# Avaliação: Mini blog DIY Django

Nesta avaliação, você usará o conhecimento de Django que adquiriu no módulo Django Web Framework (Python) para criar um blog muito básico.

| | |
|---|---|
| **Pré-requisitos:** | Antes de tentar esta avaliação, você já deve ter trabalhado em todos os artigos deste módulo. |
| **Objetivo:** | Para testar a compreensão dos fundamentos do Django, incluindo configurações de URL, modelos, visualizações, formulários e modelos. |

## Resumo do projeto

As páginas que precisam ser exibidas, seus URLs e outros requisitos estão listados abaixo:

| Página | URL | Requisitos |
|---|---|---|
| Pagina inicial | `/` e `/blog/` | Uma página de índice que descreve o site. |

| Página | URL | Requisitos |
| --- | --- | --- |
| Lista de todas as postagens do blog | `/blog/blogs/` | Lista de todas as postagens do blog:<br>• Acessível a todos os usuários a partir de um link da barra lateral.<br>• Lista ordenada por data de postagem (da mais recente para a mais antiga).<br>• Lista paginada em grupos de 5 artigos.<br>• Os itens da lista exibem o título do blog, a data da postagem e o autor.<br>• Os nomes das postagens do blog são vinculados às páginas de detalhes do blog.<br>• O Blogger (nomes dos autores) está vinculado às páginas de detalhes do autor do blog. |

| Página | URL | Requisitos |
| --- | --- | --- |
| Página de detalhes do autor do blog (blogger) | `/blog/blogger/<author-id>` | Informações para um autor especificado (por id) e lista de suas postagens de blog:<br>• Acessível a todos os usuários a partir de links de autor em postagens de blog etc.<br>• Contém algumas informações biográficas sobre o blogueiro / autor.<br>• Lista ordenada por data de postagem (da mais recente para a mais antiga).<br>• Não paginado.<br>• Os itens da lista exibem apenas o nome da postagem do blog e a data da postagem.<br>• Os nomes das postagens do blog são vinculados às páginas de detalhes do blog. |

| Página | URL | Requisitos |
| --- | --- | --- |
| Página de detalhes da postagem do blog | `/blog/<blog-id>` | Detalhes da postagem no blog.<br>• Acessível a todos os usuários das listas de postagens do blog.<br>• A página contém a postagem do blog: nome, autor, data da postagem e conteúdo.<br>• Os comentários da postagem do blog devem ser exibidos na parte inferior.<br>• Os comentários devem ser classificados em ordem: do mais antigo ao mais recente.<br>• Contém link para adicionar comentários no final para usuários conectados (consulte a página Formulário de comentários)<br>• As postagens e comentários do blog precisam apenas exibir texto simples. Não há necessidade de suportar qualquer tipo de marcação HTML (por exemplo, links, imagens, negrito / itálico, etc). |
| Lista de todos os blogueiros | `/blog/bloggers/` | Lista de blogueiros no sistema:<br>• Acessível a todos os usuários pela barra lateral do site<br>• Os nomes do Blogger estão vinculados às páginas de detalhes do autor do blog. |

| Página | URL | Requisitos |
| --- | --- | --- |
| Página de formulário de comentários | `/blog/<blog-id>/create` | Crie um comentário para a postagem do blog:<br>• Acessível para usuários conectados (apenas) a partir do link na parte inferior das páginas de detalhes de postagem do blog.<br>• Exibe formulário com descrição para inserir comentários (data de postagem e blog não são editáveis).<br>• Depois que um comentário for postado, a página será redirecionada para a página de postagem do blog associada.<br>• Os usuários não podem editar ou excluir suas postagens.<br>• Os usuários desconectados serão direcionados à página de login para fazer o login, antes de poderem adicionar comentários. Depois de fazer login, eles serão redirecionados para a página do blog que desejam comentar.<br>• As páginas de comentários devem incluir o nome / link para a postagem do blog que está sendo comentada. |
| Páginas de autenticação do usuário | `/accounts/<standard urls>` | Páginas padrão de autenticação do Django para login, logout e configuração de senha:<br>• O login / saída deve ser acessível |

através de links da barra lateral.

| Página | URL | Requisitos |
| --- | --- | --- |
| Site de administração | `/admin/<standard urls>` | O site de administração deve ser habilitado para permitir a criação / edição / exclusão de postagens de blog, autores de blog e comentários de blog (este é o mecanismo para blogueiros criarem novas postagens de blog):<br>• Os registros de postagens de blog do site do administrador devem exibir a lista de comentários associados embutidos (abaixo de cada postagem de blog).<br>• Os nomes dos comentários no site Admin são criados truncando a descrição do comentário para 75 caracteres.<br>• Outros tipos de registros podem usar o registro básico. |

Além disso, você deve escrever alguns testes básicos para verificar:

- Todos os campos do modelo têm o rótulo e o comprimento corretos.
- Todos os modelos têm o nome do objeto esperado (por exemplo, `__str__()` retorna o valor esperado).
- Os modelos têm a URL esperada para registros individuais de Blog e Comentários (por exemplo, `get_absolute_url()` retorna a URL esperada).
- O BlogListView (página de todos os blogs) pode ser acessado no local esperado (por exemplo, / blog / blogs)
- O BlogListView (página de todos os blogs) pode ser acessado no URL nomeado esperado

(por exemplo, 'blogs')

- O BlogListView (página de todos os blogs) usa o modelo esperado (por exemplo, o padrão)
- O BlogListView pagina os registros em 5 (pelo menos na primeira página)

> **Observação**
>
> Existem, é claro, muitos outros testes que você pode executar. Use o seu critério, mas esperamos que você faça pelo menos os testes acima.

A seção a seguir mostra as capturas de tela de um site que implementa os requisitos acima.

## Capturas de tela

As capturas de tela a seguir fornecem um exemplo do que o programa finalizado deve produzir.

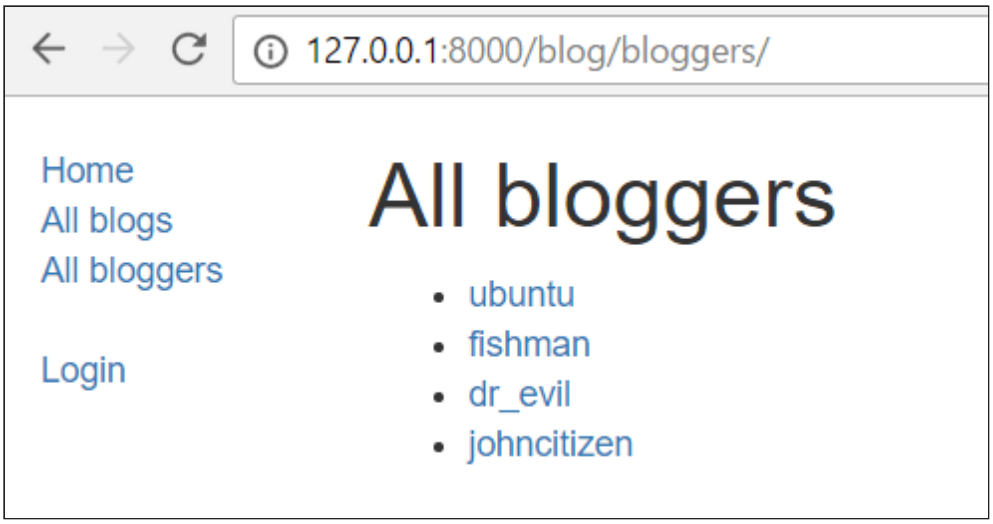
### Lista de todas as postagens do blog

Isso exibe a lista de todas as postagens do blog (acessível a partir do link "Todos os blogs" na barra lateral). Coisas a serem observadas:

- A barra lateral também lista o usuário conectado.
- Postagens de blog individuais e blogueiros podem ser acessados como links na página.
- A paginação está habilitada (em grupos de 5)
- A ordenação é da mais recente para a mais antiga.

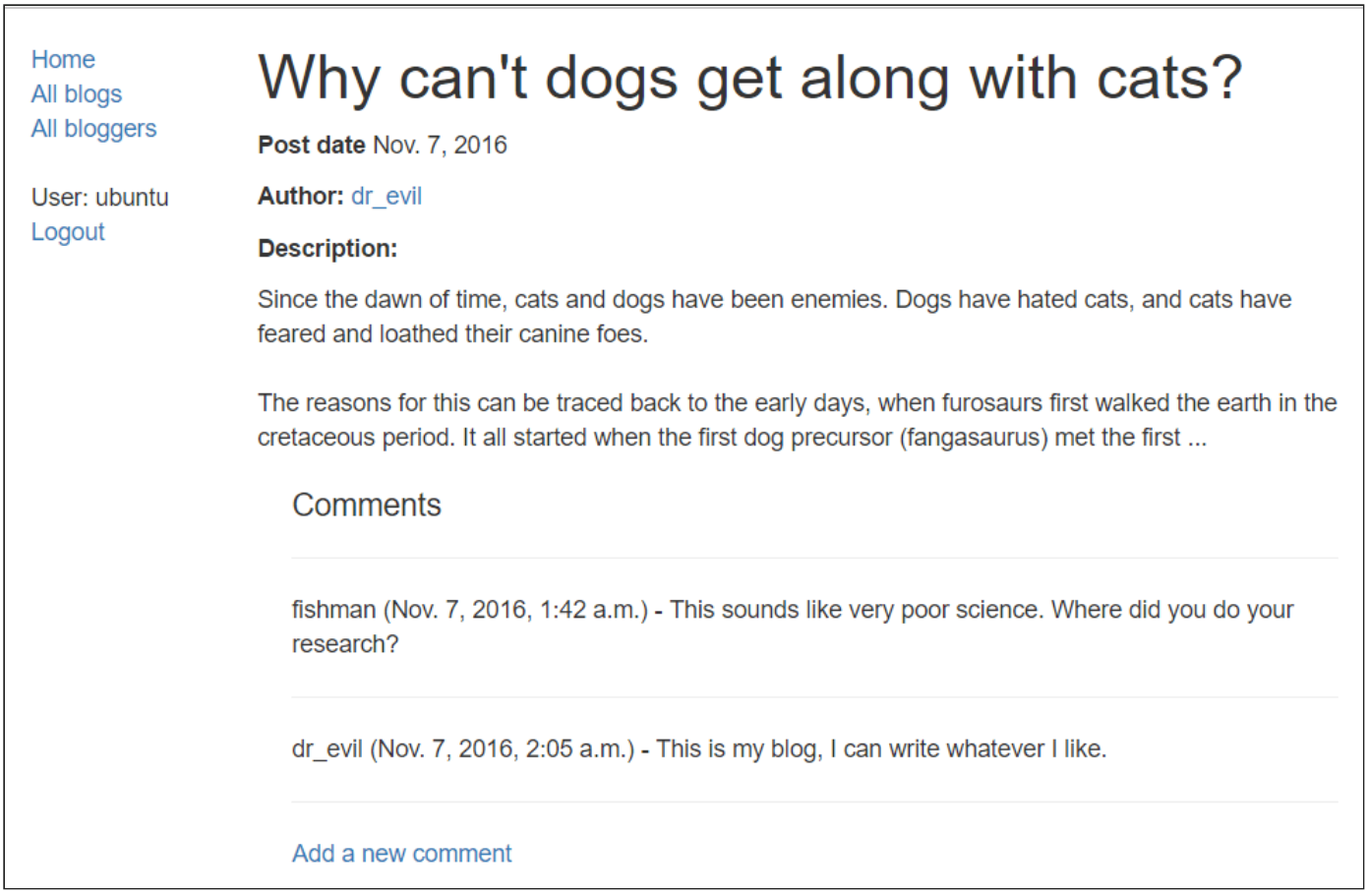

### Lista de todos os blogueiros

Isso fornece links para todos os blogueiros, conforme o link "Todos os blogueiros" na barra lateral. Neste caso, podemos ver na barra lateral que nenhum usuário está logado.

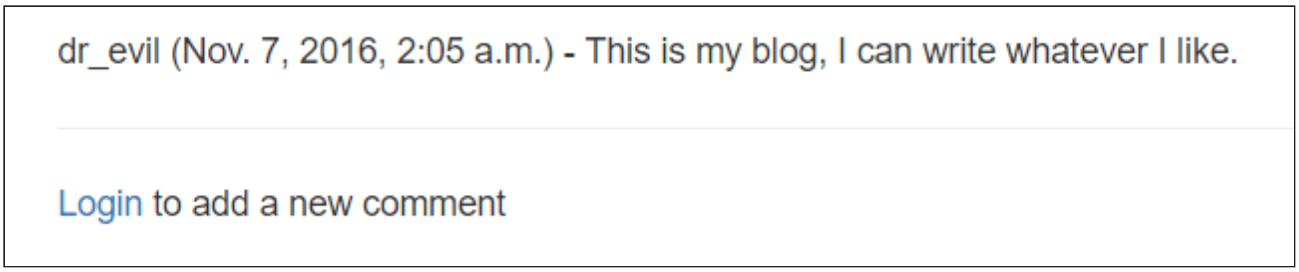



## Página de detalhes do blog

Isso mostra a página de detalhes de um determinado blog.



Note that the comments have a date *and* time, and are ordered from oldest to newest (opposite of blog ordering). At the end we have a link for accessing the form to add a new comment. If a user is not logged in we'd instead see a suggestion to log in.




## Add comment form

This is the form to add comments. Note that we're logged in. When this succeeds we should be taken back to the associated blog post page.

## Author bio

This displays bio information for a blogger along with their blog posts list.

## Steps to complete

The following sections describe what you need to do.

1. Create a skeleton project and web application for the site (as described in Django Tutorial Part 2: Creating a skeleton website). You might use 'diyblog' for the project name and 'blog' for the application name.
2. Create models for the Blog posts, Comments, and any other objects needed. When thinking about your design, remember:
    - Each comment will have only one blog, but a blog may have many comments.
    - Blog posts and comments must be sorted by post date.
    - Not every user will necessarily be a blog author though any user may be a commenter.
    - Blog authors must also include bio information.
3. Run migrations for your new models and create a superuser.
4. Use the admin site to create some example blog posts and blog comments.
5. Create views, templates, and URL configurations for blog post and blogger list pages.
6. Create views, templates, and URL configurations for blog post and blogger detail pages.
7. Create a page with a form for adding new comments (remember to make this only available to logged in users!)

## Hints and tips

This project is very similar to the LocalLibrary tutorial. You will be able to set up the skeleton, user login/logout behavior, support for static files, views, URLs, forms, base templates and admin site configuration using almost all the same approaches.

Some general hints:

1. The index page can be implemented as a basic function view and template (just like for the locallibrary).
2. The list view for blog posts and bloggers, and the detail view for blog posts can be created using the generic list and detail views.
3. The list of blog posts for a particular author can be created by using a generic blog list view and filtering for blog objects that match the specified author.
   - You will have to implement `get_queryset(self)` to do the filtering (much like in our library class `LoanedBooksAllListView`) and get the author information from the URL.
   - You will also need to pass the name of the author to the page in the context. To do this in a class-based view you need to implement `get_context_data()` (discussed below).
4. The *add comment* form can be created using a function-based view (and associated model and form) or using a generic `CreateView`. If you use a `CreateView` (recommended) then:
   - You will also need to pass the name of the blog post to the comment page in the context (implement `get_context_data()` as discussed below).
   - The form should only display the comment "description" for user entry (date and associated blog post should not be editable). Since they won't be in the form itself, your code will need to set the comment's author in the `form_valid()` function so it can be saved into the model ( as described here — Django docs). In that same function we set the associated blog. A possible implementation is shown below ( `pk` is a blog id passed in from the URL/URL configuration).

```
    def form_valid(self, form):
        """
        Add author and associated blog to form data before sett
        """
        #Add logged-in user as author of comment
        form.instance.author = self.request.user
        #Associate comment with blog based on passed id
```

```
        form.instance.blog=get_object_or_404(Blog, pk = self.kwa
        # Call super-class form validation behavior
        return super(BlogCommentCreate, self).form_valid(form)
```

- You will need to provide a success URL to redirect to after the form validates; this should be the original blog. To do this you will need to override `get_success_url()` and "reverse" the URL for the original blog. You can get the required blog ID using the `self.kwargs` attribute, as shown in the `form_valid()` method above.

We briefly talked about passing a context to the template in a class-based view in the Django Tutorial Part 6: Generic list and detail views topic. To do this you need to override `get_context_data()` (first getting the existing context, updating it with whatever additional variables you want to pass to the template, and then returning the updated context). For example, the code fragment below shows how you can add a blogger object to the context based on their `BlogAuthor` id.

```
class SomeView(generic.ListView):
    ...

    def get_context_data(self, **kwargs):
        # Call the base implementation first to get a context
        context = super(SomeView, self).get_context_data(**kwargs)
        # Get the blogger object from the "pk" URL parameter and add it
        context['blogger'] = get_object_or_404(BlogAuthor, pk = self.kwa
        return context
```

## Assessment

The assessment for this task is available on Github here. This assessment is primarily based on how well your application meets the requirements we listed above, though there are some parts of the assessment that check your code uses appropriate models, and that you have written at least some test code. When you're done, you can check out our the finished example which reflects a "full marks" project.

Once you've completed this module you've also finished all the MDN content for learning basic Django server-side website programming! We hope you enjoyed this module and feel you have a good grasp of the basics!

## In this module

- Django introduction

- Setting up a Django development environment
- Django Tutorial: The Local Library website
- Django Tutorial Part 2: Creating a skeleton website
- Django Tutorial Part 3: Using models
- Django Tutorial Part 4: Django admin site
- Django Tutorial Part 5: Creating our home page
- Django Tutorial Part 6: Generic list and detail views
- Django Tutorial Part 7: Sessions framework
- Django Tutorial Part 8: User authentication and permissions
- Django Tutorial Part 9: Working with forms
- Django Tutorial Part 10: Testing a Django web application
- Django Tutorial Part 11: Deploying Django to production
- Django web application security
- DIY Django mini blog

**Last modified:** Feb 19, 2021, by MDN contributors