Learning Objectives

- **Explain how DRF GUI is styled**
- **Identify common blocks and context variables used for styling**
- **Override the branding block and style it**
- **Explain how Swagger UI helps you build an API**
- **Add a Swagger view to Blango**
- **Explore the Blango API with Swagger UI**

Clone Blango Repo

Clone Blango Repo

Before we continue, you need to clone the `blango` repo so you have all of your code. You will need the SSH information for your repo.

In the Terminal

- Clone the repo. Your command should look something like this:

```
git clone git@github.com:<your_github_username>/blango.git
```

- You should see a `blango` directory appear in the file tree.

You are now ready for the next assignment.

Basic Customization

Basic customization

As part of this course you've had a chance to interact with the Django Rest Framework GUI. It's a great, intuitive tool, and it automatically changes the fields it displays based on the permissions available to you at the time,

Most of the time, you won't need to customize the DRF GUI at all, however we'll take a quick look at some of the things you can do.

DRF uses the Bootstrap CSS framework, although a slightly older version than what we use for the rest of the project (v3.3.5). The concepts are similar and the documentation is still available so you should be able to figure out what tools and classes you have available to use.

To customise the look of the API page you'll need to start by creating a template, `rest_framework/api.html`, which extends from `rest_framework/base.html`.

The blocks that you can then override are:

- `body`: The entire html `<body>`. You usually don't want to use this as it replaces the entire body of the page.
- `bodyclass`: Class attribute for the `<body>` tag, empty by default.
- `bootstrap_theme`: CSS for the Bootstrap theme.
- `bootstrap_navbar_variant`: CSS class for the navbar.
- `branding`: Branding section of the navbar, see Bootstrap components.
- `breadcrumbs`: Links showing resource nesting, allowing the user to go back up the resources. It's recommended to preserve these, but they can be overridden using the breadcrumbs block.
- `script`: JavaScript files for the page.
- `style`: CSS stylesheets for the page.
- `title`: Title of the page.
- `userlinks`: This is a list of links on the right of the header, by default containing login/logout links. To add links instead of replacing, use `{{ block.super }}` to preserve the authentication links.

As well as overriding these blocks, the following context variables are also passed to the template:

- `allowed_methods`: A list of methods allowed by the resource.
- `api_settings`: The API settings.
- `available_formats`: A list of formats allowed by the resource.
- `breadcrumblist`: The list of links following the chain of nested resources.
- `content`: The content of the API response.

- `description`: The description of the resource, generated from its docstring.
- `name`: The name of the resource.
- `post_form`: A form instance for use by the POST form (if allowed).
- `put_form`: A form instance for use by the PUT form (if allowed).
- `display_edit_forms`: A boolean indicating whether or not POST, PUT and PATCH forms will be displayed.
- `request`: The request object.
- `response`: The response object.
- `version`: The version of Django REST Framework.
- `view`: The view handling the request.
- `FORMAT_PARAM`: The view can accept a format override
- `METHOD_PARAM`: The view can accept a method override.

If you do want to customize the DRF GUI, most often it will be to show your own branding. Let's set up some Blango branding on our browsable API.

Try It Out

Let's make some changes to the DRF template to set our own "Blango" branding. Create a `rest_framework` directory inside the `templates` directory for your project (that is, the `templates` directory of the `blango` project, not inside the `blog` or `blango_auth` directories).

Inside this `rest_framework` directory, create a file named `api.html`.

Start by having it extend from `rest_framework/base.html`:

Open api.html

```
{% extends "rest_framework/base.html" %}
```

Next we will override the title of the page. This is the same logic that the main `api.html` template uses, to show the `name` of the page and then the name of the API. Add this under the `extends` template tag.

```
{% block title %}{% if name %}{{ name }} - {% endif %} Blango
        REST API{% endblock %}
```

Then we'll update the branding, which is displayed in the upper left corner of the page, by overriding the `branding` block. We'll set this to the *Blango REST API*. Add this next:

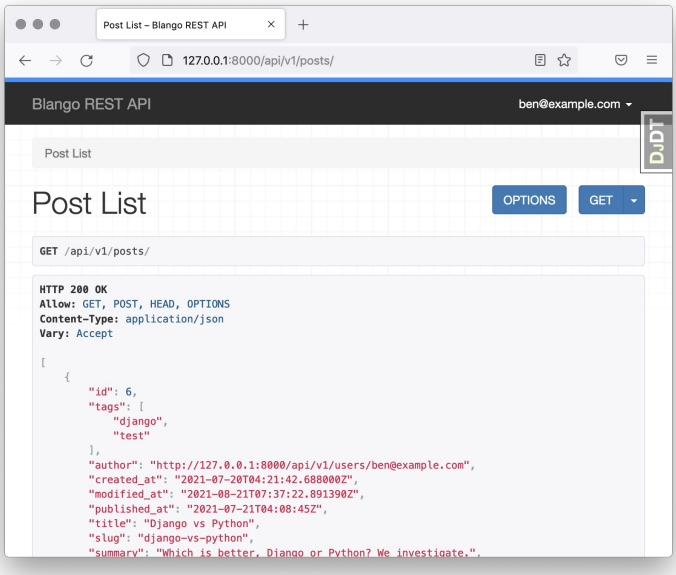
```
{% block branding %}
    <a href="/" class="navbar-brand">
        Blango REST API
    </a>
{% endblock %}
```

The final change we'll make is a small style tweak: we'll change the top border of the page from red to blue. To do this we will add a `<style>` element inside the `style` block, making sure to also include `block.super` so we retain the original style rules too. Add this to the end of the `api.html` file.

```
{% block style %}
    {{ block.super }}
    <style>
        div .navbar {
            border-top-color: #1E90FF;
        }
    </style>
{% endblock %}
```

Go ahead and load up a DRF page in your browser, you should see the changes to the title, branding and border color.

View Blog



branding changes

That's just a small sample of what can be changed, refer to the official DRF browsable API documentation for more information.

Next we'll see how to use the third-party Swagger UI.

Swagger UI

Swagger UI

Swagger is a third-party tool that generates a browsable API from an OpenAPI specification. This is an open standard, so many different APIs can generate API documents in the correct format, and use Swagger to build the UI. Different Python frameworks (like Flask) or even frameworks in other languages are compatible with OpenAPI and Swagger. You might have even used Swagger before, when working with other APIs. Because of its ubiquity you might consider providing a Swagger UI that other developers will be more familiar with than DRF's built-in one.

DRF does have built-in support for generating a simple Swagger UI, however it's a bit too basic. With the addition of a third-party library we can get a more full-featured one with only a few additions.

The library is *drf-yasg*, (*Yet another Swagger generator*), installable with `pip`.

Once installed, `drf_yasg` needs to be added to `INSTALLED_APPS` in `settings.py`. We also need to add another setting to `settings.py`: `SWAGGER_SETTINGS`. This is a dictionary to customize the Swagger UI. We'll just set some authorization settings so users can provide credentials through the Swagger UI:

```
SWAGGER_SETTINGS = {
    "SECURITY_DEFINITIONS": {
        "Token": {"type": "apiKey", "name": "Authorization",
        "in": "header"},
        "Basic": {"type": "basic"},
    }
}
```

Then, we just need to set up a couple of URL patterns.

DRF YASG implements two independent views: one that renders the specification in JSON or YAML, and another that renders the Swagger UI. Other clients can download the spec and build their own API documentation from it. Or, users can use the Swagger UI as a browsable UI to interact with our API.

First we need to generate the view class, using the `drf_yasg.views.get_schema_view` function. It takes a `drf_yasg.openapi.Info` instance to define information about the API. This

will be displayed on the Swagger UI page, so users know a bit more about our API. We're providing a `title`, `default_version` and `description`.

```
from drf_yasg import openapi
from drf_yasg.views import get_schema_view

schema_view = get_schema_view(
    openapi.Info(
        title="Blango API",
        default_version="v1",
        description="API for Blango Blog",
    ),
    public=True,
)
```

We also pass the `public=True` parameter to the `get_schema_view()` function, so that the spec includes all the API endpoints even if they might not be usable by the current client.

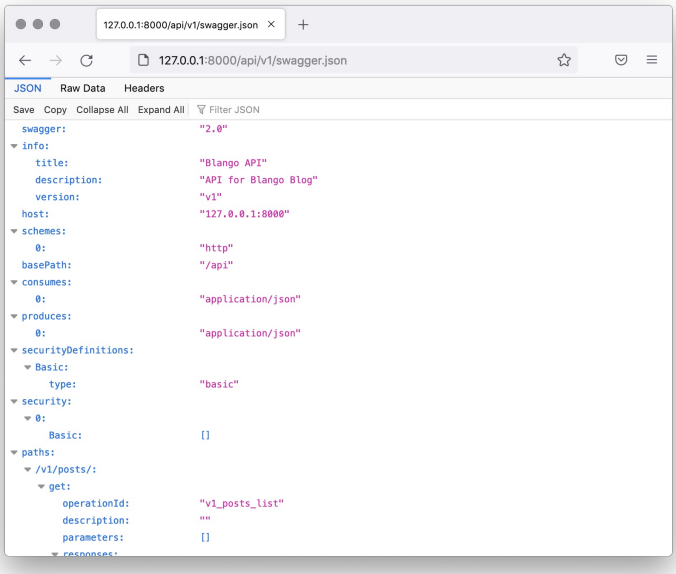
Now we can map the URLs to the view – or rather we map them to different functions on the `schema_view` generated class. First we need the `re_path` function to use regular expressions in the URL pattern.

```
from django.urls re_path
```

Then we create a URL for fetching the spec, which can be retrieved in JSON or YAML format. We're mapping the path `swagger.json` and `swagger.yaml` to the `without_ui()` method of the class.

```
urlpatterns += [
    # ...
    re_path(
        r"^swagger(?P<format>\.json|\.yaml)$",
        schema_view.without_ui(cache_timeout=0),
        name="schema-json",
    ),
    # ...
]
```

Hitting this URL will show us JSON representing all the endpoints.



open api schema

Finally we can map a URL to the Swagger UI itself. This will map the path `api/v1/swagger/` to the schema view's `with_ui()` method.

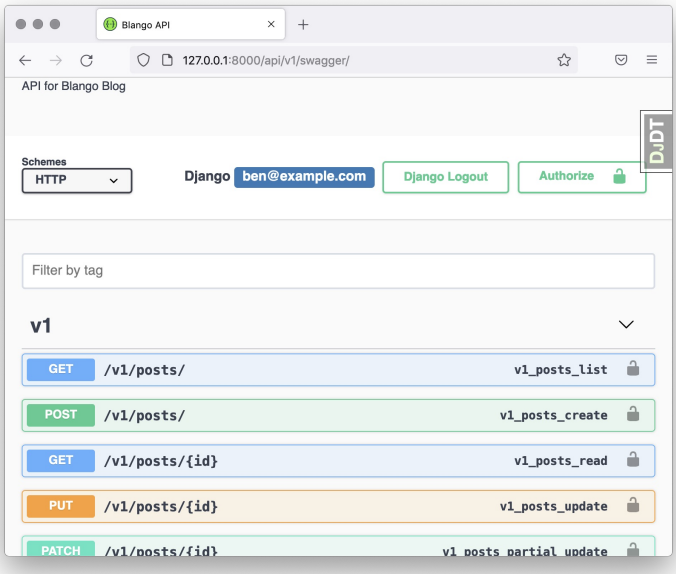
```
urlpatterns += [
    # ...
    path(
        r"^swagger/$",
        schema_view.with_ui("swagger", cache_timeout=0),
        name="schema-swagger-ui",
    ),
    # ...
]
```

We pass in `"swagger"` as the first argument to indicate we want to render a Swagger UI.

▼ **Redoc UI**

We could also pass in the string `"redoc"` to get a Redoc UI, which is a different API browsing/documentation tool.

We're all done, here's what the Swagger UI looks like in the browser.



swagger ui

Now you'll get Swagger UI up and running on your Blango project and then test it out.

Try It Out

Try It Out

Start by installing `drf-yasg` with `pip` in the terminal:

```
pip3 install drf-yasg
```

Then open `settings.py`. Add `drf_yasg` to `INSTALLED_APPS`. Then, add a new setting attribute for `SWAGGER_SETTINGS`:

Open settings.py

```
    SWAGGER_SETTINGS = {
        "SECURITY_DEFINITIONS": {
            "Token": {"type": "apiKey", "name": "Authorization",
        "in": "header"},
            "Basic": {"type": "basic"},
        }
    }
```

Now we need to map URLs to the views that DRF YASG provides. Open the `blog/api/urls.py` file, and add/update these imports at the start of the file:

Open api/urls.py

```
from django.urls import path, include, re_path
from drf_yasg import openapi
from drf_yasg.views import get_schema_view
import os
```

Then, generate the `schema_view` class:

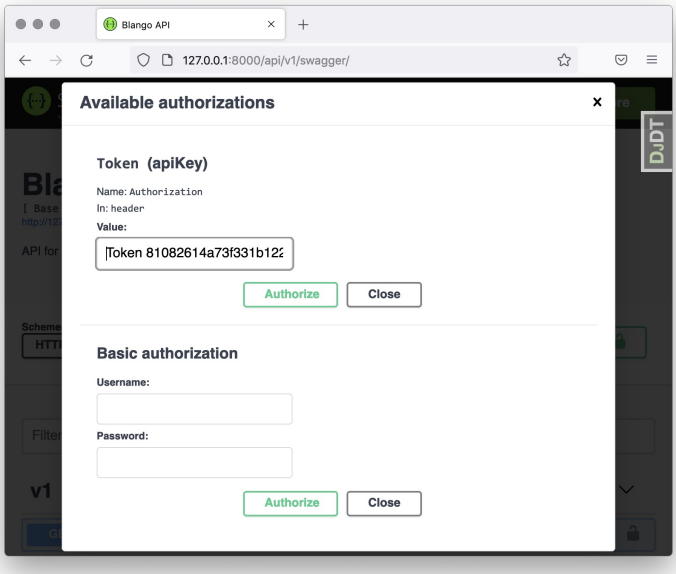
```
schema_view = get_schema_view(
    openapi.Info(
        title="Blango API",
        default_version="v1",
        description="API for Blango Blog",
    ),

        url=f"https://{os.environ.get('CODIO_HOSTNAME')}-8000.cod
        io.io/api/v1/",


    public=True,
)
```

Now we can map URLs to the methods on the `schema_view` class. Add the rules to list that is added on to the `urlpatterns`:

```
urlpatterns += [
    path("auth/", include("rest_framework.urls")),
    path("token-auth/", views.obtain_auth_token),
    re_path(
        r"^swagger(?P<format>\.json|\.yaml)$",
        schema_view.without_ui(cache_timeout=0),
        name="schema-json",
    ),
    path(
        "swagger/",
        schema_view.with_ui("swagger", cache_timeout=0),
        name="schema-swagger-ui",
    ),
]
```

Load up the Swagger UI in a browser (the path is `/api/v1/swagger/`). You can click the **Authorize** button in the top right corner to enter Basic Authentication credentials or a Token.

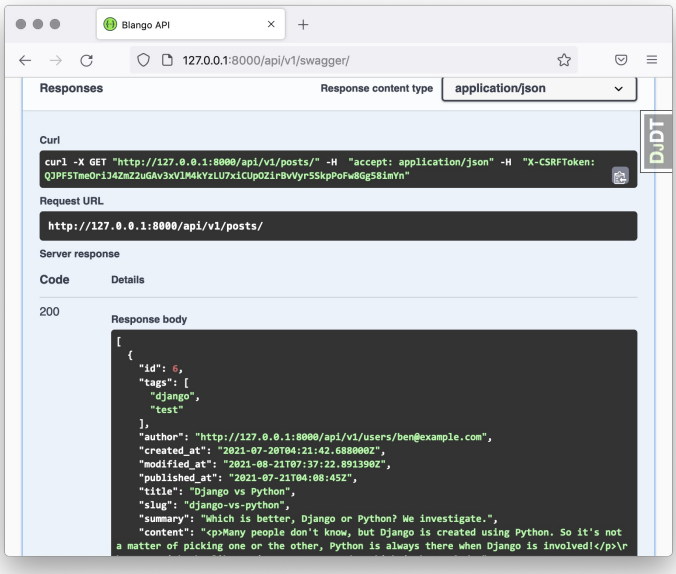
View Blog



swagger authorization

The UI also supports session authentication though, just like the DRF Browsable UI.

The Swagger UI is quite intuitive. Click a path to expand it, and see the schema details. Click **Try it out** to show fields for entering the request data, then click **Execute** to make the request. The next image shows a `Post` list response.



swagger post list response

We will finish off this module and course by looking at viewsets and routers, which can reduce the amount of code we need to write even

further.

Pushing to GitHub

Pushing to GitHub

Before continuing, you must push your work to GitHub. In the terminal:

- Commit your changes:

```
git add .
git commit -m "Finish browsable API"
```

- Push to GitHub:

```
git push
```