Views and templates

Estimated time needed: 40 minutes

In this lab, you will learn to create views and templates, use views to update objects and redirect users to other pages, and use CSS to stylize your templates.

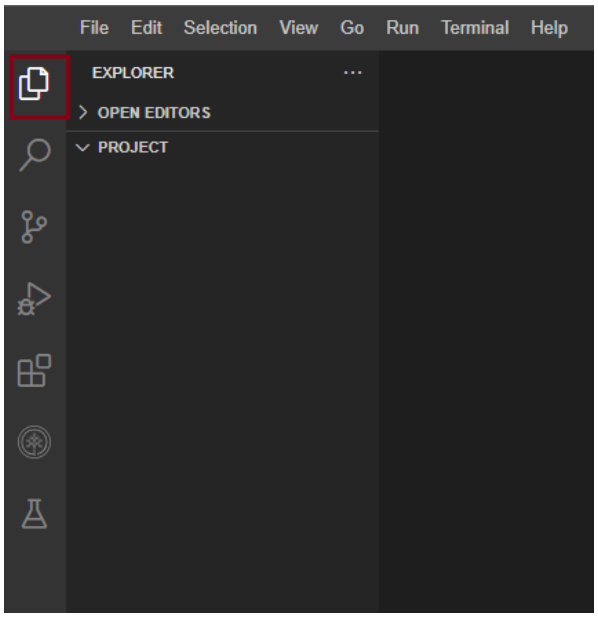
Learning Objectives

- Create function-based views to handle HTTP requests and return HTTP responses
- Create templates for rendering HTML pages

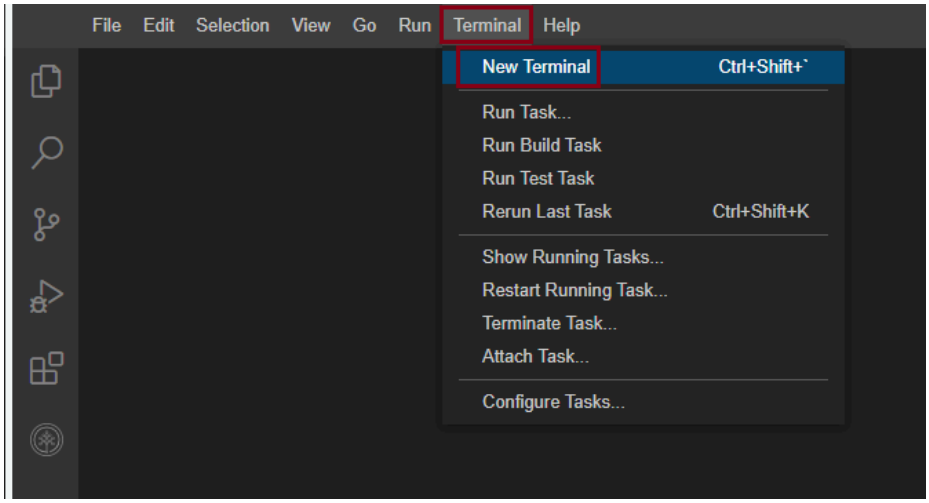
Working with files in Cloud IDE

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.

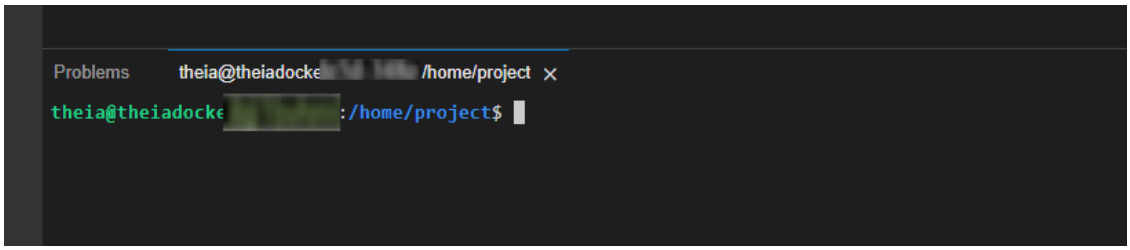
To view your files and directories inside Cloud IDE, click on this files icon to reveal it.



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



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



Concepts covered in the lab

1. `SQLite`: A lightweight relational database management system (RDBMS) that provides an SQL-based interface to store, manage, and retrieve structured data.
2. `List view`: A class-based view that provides a convenient way to fetch and present multiple records in a tabular or list format.
3. `HttpResponseRedirect`: A class that is used to redirect users to a different page or URL after processing a request.
4. `reverse()`: A utility function provided by Django that takes the name of a URL pattern as an argument and returns the corresponding URL as a string.
5. `CSS container class`: Creates a responsive fixed-width container for webpage content and provides a consistent layout and spacing for your content across different screen sizes and devices.
6. `CSS card class`: Creates a flexible and stylized container called a “card” to present information, images, and actions in a compact and organized manner.

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

```
1
2
3
```

```
wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0251EN-SkillsNetwork/labs/m4_django_app/lab3_template.zip"
unzip lab3_template.zip
rm lab3_template.zip
```

Copied! Executed!

- `cd` to the project folder

```
1
```

```
cd lab3_template
```

Copied! Executed!

- Next, install required packages for this lab:

```
1
2
3
4
5
6
```

```
pip install --upgrade distro-info
pip3 install --upgrade pip==23.2.1
pip install virtualenv
virtualenv djangoenv
source djangoenv/bin/activate
pip install -U -r requirements.txt
```

Copied! Executed!

Your Django project should look like following:

Open `myproject/settings.py` and find `DATABASES` section and you will notice that we use SQLite this lab.

SQLite is a file-based embedding database with some course data pre-loaded. Thus you don't need to use Model APIs or admin site to populate data by yourself and can focus on creating views and templates.

Next activate the models for an `onlinecourse` app.

- You need to perform migrations for first-time running to create necessary tables

```
1
```

```
python3 manage.py makemigrations
```

Copied! Executed!

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

```
1
```

```
python3 manage.py migrate
```

Copied! Executed!

Create a Popular Course List View with Template

Now we can start to write views to read data from database using methods in models and add data to templates to render dynamic HTML pages.

In this step, we will create a popular course list view as the index or main page of the `onlinecourse` app.

First let's create a course list template.

- Open `onlinecourse/templates/onlinecourse/course_list.html` and copy the following code snippet under `<!-- Add your template there -->`

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

```
{% if course_list %}
    <ul>
    {% for course in course_list %}
        <div>
            <img src="{{ MEDIA_URL }}/{{ course.image }}" width="360" height="360" >
            <h1><b>{{ course.name }}</b></h1>
        </div>
    {% endfor %}
    </ul>
{% else %}
<p>No courses are available.</p>
{% endif %}
```

Copied!

In above code snippet, we first add a `{% if course_list %}` if tag to check if the `course_list`exists in the context sent by the index view.

If it exists, we then add a `{% for course in course_list %}` to iterate the course list and display the fields of `course` such as `image`, `name`, etc.

Next, let's create a view to provide a course list to the template via `Course` model.

- Open `onlinecourse/views.py`, add a view to get top-10 popular courses from models.
  The courses objects are ordered by `total_enrollment` in desc order and the first ten objects are sliced as the top-10 courses,

```
Course.objects.order_by('-total_enrollment')[:10]
```

- Then we create a `context` dictionary object and append `course_list` into `context`.

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

```
def popular_course_list(request):
    context = {}
    # If the request method is GET
    if request.method == 'GET':
        # Using the objects model manage to read all course list
        # and sort them by total_enrollment descending
        course_list = Course.objects.order_by('-total_enrollment')[:10]
        # Appen the course list as an entry of context dict
        context['course_list'] = course_list
        return render(request, 'onlinecourse/course_list.html', context)
```

Copied!

Next, add a route path for the `popular_course_list` view.

- Open `onlinecourse/urls.py`, add a path entry in `urlpatterns`:

```
1
```

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

Copied!

and your `urlpatterns` list should look like the following:

```
1
2
3
4
5
6
```

```

urlpatterns = [
    # Add path here
    path(route='', view=views.popular_course_list, name='popular_course_list'),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)\
 + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Copied!

Now we have created a view and a template to display top-10 popular courses.

Let's test the view and template.

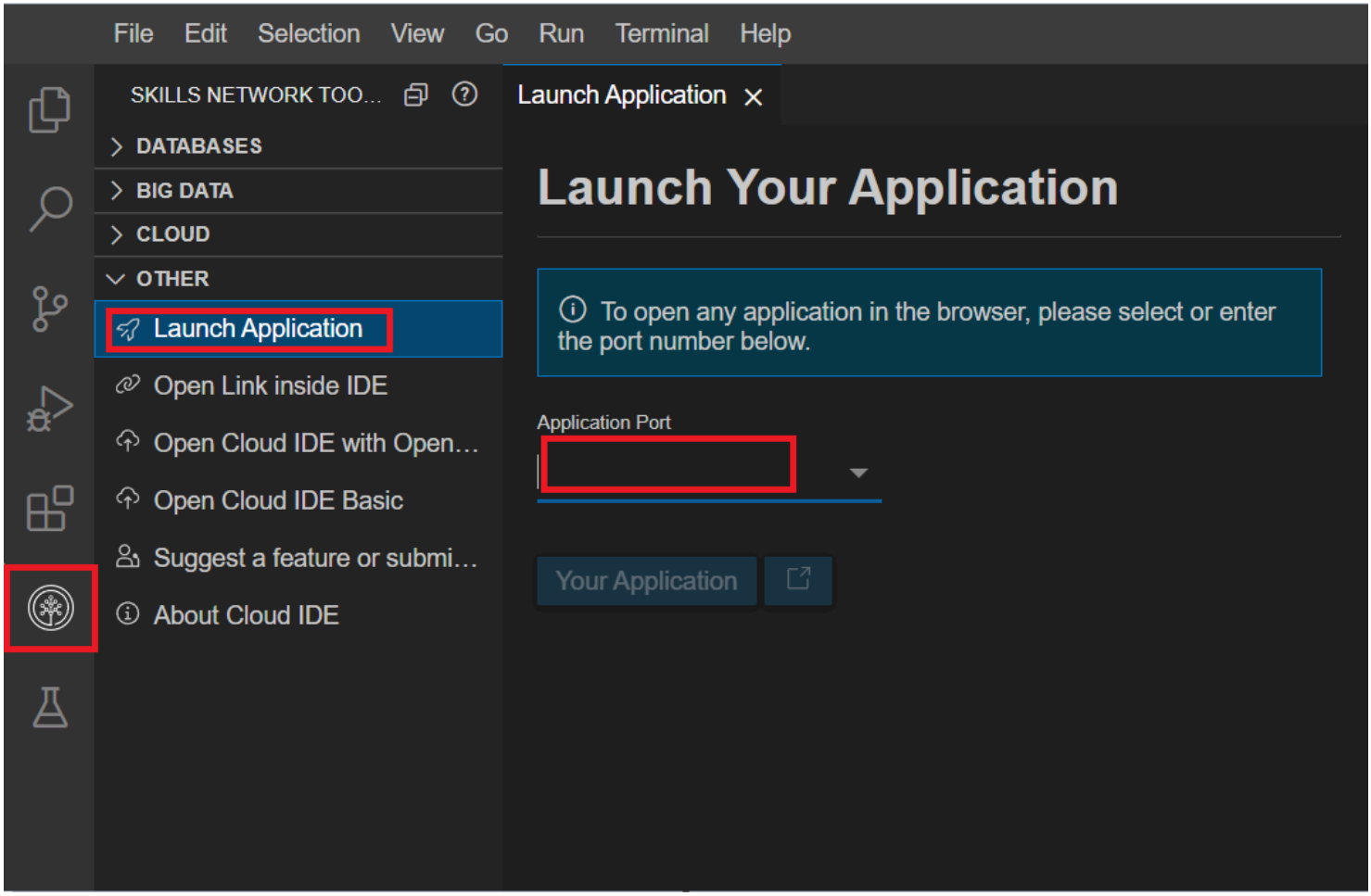
- Start the server

```
1
```

```
python3 manage.py runserver
```

Copied!

- 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 a new 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`

- And you should see a course list with two courses `Introduction to Django` and `Introduction to Python`.

The template we just created is a plain HTML template without any styling or formatting. We will improve its looking in later steps.

Coding Practice: Add More Fields to Course

Complete the following template `course_list.html` to add `total_enrollment` and `description` fields to a course on the list.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
{% if course_list %}
    <ul>
    {% for course in course_list %}
        <div>
            <img src="{{ MEDIA_URL }}/{{ course.image }}" width="360" height="360" >
            <h1><b>{{ course.name }}</b></h1>
            <p><!--<HINT> add a `total_enrollment` variable here --> enrolled</p>
            <p><!--<HINT> add a `description` variable here --></p>
        </div>
    {% endfor %}
    </ul>
{% else %}
<p>No courses are available.</p>
{% endif %}
```

Copied!

▶ Click here to see solution

Create an Course Enrollment View

Next let's create a simplified user enrollment process by creating a view to update the `total_enrollment` field of a course.

- Update `course_list.html` to add a form for submitting an enrollment request.

It sends a `POST` request to a URL parsed from `{% url 'onlinecourse:enroll' course.id %}` tag mapping to a update view.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
{% if course_list %}
    <ul>
    {% for course in course_list %}
        <div>
            <img src="{{ MEDIA_URL }}/{{ course.image }}" width="360" height="360" >
            <h1><b>{{ course.name }}</b></h1>
            <p>{{course.total_enrollment}} enrolled</p>
            <p>{{ course.description }}</p>
            <form action="{% url 'onlinecourse:enroll' course.id %}" method="post">
                {% csrf_token %}
                <input type="submit" value="Enroll">
            </form>
        </div>
    {% endfor %}
    </ul>
{% else %}
<p>No courses are available.</p>
{% endif %}
```

Copied!

Create an `enroll` view to add one to the `total_enrollment` field

- Open `onlinecourse/views.py`, add an `enroll` view:

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

```
def enroll(request, course_id):
    # If request method is POST
    if request.method == 'POST':
        # First try to read the course object
        # If could be found, raise a 404 exception
        course = get_object_or_404(Course, pk=course_id)
        # Increase the enrollment by 1
        course.total_enrollment += 1
        course.save()
        # Return a HTTP response redirecting user to course list view
        return HttpResponseRedirect(reverse(viewname='onlinecourse:popular_course_list'))
```

Copied!

1. The `enroll` view first reads a course object based on the `course_id` argument.
2. If the course doesn't exist in database, it returns a HTTP response with status code `404 Not Found` error.
3. Then it simply increases the total enrollment by one and updates the object.
4. You should always return an `HttpResponseRedirect` after successfully dealing with POST data.

5. `HttpResponseRedirect` takes a URL argument to which the user will be redirected.
6. Here we are using the `reverse()` function in the `HttpResponseRedirect` to generate the URL for `popular_course_list` view (e.g., `https://userid-8000.theiadocker-1.proxy.cognitiveclass.ai/onlinecourse/`).

For now, we just simply redirect user to the same page with `viewname='onlinecourse:popular_course_list'`.

- Open `onlinecourse/urls.py` add a `path('course/<int:course_id>/enroll/', views.enroll, name='enroll'),` to create a route to `enroll` view

```
1
```

```
path('course/<int:course_id>/enroll/', views.enroll, name='enroll'),
```

Copied!

- Save all updated files and refresh the course list page, and you should see a `Enroll` button under each course in the course list page. When you click the `Enroll` button, the count will be increased by 1.

Create a Course Details View with Template

In the previous step, a user will be redirected to the course list page after enrolled a course. This is not a practical scenario because users should be redirected into the details of a course, so that they can read the learning materials and take exams.

To make the user scenario more practical, we will be creating a course detail page and redirecting users to the detail page after enrollment.

Let's start with creating a course detail template.

- Open `onlinecourse/templates/onlinecourse/course_detail.html`, adding the following code snippet

```
1
2
3
4
```

```
 <div>
      <h2>{{ course.name }}</h2>
      <h5>{{ course.description }}</h5>
 </div>
```

Copied!

The above code snippet adds two variables `{{course.name}}` and `{{course.description}}`. They are wrapped in two HTML titles to represent the details of a course given by a course id.

Next, let's add a view to read the requested course and send context to `course_detail.html` template.

- Open `onlinecourse/views.py`, add a `course_details` view

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

```
def course_details(request, course_id):
    context = {}
    if request.method == 'GET':
        try:
            course = Course.objects.get(pk=course_id)
            context['course'] = course
            # Use render() method to generate HTML page by combining
            # template and context
            return render(request, 'onlinecourse/course_detail.html', context)
        except Course.DoesNotExist:
            # If course does not exist, throw a Http404 error
            raise Http404("No course matches the given id.")
```

Copied!

The above code snippet first try to get the specific course given by course id and add it to the context to be used for rendering HTML page.

If the course can not be found, a `Http404` exception will be raised.

Next, we can configure the route for the `course_details` page.

- Open `onlinecourse/urls.py`, add path entry `path('course/<int:course_id>/', views.course_details, name='course_details'),`
- Next, go back to `onlinecourse/views.py` and update the `enroll` view function to redirect user to the web page generated by `course_details` view.

```
1
```

```
return HttpResponseRedirect(reverse(viewname='onlinecourse:course_details', args=(course.id,)))
```

Copied!

Note that we added a `course.id` in the URL as an argument so the redirecting URL will look like this:

```
1
```

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

Copied!

We have the view and template created for course details. Let's test them.

- You can refresh the course list page and click the `enroll`, then the app will bring you the course details page generated by the view and template we just created.

Coding Practice: Add Lessons to Course Details Page

Each course has One-To-Many relationships to `Lesson`. In the pre-load database, we already created several lessons for a course (feel free to use admin site to add more lessons).

- Complete and add the following HTML code snippet (under `{{course.description }}`) to `course_details.html` to show the lessons on course details page.

```
1
2
3
4
5
6
7
```

```
    <h2>Lessons: </h2>
    {% for lesson in course.lesson_set.all %}
        <div>
            <h5>Lesson <!--<HINT>add `order` variable  --> : <!--<HINT>add `title` variable  --></h5>
            <p><!--<HINT>add `content` variable  --></p>
        </div>
    {% endfor %}
```

Copied!

▶ Click here to see solution

Add CSS to Templates

The course list and course details pages we created were looking very primitive because they are pure HTML pages without any CSS styling.

Let's try to stylize the templates by adding some CSS style classes to the HTML elements in the templates.

We have provided a sample css file called `course.css`, you just need to include it into your templates and use the style classes.

- Open `course_list.html`, include the link of `course.css` static file to `<head>` element:

```
1
2
```

```
{% load static %}
<link rel="stylesheet" type="text/css" href="{% static 'onlinecourse/course.css' %}">
```

Copied!

- Then, stylize the course list `<div>` (under `{% for course in course_list %}` tag) in a `.container` class which adds some paddings.

```
1
2
3
```

```
<div class="container">
...
</div>
```

Copied!

- Try to add a `<div class="row">` to wrap up course fields as a table row.

- Use a `<div class="column-33">` to wrap up and divide the course image as a column takes up `33%` width
- Use a `<div class="column-66">` to wrap up `name`, `total_enrollment`, `description`, and enroll form

Feel free to add more styles such as making the enrollment numbers green

- The sytlized course list should look like the following:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
{% if course_list %}
   <ul>
   {% for course in course_list %}
       <div class="container">
          <div class="row">
            <div class="column-33">
                <img src="{{MEDIA_URL}}/{{ course.image }}" width="360" height="360" >
            </div>
            <div class="column-66">
                <h1 class="xlarge-font"><b>{{ course.name }}</b></h1>
                <p style="color:MediumSeaGreen;"><b>{{course.total_enrollment}} enrolled</b></p>
                <p> {{ course.description }}</p>
                <form action="{% url 'onlinecourse:enroll' course.id %}" method="post">
                {% csrf_token %}
                <input class="button"  type="submit"  value="Enroll">
              </form>
            </div>
          </div>
       </div>
       <hr>
   {% endfor %}
   </ul>
{% else %}
<p>No courses are available.</p>
{% endif %}
```

Copied!

Refresh the course list page, and check the result. Your course list page should look like the following:

Coding Practice: Stylize Course Details Page

Playing with other CSS classes in the `onlinecourse/course.css` file, such as `.card` class to make a lesson looking like a card.

- Update `course_detail.html` and add `.card` class to the `<div>` element wrapping up course name, description and each lesson in the course:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
18. 18
19. 19
20. 20
21. 21
```

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.     <meta charset="UTF-8">
5.     {% load static %}
6.     <link rel="stylesheet" type="text/css" href="{% static 'onlinecourse/course.css' %}">
7. </head>
8. <body>
9.     <div <!--<HINT> add `.card` class here -->>
10.         <h2>{{ course.name }}</h2>
11.         <h5>{{ course.description }}</h5>
12.     </div>
13.     <h2>Lessons: </h2>
14.     {% for lesson in course.lesson_set.all %}
15.     <div <!--<HINT> add `.card` class here -->>
16.         <h5>Lesson {{lesson.order}} : {{lesson.title}}</h5>
17.         <p>{{lesson.content}}</p>
18.     </div>
19.     {% endfor %}
20. </body>
21. </html>
```

Copied!

▶ Click here to see solution

Summary

In this lab, you have learned how to create views and templates to show a list of courses and course details.
You also learned how to use views to update object and redirect users to other pages. At last, you have learned how to use CSS to stylize your templates.

Author(s)

Yan Luo

Changelog

Date	Version	Changed by	Change Description
30-Nov-2020	1.0	Yan Luo	Initial version created
04-Jul-2023	1.1	K Sundararajan	Screenshots and @IBM…2023 updated
12-Jul-2023	1.2	K Sundararajan	Updated instructions based on SME review
26-Jul-2023	1.3	K Sundararajan	Updates based on QA review