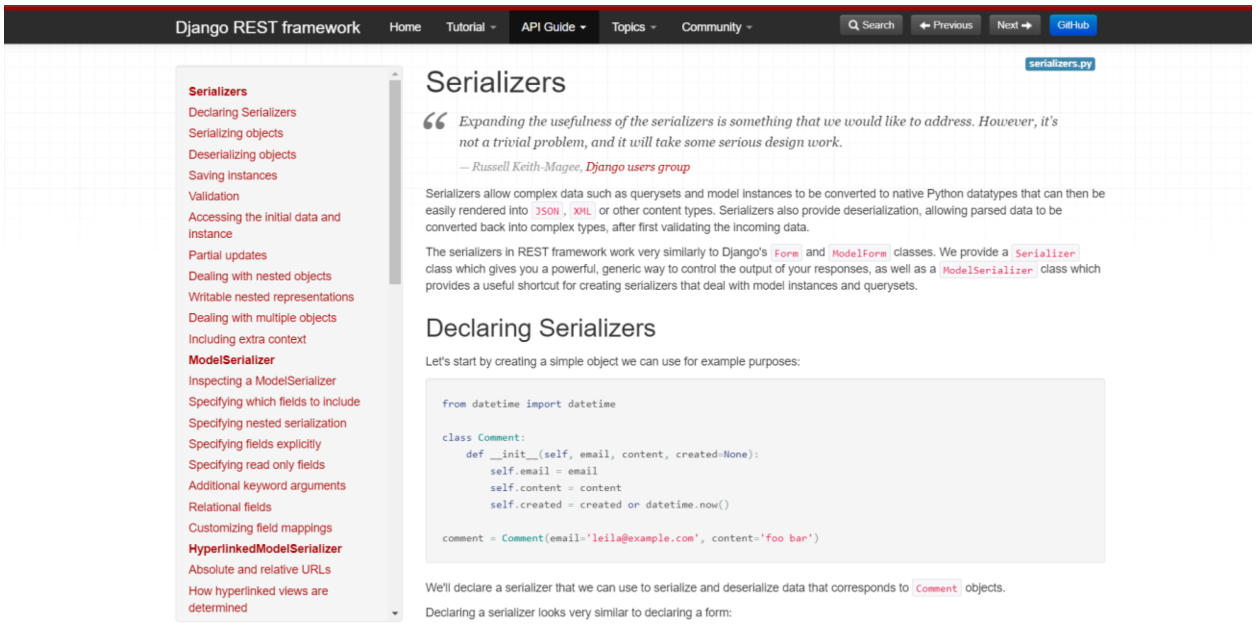
Django serializer:

https://www.django-rest-framework.org/api-guide/serializers/



Serializers

Expanding the usefulness of the serializers is something that we would like to address. However, it's not a trivial problem, and it will take some serious design work.

— Russell Keith-Magee, Django users group

Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into `JSON`, `XML` or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

The serializers in REST framework work very similarly to Django's `Form` and `ModelForm` classes. We provide a `Serializer` class which gives you a powerful, generic way to control the output of your responses, as well as a `ModelSerializer` class which provides a useful shortcut for creating serializers that deal with model instances and querysets.

Declaring Serializers

Let's start by creating a simple object we can use for example purposes:

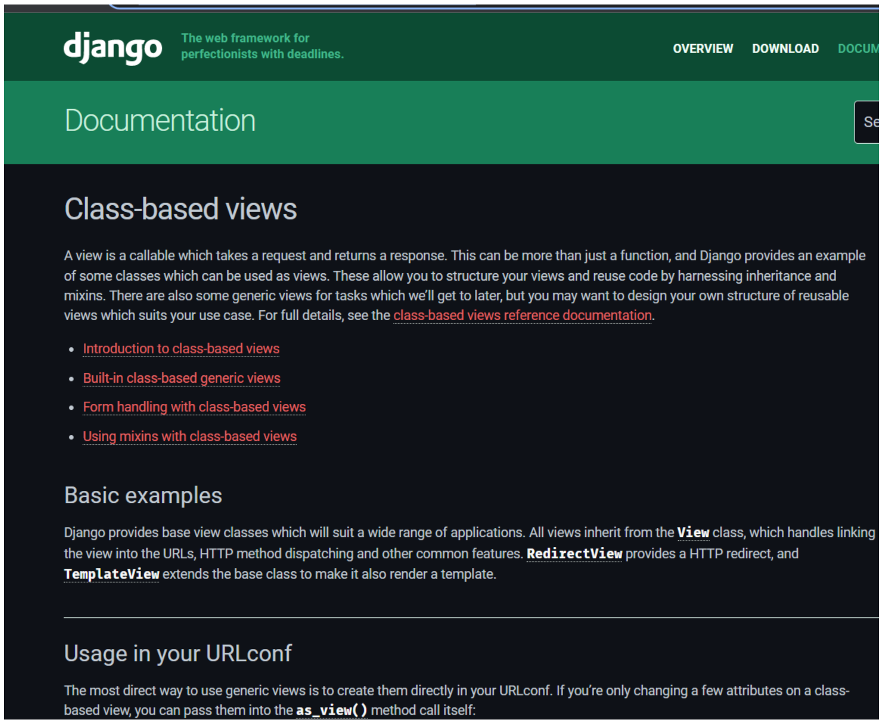
```
from datetime import datetime

class Comment:
    def __init__(self, email, content, created=None):
        self.email = email
        self.content = content
        self.created = created or datetime.now()

comment = Comment(email='leila@example.com', content='foo bar')
```

We'll declare a serializer that we can use to serialize and deserialize data that corresponds to `Comment` objects.

Declaring a serializer looks very similar to declaring a form:

Class based view:

https://docs.djangoproject.com/en/4.1/topics/class-based-views/



Class-based views

A view is a callable which takes a request and returns a response. This can be more than just a function, and Django provides an example of some classes which can be used as views. These allow you to structure your views and reuse code by harnessing inheritance and mixins. There are also some generic views for tasks which we'll get to later, but you may want to design your own structure of reusable views which suits your use case. For full details, see the class-based views reference documentation.

- Introduction to class-based views
- Built-in class-based generic views
- Form handling with class-based views
- Using mixins with class-based views

Basic examples

Django provides base view classes which will suit a wide range of applications. All views inherit from the **`View`** class, which handles linking the view into the URLs, HTTP method dispatching and other common features. **`RedirectView`** provides a HTTP redirect, and **`TemplateView`** extends the base class to make it also render a template.

Usage in your URLconf

The most direct way to use generic views is to create them directly in your URLconf. If you're only changing a few attributes on a class-based view, you can pass them into the **`as_view()`** method call itself:

API Integration:

https://www.youtube.com/watch?v=D5nVnRjPrKA&t=1047s

Carousel Slider:

https://www.youtube.com/watch?v=p3gFikowJVI&t=2s