Class-based and Generic Views

Estimated time needed: 30 minutes

Learning Objectives

- Understand class-based and generic views
- Create class-based views to handle HTTP requests and send HTTP responses
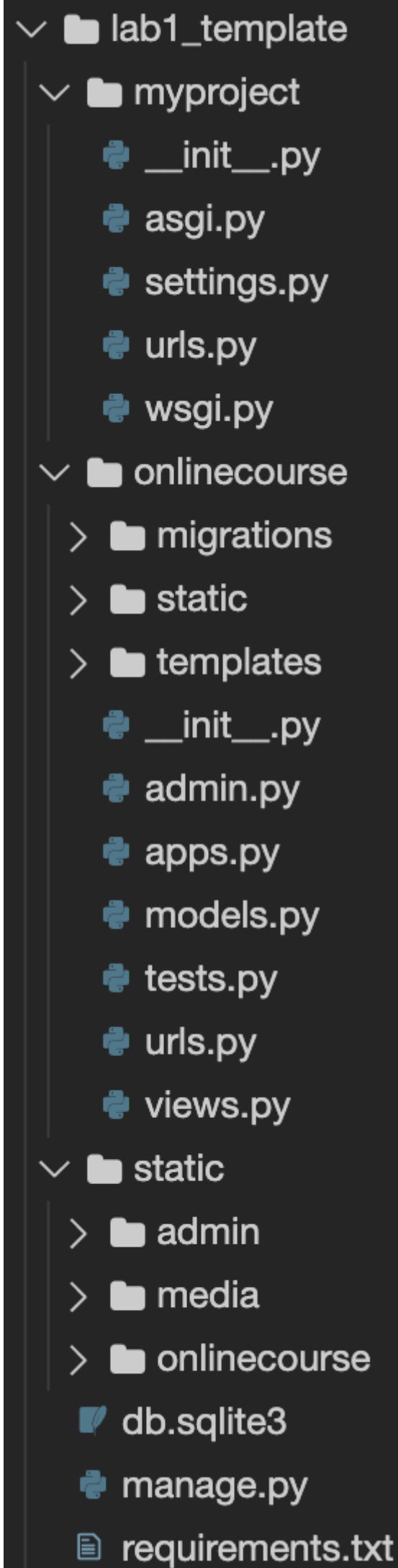
Import an `onlinecourse` App Template and Database

If the terminal was not open, go to `Terminal > New Terminal` and make sure your current Theia directory is `/home/project`.

- Run the following command-lines to download a code template for this lab

```
wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0251EN-
SkillsNetwork/labs/m5_django_advanced/lab1_template.zip"
unzip lab1_template.zip
rm lab1_template.zip
```

Your Django project should look like following:

```
∨ 📁 lab1_template
  ∨ 📁 myproject
      🐍 __init__.py
      🐍 asgi.py
      🐍 settings.py
      🐍 urls.py
      🐍 wsgi.py
  ∨ 📁 onlinecourse
    > 📁 migrations
    > 📁 static
    > 📁 templates
      🐍 __init__.py
      🐍 admin.py
      🐍 apps.py
      🐍 models.py
      🐍 tests.py
      🐍 urls.py
      🐍 views.py
  ∨ 📁 static
    > 📁 admin
    > 📁 media
    > 📁 onlinecourse
    📄 db.sqlite3
    🐍 manage.py
    📄 requirements.txt
```

First, we need to install the necessary Python packages.

- `cd` to the project folder:

```
cd lab1_template
```

```
pip3 install -r requirements.txt
```

Open `myproject/settings.py` and find `DATABASES` section and you can see that we use SQLite database in this lab, which is a file-based embedding database with some course data pre-loaded.

Next activate the models for the `onlinecourse` app.

- Perform migrations to create necessary tables:

```
python3 manage.py makemigrations
```

- and run migration to activate models for `onlinecourse` app.

```
python3 manage.py migrate
```

In our previous labs, we only created function-based views, i.e., each view is a function to receive a HTTP request and return a HTTP response. In this lab, we will be focusing on creating class-based views.

Create Class-based Views

Open `onlinecourse/views.py`, you should note that the previous function-based course list, course enrollment, and course details views have been commented out.

In this lab, we will create class-based views to return the same HTTP response for those commented out function-based views. You could compare the difference between a function-based or a class-based view.

First, let's create a class-based course list view

- Open `onlinecourse/views.py`, add a `CourseListView` class with a `get()` method to handle HTTP GET request.

```python
# Note that we are subclassing CourseListView from base View class
class CourseListView(View):

    # Handles get request
    def get(self, request):
        context = {}
        course_list = Course.objects.order_by('-total_enrollment')[:10]
        context['course_list'] = course_list
        return render(request, 'onlinecourse/course_list.html', context)
```

In the `get()` method, the top-10 popular courses were queried based on the field `total_enrollment`. The course list is appended to context and render an HTML page using `onlinecoures/course_list.html` template.

Next, we need to configure the route for the `CourseListView`

- Open `onlinecourse/urls.py`, add the following path entry to `urlpatterns` list:

```python
path(route='', view=views.CourseListView.as_view(), name='popular_course_list'),
```

Note that for the `view` argument, we actually added the `as_view()` method for `CourseListView` class. For function-based view, we use the view function name directly in `view` argument.

Next, we can try to create an enrollment class view to handle course enrollment.

- Open `onlinecourse/views.py`, add a `EnrollView` class with a `post` method to

handle HTTP POST request

```python
class EnrollView(View):

    # Handles post request
    def post(self, request, *args, **kwargs):
        course_id = kwargs.get('pk')
        course = get_object_or_404(Course, pk=course_id)
        # Increase total enrollment by 1
        course.total_enrollment += 1
        course.save()
        return HttpResponseRedirect(reverse(viewname='onlinecourse:course_details', args=(course.id,)))
```

- Open `onlinecourse/urls.py`, add the following path entry to `urlpatterns` list:

```python
path(route='course/<int:pk>/enroll/', view=views.EnrollView.as_view(), name='enroll'),
```

Same as the `CourseListView`, we added the `as_view()` method for the `view` argument.

Now we have created class-based view for returning a course list, let's start the development server to test it.

```
python3 manage.py runserver
```

- Click `Launch Application` and enter the port for the development server `8000`

When the browser tab opens, add the `/onlinecourse` path and your full URL should look like the following

`https://userid-8000.theiadocker-1.proxy.cognitiveclass.ai/onlinecourse`

You should see a course list generated by the class-based `CourseListView`

Coding Practice: Create a Class-based Course Detail View

- Complete the following code snippet to create a `CourseDetailsView` class

in `onlinecourse/views.py`:

```python
class CourseDetailsView(View):

    # Handles get request
    def get(self, request, *args, **kwargs):
        context = {}
        # We get URL parameter pk from keyword argument list as course_id
        course_id = kwargs.get('pk')
        try:
            #<HINT> Get the course object based on course_id
            #<HINT> Append the course object to context
            #<HINT> Use render method to return a HTTP response with template
        except Course.DoesNotExist:
            raise Http404("No course matches the given id.")
```

▼ Click here to see solution

```python
class CourseDetailsView(View):

    # Handles get request
    def get(self, request, *args, **kwargs):
        context = {}
        # We get URL parameter pk from keyword argument list as course_id
        course_id = kwargs.get('pk')
        try:
            course = Course.objects.get(pk=course_id)
            context['course'] = course
            return render(request, 'onlinecourse/course_detail.html', context)
        except Course.DoesNotExist:
            raise Http404("No course matches the given id.")
```

- Add its path entry to `onlinecourse/urls.py`:

```python
path(route='course/<int:pk>/', view=views.CourseDetailsView.as_view(), name='course_details'),
```

Now you can click the `Enroll` button to send a POST request to `EnrollView` and be redirected to course details page generated by `CourseDetailsView`.

Utilize Generic Built-in Views

In previous steps, we have to write the full logic to handle the GET or POST requests. For example, returning a list of objects or return the details of the object.

In fact, these are very common user scenarios for most web apps and should be abstracted and easily reused to similar scenarios. To facilitate app development, Django provides developers with many commonly used view templates/super-classes called Generic Views.

Now, let's try to replace the class-based views we created in the previous step with the generic class view.

- Open `onlinecourse/views.py`, comment out both `CourseListView` and `CourseDetailsView` classes, and add

the following generic class views:

```
# Note that CourseListView is subclassing from generic.ListView instead of View
# so that it can use attributes and override methods from ListView such as get_queryset()
class CourseListView(generic.ListView):
    template_name = 'onlinecourse/course_list.html'
    context_object_name = 'course_list'

    # Override get_queryset() to provide list of objects
    def get_queryset(self):
        courses = Course.objects.order_by('-total_enrollment')[:10]
        return courses
```

The `CourseListView` is a subclass of `generic.ListView`. By subclassing `ListView` class, the newly added `CourseListView` inherites many useful fields and methods to quickly build a list view.

Here, we just need to specify the `template` and `context_object_name` and override the `def get_queryset(self)` method to query a course list. The method's return, i.e., the obtained course list will be append into the context called `course_list` automatically.

This implementation is much simpler than both function-based or class-based views we created.

- Similarly, we can create a `CourseDetailsView` by subclassing a generic `generic.Details` view:

```
# Note that CourseDetailsView is now subclassing DetailView
class CourseDetailsView(generic.DetailView):
    model = Course
    template_name = 'onlinecourse/course_detail.html'
```

The `CourseDetailsView` is even simpler to use as we just need to specify the `model` to be `Course` and `template_name` to be `onlinecourse/course_detail.html`.

Summary

In this lab, you have learned how to implement the app features such as returning a list of courses and course details using class-based and generic views. You may compare the commented-out function-based views with the built-in generic views and understand how class-based views help reduce the workload by abstracting the common tasks in super classes.

Author(s)

[Yan Luo](linkedin.com/in/yan-luo-96288783)

Changelog

Date	Version	Changed by	Change Description
14-Dec-2020	1.0	Yan Luo	Initial version created