Class-based and Generic Views

Estimated time needed: 30 minutes

In this lab, you will create your class-based and generic views.

Learning Objectives

- Understand class-based and generic views
- Create class-based views to handle HTTP requests and send HTTP responses

Working with files in Cloud IDE

If you are new to Cloud IDE, this section will show you how to create and edit files, which are part of your project, in Cloud IDE.

To view your files and directories inside Cloud IDE, click on this files icon to reveal it.

Click on `New`, and then `New Terminal`.

This will open a new terminal where you can run your commands.

Concepts covered in the lab

1. `Class-based views` (CBVs): In Django, class-based views refer to a method of defining views using classes instead of functions. CBVs provide a more structured and reusable approach to handling HTTP requests and generating responses.

2. `Generic views`: Generic views in Django are a type of class-based view that offers pre-defined, reusable views for common use cases. They aim to reduce repetitive code and simplify common web development patterns.

3. `Function-based views` (FBVs): In Django, function-based views are a way of defining views using regular Python functions. They are the simplest and most basic approach for handling HTTP requests and generating responses in Django.

Import an onlinecourse App Template and Database

1. 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

```
1
2
3
```

```
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
```

Copied! Executed!

Your Django project should look like following:

2. First, we need to install the necessary Python packages.

- Change to the project folder:

```
1
```

```
cd lab1_template
```

Copied! Executed!

- Install these must-have packages and setup the environment.

```
1
2
3
4
5
```

```
pip install --upgrade distro-info
pip3 install --upgrade pip==23.2.1
pip install virtualenv
virtualenv djangoenv
source djangoenv/bin/activate
```

Copied! Executed!

```
1
```

```
1. pip install -r requirements.txt
```

Copied! Executed!

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.

3. Next activate the models for the `onlinecourse` app.

- Perform migrations to create necessary tables:

```
1. 1
```

```
1. python3 manage.py makemigrations
```

Copied! Executed!

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

```
1. 1
```

```
1. python3 manage.py migrate
```

Copied! Executed!

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.

1. 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.

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
6
7
8
9
```

```
# 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)
```

Copied!

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.

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

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

```
1
```

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

Copied!

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.

3. 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

```
1
2
3
4
5
6
7
8
9
10
```

```
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,)))
```

Copied!

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

```
1
```

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

Copied!

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

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

```
1
```

```
python3 manage.py runserver
```

Copied! Executed!

- Click on the Skills Network button on the left, it will open the **"Skills Network Toolbox"**. Then click the **Other** then **Launch Application**. From there you should be able to enter the port `8000` and launch.

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`:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
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.")
```

Copied!

▼ Click here to see solution

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
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.")
```

Copied!

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

```
1
```

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

Copied!

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:

```
1
2
3
4
5
6
7
8
9
10
```

```
# 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
```

Copied!

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:

```
1
2
3
4
```

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

Copied!

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 were taught how to incorporate features into an app by utilizing class-based and generic views. By examining the commented-out function-based views and comparing them to the built-in generic views, you were able to observe how class-based views streamline the development process by abstracting common tasks in super classes. This abstraction helps reduce the workload and simplifies the implementation of app features.

Author(s)

Yan Luo