909
910         # Get course questions
911         context['questions'] = CourseDiscussionQuestion.objects.filter(
912             course_id=context['course'].id
913         ).all()
914
915         # Put comment form on context
916         context['form'] = CommentForm()
917
918         # All lessons' progress
919         context['user_progress'] = UserCourseProgress.objects.filter(
920             user_id=self.request.user.id,
921             course=context['course'],
922         ).all()
923
924         # Last watched lesson
925         last_watched_id = UserCourseProgress.objects.filter(
926             user_id=self.request.user.id,
927             course=context['course'],
928         ).first()
929         accomplished=True
930         .aggregate(Max('lesson'))
931
932         if last_watched_id is None:
933             context['last_watched_lesson'] = None
934         else:
935             context['last_watched_lesson'] = Lesson.objects.filter(
936                 id=last_watched_id.get('lesson__max')
937             ).first()
938
939         return context
940
941
942 # COURSE RANKING INDEX INDEX INDEX VIEW
943
944 class CourseRankingIndexIndexIndexView(DashboardMixin, ListView):
945     template_name = 'dashboard/course/ranking/index/index/index.html'
946     model = Course
947
948     def get_context_data(self, **kwargs):
949         context = super().get_context_data()
950
951         # Get course questions
952         context['questions'] = CourseDiscussionQuestion.objects.filter(
953             course_id=context['course'].id
954         ).all()
955
956         # Put comment form on context
957         context['form'] = CommentForm()
958
959         # All lessons' progress
960         context['user_progress'] = UserCourseProgress.objects.filter(
961             user_id=self.request.user.id,
962             course=context['course'],
963         ).all()
964
965         # Last watched lesson
966         last_watched_id = UserCourseProgress.objects.filter(
967             user_id=self.request.user.id,
968             course=context['course'],
969         ).first()
970         accomplished=True
971         .aggregate(Max('lesson'))
972
973         if last_watched_id is None:
974             context['last_watched_lesson'] = None
975         else:
976             context['last_watched_lesson'] = Lesson.objects.filter(
977                 id=last_watched_id.get('lesson__max')
978             ).first()
979
980         return context
981
982
983 # LESSON RANKING INDEX INDEX INDEX VIEW
984
985 class LessonRankingIndexIndexIndexView(DashboardMixin, ListView):
986     template_name = 'dashboard/lesson/ranking/index/index/index.html'
987     model = Lesson
988
989     def get_context_data(self, **kwargs):
990         context = super().get_context_data()
991
992         # Get course questions
993         context['questions'] = CourseDiscussionQuestion.objects.filter(
994             course_id=context['course'].id
995         ).all()
996
997         # Put comment form on context
998         context['form'] = CommentForm()
999
1000        # All lessons' progress
1001        context['user_progress'] = UserCourseProgress.objects.filter(
1002            user_id=self.request.user.id,
1003            course=context['course'],
1004        ).all()
1005
1006        # Last watched lesson
1007        last_watched_id = UserCourseProgress.objects.filter(
1008            user_id=self.request.user.id,
1009            course=context['course'],
1010        ).first()
1011        accomplished=True
1012        .aggregate(Max('lesson'))
1013
1014        if last_watched_id is None:
1015            context['last_watched_lesson'] = None
1016        else:
1017            context['last_watched_lesson'] = Lesson.objects.filter(
1018                id=last_watched_id.get('lesson__max')
1019            ).first()
1020
1021        return context
1022
1023
1024 # COURSE RANKING INDEX INDEX INDEX DETAIL VIEW
1025
1026 class CourseRankingIndexIndexIndexDetailView(DashboardMixin, DetailView):
1027     template_name = 'dashboard/course/ranking/index/index/index/detail.html'
1028     model = Course
1029
1030     def get_context_data(self, **kwargs):
1031         context = super().get_context_data()
1032
1033         # Get course questions
1034         context['questions'] = CourseDiscussionQuestion.objects.filter(
1035             course_id=context['course'].id
1036         ).all()
1037
1038         # Put comment form on context
1039         context['form'] = CommentForm()
1040
1041         # All lessons' progress
1042         context['user_progress'] = UserCourseProgress.objects.filter(
1043             user_id=self.request.user.id,
1044             course=context['course'],
1045         ).all()
1046
1047         # Last watched lesson
1048         last_watched_id = UserCourseProgress.objects.filter(
1049             user_id=self.request.user.id,
1050             course=context['course'],
1051         ).first()
1052         accomplished=True
1053         .aggregate(Max('lesson'))
1054
1055         if last_watched_id is None:
1056             context['last_watched_lesson'] = None
1057         else:
1058             context['last_watched_lesson'] = Lesson.objects.filter(
1059                 id=last_watched_id.get('lesson__max')
1060             ).first()
1061
1062         return context
1063
1064
1065 # LESSON RANKING INDEX INDEX INDEX DETAIL VIEW
1066
1067 class LessonRankingIndexIndexIndexDetailView(DashboardMixin, DetailView):
1068     template_name = 'dashboard/lesson/ranking/index/index/index/detail.html'
1069     model = Lesson
1070
1071     def get_context_data(self, **kwargs):
1072         context = super().get_context_data()
1073
1074         # Get course questions
1075         context['questions'] = CourseDiscussionQuestion.objects.filter(
1076             course_id=context['course'].id
1077         ).all()
1078
1079         # Put comment form on context
1080         context['form'] = CommentForm()
1081
1082         # All lessons' progress
1083         context['user_progress'] = UserCourseProgress.objects.filter(
1084             user_id=self.request.user.id,
1085             course=context['course'],
1086         ).all()
1087
1088         # Last watched lesson
1089         last_watched_id = UserCourseProgress.objects.filter(
1090             user_id=self.request.user.id,
1091             course=context['course'],
1092         ).first()
1093         accomplished=True
1094         .aggregate(Max('lesson'))
1095
1096         if last_watched_id is None:
1097             context['last_watched_lesson'] = None
1098         else:
1099             context['last_watched_lesson'] = Lesson.objects.filter(
1100                 id=last_watched_id.get('lesson__max')
1101             ).first()
1102
1103         return context
1104
1105
1106 # COURSE RANKING INDEX INDEX INDEX INDEX VIEW
1107
1108 class CourseRankingIndexIndexIndexIndexView(DashboardMixin, ListView):
1109     template_name = 'dashboard/course/ranking/index/index/index/index.html'
1110     model = Course
1111
1112     def get_context_data(self, **kwargs):
1113         context = super().get_context_data()
1114
1115         # Get course questions
1116         context['questions'] = CourseDiscussionQuestion.objects.filter(
1117             course_id=context['course'].id
1118         ).all()
1119
1120         # Put comment form on context
1121         context['form'] = CommentForm()
1122
1123         # All lessons' progress
1124         context['user_progress'] = UserCourseProgress.objects.filter(
1125             user_id=self.request.user.id,
1126             course=context['course'],
1127         ).all()
1128
1129         # Last watched lesson
1130         last_watched_id = UserCourseProgress.objects.filter(
1131             user_id=self.request.user.id,
1132             course=context['course'],
1133         ).first()
1134         accomplished=True
1135         .aggregate(Max('lesson'))
1136
1137         if last_watched_id is None:
1138             context['last_watched_lesson'] = None
1139         else:
1140             context['last_watched_lesson'] = Lesson.objects.filter(
1141                 id=last_watched_id.get('lesson__max')
1142             ).first()
1143
1144         return context
1145
1146
```

Dito isso, agora vamos tratar detalhes do tratamento de requisições através de *Class Based Views*.

## Classes (CBV - *Class Based Views*)

Conforme expliquei ali em cima, as *Class Based Views* servem para automatizar e facilitar nossa vida, encapsulando funcionalidades comuns que todo desenvolvedor sempre acaba implementando. Por exemplo, geralmente:

- Queremos que quando um usuário vá para página inicial, seja mostrado **apenas uma página simples**.
- Queremos que nossa **página de listagem** contenha a **lista** de todos os funcionários (por exemplo) cadastrados no banco de dados.
- Queremos uma **página com um formulário** contendo todos os campos pré-preenchidos para **atualização** de dado funcionário.
- Queremos uma **página de exclusão** de funcionários.
- Queremos um formulário em branco para **inclusão de um novo funcionário**.

**Certo?!**

Pois é, as **CBVs** - *Class Based Views* - fazem isso!

Temos basicamente **duas formas** para utilizar uma **CBV**.

Primeiro, podemos utilizá-las diretamente no nosso **URLConf** (`urls.py`), assim:

```
from django.urls import path
from django.views.generic import TemplateView

urlpatterns = [
    path('', TemplateView.as_view(template_name="index.html")),
]
```

E a segunda maneira, a mais utilizada e mais poderosa, é herdando da *View* desejada e sobrescrever os atributos e métodos na subclasse.

*Eu particularmente prefiro a segunda forma, pois encapsula todas as Views no mesmo arquivo: no views.py*

## TemplateView

Por exemplo, para o **primeiro caso**, podemos utilizar a `TemplateView` ([documentação](#)) para apenas mostrar uma página, da seguinte forma:

```
class IndexTemplateView(TemplateView):
    template_name = "index.html"
```

E a configuração de rotas fica assim:

```
from django.urls import path
from helloworld.views import IndexTemplateView

urlpatterns = [
    path('', IndexTemplateView.as_view()),
]
```

## ListView

Já para o segundo caso, de **listagem de funcionários**, podemos utilizar a `List-View` ([documentação](#)). Nela, nós configuramos o *Model* que deve ser buscado (`Funcionario` no nosso caso), e ela automaticamente faz a busca por todos os registros presentes no banco de dados da entidade informada.

Por exemplo, a seguinte *View*:

```
from django.views.generic.list import ListView
from helloworld.models import Funcionario

class FuncionarioListView(ListView):
    template_name = "website/lista.html"
    model = Funcionario
    context_object_name = "funcionarios"
```

Com essa configuração:

```
from django.urls import path
from helloworld.views import FuncionarioListView

urlpatterns = [
    path('funcionarios/', FuncionarioListView.as_view()),
]
```

Resulta em uma página **lista.html** contendo um objeto chamado "**funcionarios**" contendo todos os Funcionários disponível para iteração.

**Dica:** Eu geralmente coloco o nome da View como sendo o Model com a CBV base. Por exemplo: se eu fosse criar uma view que vai listar todos os Cursos cadastrados, eu daria o nome de `CursoListView` (*Model*=`<Curso>`, *CBV*=`<ListView>`).

## UpdateView

Já para **aualização de usuários** podemos utilizar a `UpdateView` ([documentação](#)). Com ela, setamos (no mínimo) qual o *Model*, quais campos e qual o nome do template, e com isso temos um formulário para atualização do modelo em questão.

```
from django.views.generic.edit import UpdateView
from helloworld.models import Funcionario

class FuncionarioUpdateView(UpdateView):
    template_name = "atualiza.html"
    model = Funcionario
    fields = [
        'nome',
        'sobrenome',
        'cpf',
        'tempo_de_servico',
        'remuneracao'
    ]
```

**Dica:** Ao invés de todos os campos em `fields` em formato de lista de strings, podemos utilizar `fields = '__all__'` que o Django irá buscar todos os campos para você!

**Mas e de onde o Django vai pegar o `id` do objeto a ser buscado?**

O Django precisa ser informado do `id` ou `slug` para poder buscar o objeto correto a ser atualizado.

Podemos fazer isso de **duas formas**.

**Primeiro**, na configuração de rotas (`urls.py`), dessa forma:

```
from django.urls import path
from helloworld.views import FuncionarioUpdateView

urlpatterns = [
    # Utilizando o {id} para buscar o objeto
    path('funcionario/<id>', FuncionarioUpdateView.as_view()),

    # Utilizando o {slug} para buscar o objeto
    path('funcionario/<slug>', FuncionarioUpdateView.as_view()),
]
```

## Mas e o que é slug?

*Slug* é uma forma de gerar URLs mais legíveis a partir de dados já existentes.

Exemplo: podemos criar um campo *slug* utilizando o campo `nome` do funcionário. Dessa forma, as URLs ficariam assim:

- **/funcionario/vinicius-ramos**

e não assim (utilizando o **id** na URL):

- **/funcionario/175**

No campo *slug*, **todos os caracteres** são transformados em minúsculos e os espaços são transformados em hífens.

A **segunda forma** de buscar o objeto que estará disponível na tela de atualização é utilizando (ou **sobrescrevendo**) o método `get_object()` da classe pai `UpdateView`.

A documentação desse método traz (traduzido):

*“Retorna o objeto que a View irá mostrar. Requer self.queryset e um argumento pk ou slug no URLConf. Subclasses podem sobrescrever esse método e retornar qualquer objeto.”*

Ou seja, o Django nos dá total liberdade de utilizarmos a **convenção** (parâmetros passados pela *URLConf*) ou a **configuração** (sobrescrevendo o método `get_object()`).

Basicamente, o método `get_object()` deve pegar o `id` ou `slug` da url e realizar a busca no banco de dados até encontrar aquele `id`:

```

from django.views.generic.edit import UpdateView
from helloworld.models import Funcionario

class FuncionarioUpdateView(UpdateView):
    template_name = "atualiza.html"
    model = Funcionario
    fields = '__all__'
    context_object_name = 'funcionario'

    def get_object(self, queryset=None):
        funcionario = None

        # Se você utilizar o debug, verá que os
        # campos {pk} e {slug} estão presente em self.kwargs
        id = self.kwargs.get(self.pk_url_kwarg)
        slug = self.kwargs.get(self.slug_url_kwarg)

        if id is not None:
            # Busca o funcionario apartir do id
            funcionario = Funcionario.objects.filter(id=id).first()

        elif slug is not None:
            # Pega o campo slug do Model
            campo_slug = self.get_slug_field()

            # Busca o funcionario apartir do slug
            funcionario = Funcionario.objects.filter(**{campo_slug:
                slug}).first()

        # Retorna o objeto encontrado
        return funcionario

```

Dessa forma, os dados do funcionário de `id` ou `slug` igual ao que foi passado na URL estarão disponíveis para visualização no template `atualiza.html` utilizando o objeto `funcionario`!

*No caso geral, eu prefiro utilizar a convenção (configuração no **URLConf**).*

## DeleteView

Para deletar funcionários, utilizamos a `DeleteView` ([documentação](#)).

Sua configuração é similar à `UpdateView`: nós devemos informar via `URLConf` ou `get_object()` qual o objeto que queremos excluir.

Precisamos configurar:

- O *template* que será renderizado.
- O *model* associado à essa *view*.
- O nome do objeto que estará disponível no *template* (para confirmar ao usuário, por exemplo, o nome do funcionário que será excluído).
- A URL de retorno, caso haja sucesso na deleção do Funcionário.

Com isso, a *view* pode ser codificada da seguinte forma:

```
class FuncionarioDeleteView(DeleteView):  
    template_name = "website/exclui.html"  
    model = Funcionario  
    context_object_name = 'funcionario'  
    success_url = reverse_lazy("website:lista_funcionarios")
```

Assim como na `UpdateView`, fazemos a configuração do `id` a ser buscado no **URLConf**, da seguinte forma:

```
urlpatterns = [  
    path('funcionario/excluir/<pk>', FuncionarioDeleteView.as_view()),  
]
```

Assim, precisamos apenas fazer um *template* de confirmação da exclusão do funcionário.

Podemos fazer da seguinte forma:

```
<form method="post">
    {% raw %}{% csrf_token %}{% endraw %}

    Você tem certeza que quer excluir o funcionário <b>{{ raw }}{{ fun-
    cionario.nome }}</b>? <br><br>

    <button type="button">
        <a href="{% raw %}{% url 'lista_funcionarios' %}{% endraw %}">Cancelar</a>
    </button>
    <button>Excluir</button>
</form>
```

Algumas colocações:

- Se lembra do atributo `context_object_name`? Olha ele ali presente na quarta linha!
- A *tag* do Django `{% raw %}{% csrf_token %}{% endraw %}` é obrigatório em todos os *forms* pois está relacionado à proteção que o Django provê ao **CSRF** - *Cross Site Request Forgery* (tipo de ataque malicioso - [saiba mais aqui](#)).
- Não se preocupe com a sintaxe do *template* veremos mais sobre ele no **próximo post!**

## CreateView

Essa *view*, de criação, é bem simples!

Precisamos apenas dizer para o Django o *model*, o nome do *template*, a classe do formulário (vamos tratar mais sobre *Forms* ali embaixo) e a URL de retorno, caso o haja sucesso na inclusão do Funcionário.

Podemos fazer isso dessa forma:

```
from django.views.generic import CreateView

class FuncionarioCreateView(CreateView):
    template_name = "website/cria.html"
    model = Funcionario
    form_class = InsereFuncionarioForm
    success_url = reverse_lazy("website:lista_funcionarios")
```

`reverse_lazy()` traduz a View em URL. No nosso caso, queremos que quando haja a inclusão do Funcionário, sejamos redirecionados para a página de listagem, para podermos conferir que o Funcionário foi realmente adicionado.

E a configuração da rota no arquivo `urls.py`:

```
from django.urls import path
from helloworld.views import FuncionarioCreateView

urlpatterns = [
    path('funcionario/cadastrar/', FuncionarioCreateView.as_view()),
]
```

Com isso, estará disponível no `template` configurado (`website/cria.html`, no nosso caso), um objeto `form` contendo o formulário para criação do novo funcionário.

Podemos mostrar o formulário de duas formas.

A **primeira**, mostra o formulário inteiro **cru**, isto é, sem formatação e da forma como o Django nos entrega. Podemos mostrá-lo no nosso template da seguinte forma:

```
<form method="post">  
  {% raw %}{% csrf_token %}{% endraw %}  
  
  {% raw %}{% endraw %}  
  
  <button type="submit">Cadastrar</button>  
</form>
```

Uma observação: apesar de ser um `Form`, sua renderização não contém as *tags* `<form></form>` - cabendo a nós incluí-los no *template*.

Já a **segunda**, é mais trabalhosa, pois temos que renderizar campo a campo no *template*. Porém, nos dá um nível maior de customização.

Podemos renderizar cada campo do *form* dessa forma:

```

<form method="post">
    {% raw %}{% csrf_token %}{% endraw %}

    <label for="{% raw %}{{ form.nome.id_for_label }}{% endraw %}">Nome</label>
    {% raw %}{% form.nome %}{% endraw %}

    <label for="{% raw %}{{ form.sobrenome.id_for_label }}{% endraw %}">Sobrenome</label>
    {% raw %}{% form.sobrenome %}{% endraw %}

    <label for="{% raw %}{{ form.cpf.id_for_label }}{% endraw %}">CPF</label>
    {% raw %}{% form.cpf %}{% endraw %}

    <label for="{% raw %}{{ form.tempo_de_servico.id_for_label }}{% endraw %}">Tempo de Serviço</label>
    {% raw %}{% form.tempo_de_servico %}{% endraw %}

    <label for="{% raw %}{{ form.remuneracao.id_for_label }}{% endraw %}">Remuneração</label>
    {% raw %}{% form.remuneracao %}{% endraw %}

    <button type="submit">Cadastrar</button>
</form>

```

Dessa forma:

- `{% raw %}{{ form.campo.id_for_label }}{% endraw %}` traz o `id` da tag `<input ...>` para adicionar à tag `<label></label>`.
- Utilizamos o `{% raw %}{{ form.campo }}{% endraw %}` para renderizar um campo do formulário, e não ele inteiro.

*Antes de continuarmos, vamos respirar um pouco... **Encha sua xícara de café** que ainda tem muita coisa!*

Agora vamos detalhar mais a classe `Form`, abordada aqui em cima!

# Forms

O tratamento de formulários é uma tarefa que pode ser **bem complexa**.

Considere um formulário com diversos campos e diversas regras de validação: seu tratamento não é mais um processo simples.

Os *Forms* do Django são formas de descrever os elementos `<form> . . . </form>` das páginas HTML, simplificando e automatizando o processo de validação.

O Django trata três partes distintas dos formulários:

- Preparação dos dados tornando-os prontos para renderização
- Criação de formulários HTML para os dados
- Recepção e processamento dos formulários enviados ao servidor

Basicamente, queremos uma forma de renderizar em nosso *template* o código HTML:

```
<form action="/insere-funcionario/" method="post">
    <label for="nome">Your name: </label>
    <input id="nome" type="text" name="nome" value="">
    <input type="submit" value="Enviar">
</form>
```

Que, ao ser submetido ao servidor, tenha seus campos de entrada validados e inseridos no banco de dados.

No centro desse sistema de formulários do Django está a classe `Form`.

Nela, nós descrevemos os campos que estarão disponíveis no formulário HTML e os métodos de validação.

Para o formulário acima, podemos descrevê-lo da seguinte forma.

```
from django import forms

class InsereFuncionarioForm(forms.Form):
    nome = forms.CharField(
        label='Nome do Funcionário',
        max_length=100
    )
```

Nesse formulário:

- Utilizamos a classe `forms.CharField` para descrever um campo de texto.
- O parâmetro `label` descreve um rótulo para esse campo.
- `max_length` decreve o tamanho máximo que esse *input* pode receber (100 caracteres, no caso).

Veja os diversos tipos de campos disponíveis [acessando aqui](#)

A classe `forms.Form` possui um método muito importante, chamado `is_valid()`.

Quando um formulário é submetido ao servidor, esse é um dos métodos que irá realizar a validação dos campos do formulário.

Se tudo estiver **OK**, ele colocará os dados do formulário no atributo `cleaned_data` (que pode ser acessado por você posteriormente para pegar alguma informação - como o nome que foi inserido pelo usuário no campo `<input name='nome'>`).

Como o processo de validação do Django é bem complexo e para não prolongar muito o *post*, [acesse a documentação aqui](#) para saber mais.

Vamos ver agora um exemplo mais complexo com um formulário de inserção de um Funcionário com todos os campos.

Vamos começar criando o arquivo `forms.py` dentro do app `website` do nosso projeto usando como base os campos do `model Funcionario`.

Se você não se lembra dos campos - que descrevemos no [post](#) passado sobre a camada `Model` - aqui vão eles:

```
from django.db import models

class Funcionario(models.Model):

    nome = models.CharField(
        max_length=255,
        null=False,
        blank=False
    )

    sobrenome = models.CharField(
        max_length=255,
        null=False,
        blank=False
    )

    cpf = models.CharField(
        max_length=14,
        null=False,
        blank=False
    )

    tempo_de_servico = models.IntegerField(
        default=0,
        null=False,
        blank=False
    )

    remuneracao = models.DecimalField(
        max_digits=8,
        decimal_places=2,
        null=False,
        blank=False
    )
```

Dessa forma, e consultando a [documentação](#) dos possíveis campos do nosso formulário, nós podemos descrever um formulário de inserção da seguinte forma:

```
from django import forms

class InsereFuncionarioForm(forms.Form):

    nome = forms.CharField(
        required=True,
        max_length=255
    )

    sobrenome = forms.CharField(
        required=True,
        max_length=255
    )

    cpf = forms.CharField(
        required=True,
        max_length=14
    )

    tempo_de_servico = forms.IntegerField(
        required=True
    )

    remuneracao = forms.DecimalField(
    )
```

Affff, o Model e o Form são quase iguais... Terei que reescrever os campos toda vez?



**Claro que não, jovem!** Pra isso o Django criou o incrível `ModelForm` !!! 😊

Com o `ModelForm` nós descrevemos os campos que queremos (atributo `fields`) e/ou os campos que não queremos (atributo `exclude`) no formulário em forma de lista.

Para isso, utilizamos a classe interna `Meta` para incluirmos esses metadados na nossa classe.

Metadado (no caso do *Model* e do *Form*) é tudo aquilo que não será transformado em campo, como `model`, `fields`, `ordering` etc ([mais sobre `Meta options`](#))

Nosso `ModelForm`, pode ser descrito da seguinte forma:

```
from django import forms

class InsereFuncionarioForm(forms.ModelForm):
    class Meta:
        # Modelo base
        model = Funcionario

        # Campos que estarão no form
        fields = [
            'nome',
            'sobrenome',
            'cpf',
            'remuneracao'
        ]

        # Campos que não estarão no form
        exclude = [
            'tempo_de_servico'
        ]
```

Podemos utilizar apenas o campo `fields`, apenas o campo `exclude` ou os dois juntos.

Mesmo utilizando os atributos `fields` e `exclude`, ainda podemos adicionar outros campos, independente dos campos do *Model*.

O resultado será um formulário com todos os campos presentes no `fields`, menos os campos do `exclude` mais os outros campos que adicionarmos.

Ficou confuso? Então vamos ver o exemplo:

```

from django import forms

class InsereFuncionarioForm(forms.ModelForm):

    chefe = forms.BooleanField(
        label='Chefe?',
        required=True,
    )

    biografia = forms.CharField(
        label='Biografia',
        required=False,
        widget=forms.TextArea
    )

    class Meta:
        # Modelo base
        model = Funcionario

        # Campos que estarão no form
        fields = [
            'nome',
            'sobrenome',
            'cpf',
            'remuneracao'
        ]

        # Campos que não estarão no form
        exclude = [
            'tempo_de_servico'
        ]

```

Isso vai gerar um formulário com:

- Todos os campos contidos em `fields` menos os campos contidos em `exclude`
- O campo `forms.BooleanField`, renderizado como um `checkbox`  
`(<input type='checkbox' name='chefe' ...>)`

- Uma área de texto ( `<textarea name='biografia' ...></textarea>` )

Assim como é possível definir atributos nos modelos, os campos do formulário também são customizáveis.

Veja que o campo `biografia` é do tipo `CharField`, portanto deveria ser renderizado como um campo `<input type='text' ...>'.`

Contudo, eu modifiquei o campo setando o atributo `widget` com `forms.TextArea`.

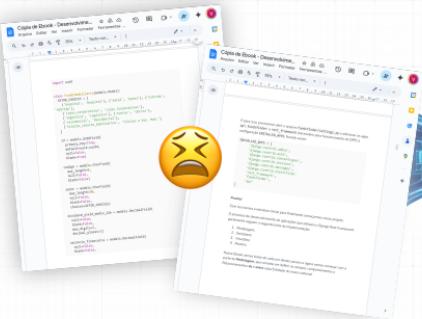
Assim, ele não mais será um simples `input`, mas será renderizado como um `<textarea></textarea>` no nosso `template`!



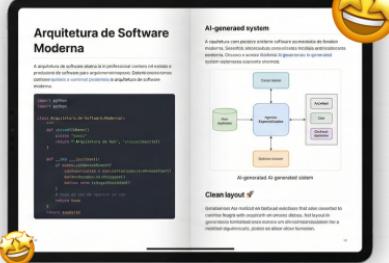
*Estou desenvolvendo o **DevBook**, uma plataforma que usa IA para gerar ebooks técnicos profissionais. Não deixe de conferir clicando no botão abaixo!*

## Crie Ebooks técnicos incríveis em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



Deixe que nossa IA faça o trabalho pesado

 Syntax Highlight

 Adicione Banners Promocionais

 Edite em Markdown em Tempo Real

 Infográficos feitos por IA

**TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS** 

## Middlewares

*Middlewares* são trechos de códigos que podem ser executados antes ou depois do processamento de requisições/respostas pelo Django.

É uma forma que os desenvolvedores, nós, temos para alterar como o Django processa algum dado de entrada ou de saída.

Se você olhar no arquivo `settings.py`, nós temos a lista `MIDDLEWARE` com diversos *middlewares* pré-configurados:

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

Por exemplo, o *middleware* `AuthenticationMiddleware` é responsável por adicionar a variável `user` a todas as requisições.

Dessa forma, você pode, por exemplo, mostrar o usuário logado no seu *template*:

```
{% raw %}  
<li>  
    <a href="{% url 'profile' id=user.id %}">Olá, {{ user.email }}</a>  
</li>  
{% endraw %}
```

Vamos ver agora como podemos criar o nosso próprio *middleware*.

Um *middleware* é um método *callable* (que tem uma implementação do método `__call__( )`) que recebe uma **requisição** e retorna uma **resposta**, assim como uma *View*, e pode ser escrito como função ou como Classe.

Um exemplo de *middleware* escrito como função é:

```

def middleware_simples(get_response):
    # Código de inicialização do Middleware

    def middleware(request):
        # Código a ser executado para cada requisição
        # antes da View, e outros middlewares, serem executada

        response = get_response(request)

        # Código a ser executado para cada requisição/resposta
        # após a execução da View que irá processar

        return response

    return middleware

```

E como Classe:

```

class MiddlewareSimples:
    def __init__(self, get_response):
        self.get_response = get_response
        # Código de inicialização do Middleware

    def __call__(self, request):
        # Código a ser executado para cada requisição
        # antes da View, e outros middlewares, serem executada

        response = self.get_response(request)

        # Código a ser executado para cada requisição/resposta
        # após a execução da View que irá processar

        return response

```

Como cada *Middleware* é executado de maneira encadeada, do topo da lista **MIDDLEWARE** para o fim, a saída de um é a entrada do próximo.

O método `get_response()` pode ser a própria *View*, caso ela seja a última configurada no `MIDDLEWARE` do `settings.py`, ou o próximo *middleware* da cadeia.

Utilizando a construção do *middleware* via Classe, nós temos três métodos importantes:

## process\_view

Assinatura: `process_view(request, func, args, kwargs)`

Esse método é chamado logo antes do Django executar a *View* que vai processar a requisição e possui os seguintes parâmetros:

- `request` é o objeto `HttpRequest`.
- `func` é a própria *view* que o Django está para chamar ao final da cadeia de *middlewares*.
- `args` é a lista de parâmetros posicionais que serão passados à *view*.
- `kwargs` é o *dict* contendo os argumentos nomeados (*keyword arguments*) que serão passados à *view*.

Esse método deve retornar `None` ou um objeto `HttpResponse`:

- Caso retorne `None`, o Django entenderá que deve continuar a cadeia de *Middlewares*.
- Caso retorne `HttpResponse`, o Django entenderá que a resposta está pronta para ser enviada de volta e não vai se preocupar em chamar o resto da cadeia de *Middlewares*, nem a *view* que iria processar a requisição.

## process\_exception

Assinatura: `process_exception(request, exception)`

Esse método é chamada quando uma *view* lança uma exceção e deve retornar ou `None` ou `HttpResponse`. Caso retorne um objeto `HttpResponse`, o Django irá aplicar o *middleware* de resposta e o de *template*, retornando a requisição ao *browser*.

- `request` é o objeto `HttpRequest`
- `exception` é a exceção propriamente dita lançada pela *view* (`Exception`).

## process\_template\_response

Assinatura: `process_template_response(request, response)`

Esse método é chamado logo após a *view* ter terminado sua execução, caso a resposta tenha uma chamada ao método `render()` indicando que a resposta possui um *template*.

Possui os seguintes parâmetros:

- `request` é um objeto `HttpRequest`.
- `response` é o objeto `TemplateResponse` retornado pela *view* ou por outro *middleware*.

Agora vamos criar um *middleware* um pouco mais complexo para exemplificar o que foi dito aqui!

Vamos supor que queremos um *middleware* que filtre requisições e só processe aquelas que venham de uma determinada lista de IP's.

O que precisamos fazer é abrir o cabeçalho de todas as requisições que chegam no nosso servidor e verificar se o IP de origem bate com a nossa lista de IP's.

Para isso, colocamos a lógica no método `process_view`, da seguinte forma:

```
class FiltraIPMiddleware:

    def __init__(self, get_response=None):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)

        return response

    def process_view(self, request, func, args, kwargs):
        # Lista de IPs autorizados
        ips_autorizados = ['127.0.0.1']

        # IP do usuário
        ip = request.META.get('REMOTE_ADDR')

        # Verifica se o IP do cliente está na lista de IPs autorizados
        if ip not in ips_autorizados:
            # Se usuário não autorizado > HTTP 403: Não Autorizado
            return HttpResponseForbidden("IP não autorizado")

        # Se for autorizado, não fazemos nada
        return None
```

Depois disso, precisamos registrar nosso *middleware* no arquivo de configurações `settings.py` (na configuração `MIDDLEWARE`):

```

MIDDLEWARE = [
    # Middlewares do próprio Django
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    # Nosso Middleware
    'helloworld.middlewares.FiltrarIPMiddleware',
]

```

Agora, podemos testar seu funcionamento alterando a lista `ips_autorizados` :

- Coloque `ips_autorizados = ['127.0.0.1']` e tente acessar alguma URL da nossa aplicação: devemos conseguir acessar normalmente nossa aplicação, pois como estamos executando o servidor localmente, nosso IP será 127.0.0.1 e, portanto, passaremos no teste.
- Coloque `ips_autorizados = []` e tente acessar alguma URL da nossa aplicação: deve aparecer a mensagem de “**IP não autorizado**”, pois nosso IP (127.0.0.1) não está autorizado a acessar o servidor.

## Código

Se quiser fazer o *download* do código desenvolvido até aqui, [clique aqui para baixá-lo][django-view-code]!

# Conclusão

Ufa! Acho melhor parar por aqui... 😞

Vimos vários conceitos sobre os tipos de *Views* (funções e classes), alguns tipos de CBV (*Class Based Views*), como mapear suas URL para suas *views* através do URL-Conf, como entender o fluxo da sua requisição utilizando o debug da sua IDE, como utilizar os poderosos `Forms` do Django, como utilizar *middlewares* para adicionar camadas extras de processamento às requisições e respostas que chegam e saem da nossa aplicação.

Com certeza ainda tem muita coisa para você descobrir e desvendar! Mas não se esqueça que qualquer dúvida que você tiver no seu processo de aprendizagem, não exite em entrar em contato comigo pelas minhas redes sociais ou pelo **box de comentário** aqui embaixo!

Espero ter facilitado seu entendimento sobre a camada *View* do Django!

No *post* sobre a Camada *Template* ([que já está disponível aqui](#)) construímos os *templates* e páginas HTML da nossa aplicação!

Quer levar esse conteúdo para onde for com nosso **ebook GRÁTIS**?

Então aproveita essa chance 

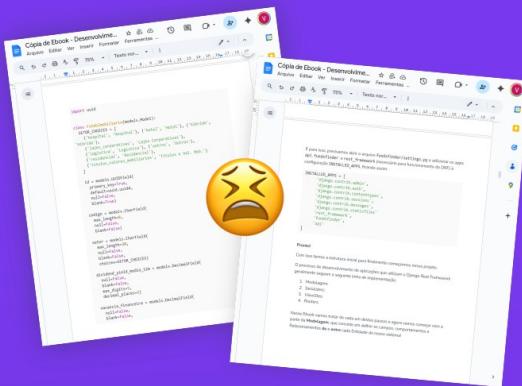
Nos vemos!

Bom desenvolvimento! 😊



# Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



**Arquitetura de Software Moderna**

A arquitetura de software alvo é profissional contendo o e-mail e produções de software para arquiteturas modernas. Oferece recursos como interface gráfica com interface de usuário.

```
import python
import python

class Arquitetura_de_Software_Moderna:
    ...
    def share(self):
        pass
    ...
    return "Arquitetura de NeXt", "arquitetura_moderna"
    ...

    def __init__(self):
        if user_authenticated():
            self.user_authenticated = user_authenticated()
            self.user_email = user_email()
            self.user_name = user_name()
        ...
        # Envie AI para gerar um código
        return type
    ...
    return self

    
```

**AI-generated system**

A arquitetura com propósito alvo é software amigável de usuários modernos. Seus recursos incluem interface gráfica amigável de usuário, interface de usuário e outras funcionalidades AI-generadas. O sistema gerado é composto por:

- 1. Webhooks
- 2. Triggers
- 3. Functions

Mais Devs ajuda a criar um ambiente perfeito e seguro para sua criação e desenvolvimento. Basta usar o botão de cima para começar.



</> Syntax Highlight



Deixe que nossa IA faça o trabalho pesado

Adicione Banners Promocionais

Edite em Markdown em Tempo Real

**TESTE AGORA**

PRIMEIRO CAPÍTULO 100% GRÁTIS