Final Project: Add a New Assessment Feature to an Online Course Application

Estimated time needed: 90 mins

As a newly onboarded full-stack developer, your lead developer has entrusted you with implementing a new course assessment feature. To successfully deliver this feature, you will use your Django full-stack skills to design and develop the necessary models, templates, and views. Finally, you will run and thoroughly test your online course application to ensure its functionality.

Objectives

By the end of this lab you will be able to:

1. Understand the requirements of the new course assessment feature
2. Create question, choice, and submission models
3. Create a new course object with exam related models using the admin site
4. Update the course details template to show questions and choices
5. Create a new exam result template to show the result of the submission
6. Create a new exam result submission view
7. Create a new view to display and evaluate exam result

Working with Files in Cloud IDE

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

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

If you have cloned (using the `git clone` command) boilerplate/starting code, then it will look like the image below:

If you have not cloned and are starting with a blank project, it will look like this:

Create a New File

To create a new file in your project, right-click and select the New File option. You can also choose `File -> New File` to do the same.

You will then be prompted to name the new file. In this scenario, let's name it `sample.html`.

Clicking the file name `sample.html` in the directory structure will open the file on the right pane. You can create all different types of files; for example, `FILE_NAME.js` for JavaScript files.

In the example below, we pasted some basic HTML code and then saved the file.

We save this file by:

- Going in the menu
- Press `Command + S` on Mac or `CTRL + S` on Windows
- Alternatively, it will Autosave your work as well

Set-up: Create an Application

1. Open a terminal window using the editor's menu: Select **Terminal > New Terminal**.

2. If you are not currently in the project folder, copy and paste the following code to change to your project folder. Select the copy button to the right of the code to copy it.

   ```
   cd /home/project
   ```
   Copied! Executed!

3. Run the following command to clone the Git repository that contains the starter code needed for this project, if the Git repository doesn't already exist.

   ```
   [ ! -d 'tfjzl-final-cloud-app-with-database' ] && git clone https://github.com/ibm-developer-skills-network/tfjzl-final-cloud-app-with-database
   ```
   Copied! Executed!

4. Change to the directory **tfjzl-final-cloud-app-with-database** to begin working on the lab.

   ```
   cd tfjzl-final-cloud-app-with-database
   ```
   Copied! Executed!

5. List the contents of this directory to see the artifacts for this lab.

```
ls
```

Copied! Executed!

6. Let us set up a virtual environment to contain all the packages we need.

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

Copied! Executed!

7. Set up the Python runtime and test the template project.

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

Copied! Executed!

8. Create the initial migrations and generate the database schema:

   Migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema. They are designed to be mostly automatic, but you will need to know when to make migrations, when to run them, and the common problems you might run into. There are several commands which you will use to interact with migrations and Django's handling of database schema:

   1. ***migrate***, which is responsible for applying and unapplying migrations
   2. ***makemigrations***, which is responsible for creating new migrations based on the changes you have made to your models
   3. ***sqlmigrate***, which displays the SQL statements for a migration
   4. ***showmigrations***, which lists a project's migrations and their status

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Copied! Executed!

9. Run server successfully this time.

```
python3 manage.py runserver
```

Copied! Executed!

Launch Application

12. It will look like the image below:

13. In your terminal, press `CTRL+C` to stop your web server.

Working with a Git Repo

It is important to understand that the lab environment is temporary. It only lives for a short while before it is destroyed. It would be best if you pushed all changes made to your own GitHub repository to recreate it in a new lab environment when required.

Also, note that this environment is shared and, therefore, not secure. You should not store personal information, usernames, passwords, or access tokens in this environment for any purpose.

To protect yourself from re-work, you must occasionally commit and push your code to a GitHub repository.

Review changes

To review the changes that have been made, run the following commands in the terminal:

```
cd [your repo name]
git status
```

Copied! Executed!

Mark changes for commit

You now need to commit the changes you've made. Before you can do that, you need to add the new and revised files to the commit:

```
git add sample.html
git add existing_file.html
```

Copied! Executed!

After adding the files, rerun `git status`.

Git setup - Your identity

The first thing you should do when you start using Git is to set your user name and email address. This is important because every Git commit uses this information, and it will be part of the commits you start creating. Replace the given Username and Email ID with your personal credentials:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Copied! Executed!

Commit the changes

You are now able to commit the changes you've made. Run the following command to commit the changes. You will pass a commit message using the `-m` option.

```
git commit -m 'changes made from the lab environment'
```

Copied! Executed!

Git Remote

1. Create a free account on GitHub
2. Verify your email address, if you haven't done so already
3. Go to Settings: In the upper-right corner of any page, click your profile photo, then click Settings
4. Go to Developer Settings
5. Create personal access token to authenticate to GitHub from your environment
6. Set privileges for the token and generate
7. Create a remote repository to push your code
8. When you clone a repository with git clone, it automatically creates a remote connection called origin, which points back to the cloned repository. This is useful for developers creating a local copy of a central repository since it provides an easy way to pull upstream changes or publish local commits.

   If the remote origin is already set (most likely to happen when you clone from GitHub).
   Replace `YOUR_GITHUB_USER` in the following commands with your actual GitHub username.

   ```
   git remote set-url origin https://github.com/YOUR_GITHUB_USER/my-course-repo.git
   ```

   Copied! Executed!

   Creating remote origin the first time (when you start with a blank repository locally).

   ```
   git remote add origin https://github.com/YOUR_GITHUB_USER/my-course-repo.git
   ```

   Copied! Executed!

9. You will be presented with multiple options for adding code to this repository. In your lab environments, you will be provided boilerplate code, so the best option is to **push an existing repository from the command line**.

   While running the below commands, you will be prompted to enter a `username`. This will be your GitHub username. And the `password` will be the access token that you generated earlier.

   Then:

   ```
   git branch -M main
   git push -u origin main
   ```

   Copied! Executed!

Task 1: Build New Models

You will need to create several new models in `onlinecourse/models.py`

Open **models.py** in IDE

Question model

A `Question` model will save the questions of an exam with the following characteristics:

- Used to persist questions for a course
- Has a Many-To-One relationship with the course
- Has question text
- Has a grade point for each question

▼ Hint

```
class Question(models.Model):
    Foreign key to course
    Question text
    Question grade
```

Copied!

▼ Solution

```
class Question(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    content = models.CharField(max_length=200)
    grade = models.IntegerField(default=50)

    def __str__(self):
        return "Question: " + self.content
```

Copied!

Get Score

Additionally, you can add the following function to your Question model, which calculates the score:

```
    # method to calculate if the learner gets the score of the question
    def is_get_score(self, selected_ids):
        all_answers = self.choice_set.filter(is_correct=True).count()
        selected_correct = self.choice_set.filter(is_correct=True, id__in=selected_ids).count()
        if all_answers == selected_correct:
            return True
        else:
            return False
```

Copied!

Choice model

A `Choice` model saves all of the choices of a question:

- One-To-One relationship with `Question` model
- The choice text
- Indicates if this choice is the correct one or not

▶ Hint

▼ Solution

```
class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    content = models.CharField(max_length=200)
    is_correct = models.BooleanField(default=False)
```

Copied!

Submission Model

You are provided with commented out `Submission` model, which has:

- One-to-Many relationships with Enrollment, for example, one course enrollment could have multiple exam submissions
- Many-to-Many relationship with choices or questions. For simplicity, you could relate the submission with the Choice model

You need to uncomment the `Submission` model and use it to associate selected choices.

Refer to other models in `models.py` as examples.

Here is an example ER Diagram for your reference:

Final solution

Additionally, you can look at the final solution provided below.

▼ Click here to see final onlinecourse/models.py

```
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
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
```

```
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
```

```
import sys
from django.utils.timezone import now
try:
    from django.db import models
except Exception:
    print("There was an error loading django modules. Do you have django installed?")
    sys.exit()

from django.conf import settings
import uuid


# Instructor model
class Instructor(models.Model):
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
    )
    full_time = models.BooleanField(default=True)
    total_learners = models.IntegerField()

    def __str__(self):
        return self.user.username


# Learner model
class Learner(models.Model):
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
    )
    STUDENT = 'student'
    DEVELOPER = 'developer'
    DATA_SCIENTIST = 'data_scientist'
    DATABASE_ADMIN = 'dba'
    OCCUPATION_CHOICES = [
        (STUDENT, 'Student'),
        (DEVELOPER, 'Developer'),
        (DATA_SCIENTIST, 'Data Scientist'),
        (DATABASE_ADMIN, 'Database Admin')
    ]
    occupation = models.CharField(
        null=False,
        max_length=20,
        choices=OCCUPATION_CHOICES,
        default=STUDENT
    )
    social_link = models.URLField(max_length=200)

    def __str__(self):
        return self.user.username + "," + \
               self.occupation


# Course model
class Course(models.Model):
    name = models.CharField(null=False, max_length=30, default='online course')
    image = models.ImageField(upload_to='course_images/')
    description = models.CharField(max_length=1000)
    pub_date = models.DateField(null=True)
    instructors = models.ManyToManyField(Instructor)
    users = models.ManyToManyField(settings.AUTH_USER_MODEL, through='Enrollment')
    total_enrollment = models.IntegerField(default=0)
    is_enrolled = False

    def __str__(self):
        return "Name: " + self.name + "," + \
               "Description: " + self.description


# Lesson model
class Lesson(models.Model):
    title = models.CharField(max_length=200, default="title")
    order = models.IntegerField(default=0)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    content = models.TextField()


# Enrollment model
# <HINT> Once a user enrolled a class, an enrollment entry should be created between the user and course
# And we could use the enrollment to track information such as exam submissions
class Enrollment(models.Model):
    AUDIT = 'audit'
    HONOR = 'honor'
    BETA = 'BETA'
    COURSE_MODES = [
        (AUDIT, 'Audit'),
        (HONOR, 'Honor'),
        (BETA, 'BETA')
    ]
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    date_enrolled = models.DateField(default=now)
    mode = models.CharField(max_length=5, choices=COURSE_MODES, default=AUDIT)
    rating = models.FloatField(default=5.0)


class Question(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    content = models.CharField(max_length=200)
    grade = models.IntegerField(default=50)

    def __str__(self):
        return "Question: " + self.content

    def is_get_score(self, selected_ids):
        all_answers = self.choice_set.filter(is_correct=True).count()
        selected_correct = self.choice_set.filter(is_correct=True, id__in=selected_ids).count()
        if all_answers == selected_correct:
            return True
        else:
            return False

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    content = models.CharField(max_length=200)
    is_correct = models.BooleanField(default=False)

class Submission(models.Model):
    enrollment = models.ForeignKey(Enrollment, on_delete=models.CASCADE)
    choices = models.ManyToManyField(Choice)
```

Copied!

Run migrations

```
python3 manage.py makemigrations onlinecourse
python3 manage.py migrate
```

Copied! Executed!

Note: If you see any errors related to model migrations, you could delete the existing database `db.sqlite3` and rerun the above migration again.

Assessment: Take a screen capture of the Question, Choice, and Submission model below and save the screen capture as `01-models.png`.

Task 2: Register Model Changes

You will now make changes to `onlinecourse/admin.py` to be able to use the new features you have built.

Open **admin.py** in IDE

Import new models

At the moment, you are only importing `Course`, `Lesson`, `Instructor`, and `Learner` in `onlinecourse/admin.py`

You need to add `Question`, `Choice`, and `Submission`

▶ Solution

Create QuestionInline and ChoiceInline

Create `QuestionInline` and `ChoiceInline` classes so that you could edit them together on one page in the admin site.

▶ Hint

▼ Solution

```
class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 2

class QuestionInline(admin.StackedInline):
    model = Question
    extra = 2
```

Copied!

Create QuestionAdmin class

▶ Hint

▼ Solution

```
class QuestionAdmin(admin.ModelAdmin):
    inlines = [ChoiceInline]
    list_display = ['content']
```

Copied!

Register Question, Choice, and Submission

After you register the new models, you could create a new course with lessons, questions, and question choices using the admin site.

The register decorator: `register(*models, site=django.contrib.admin.sites.site)`

▶ Hint

▼ Solution

```
admin.site.register(Question, QuestionAdmin)
admin.site.register(Choice)
admin.site.register(Submission)
```

Copied!

See the final `admin.py` here:

▼ Solution

```
from django.contrib import admin
# <HINT> Import any new Models here
from .models import Course, Lesson, Instructor, Learner, Question, Choice, Submission

# <HINT> Register QuestionInline and ChoiceInline classes here


class LessonInline(admin.StackedInline):
    model = Lesson
    extra = 5

class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 2

class QuestionInline(admin.StackedInline):
    model = Question
    extra = 2

# Register your models here.
class CourseAdmin(admin.ModelAdmin):
    inlines = [LessonInline]
    list_display = ('name', 'pub_date')
    list_filter = ['pub_date']
    search_fields = ['name', 'description']

class QuestionAdmin(admin.ModelAdmin):
    inlines = [ChoiceInline]
    list_display = ['content']

class LessonAdmin(admin.ModelAdmin):
    list_display = ['title']


# <HINT> Register Question and Choice models here

admin.site.register(Course, CourseAdmin)
admin.site.register(Lesson, LessonAdmin)
admin.site.register(Instructor)
admin.site.register(Learner)
admin.site.register(Question, QuestionAdmin)
admin.site.register(Choice)
admin.site.register(Submission)
```

Copied!

Assessment: Take a screen capture of the `admin.py` and save the screen capture as `02-admin-file.png`.

Create an admin user

Let's create an admin user with the following details:

1. Username: `admin`
2. Email address: *leave blank by pressing enter*
3. Password: Your choice, or use `p@ssword123`

```
python3 manage.py createsuperuser
```

Copied! Executed!

Save your changes

Run the Django development server and check if you can add Question and Choice objects using the admin site.

```
python3 manage.py runserver
```

Copied! Executed!

Launch Django admin

Assessment: Take a screen capture of the admin site and save the screen capture as `03-admin-site.png`.

Task 3: Update the Course Detail Template

You will now update the course detail template to create an exam section with a list of questions and choices.

One exam contains multiple questions, and each should have more than one correct answer (multiple-selection).

The changes will be made in `templates/onlinecourse/course_details_bootstrap.html`

Open **course_detail_bootstrap.html** in IDE

Start editing the code in the placeholder provided:

1. If the user is authenticated, show the course exam with a list of questions and choices:

   ▼ Hint

   ```
    {% if CONDITION %}
    </br>
    <!-- Remaining code will go here -->
    {% endif %}
   ```

   Copied!

   ▼ Solution

   ```
    {% if user.is_authenticated %}
    </br>
    <!-- Remaining code will go here -->
    {% endif %}
   ```

   Copied!

2. Add a button to start the exam:

   ▶ Hint
   ▶ Solution

3. Add a collapsable `div`:

   ▶ Hint
   ▶ Solution

4. Add the Question logic inside a `form`:

   ▶ Hint
   ▶ Solution

5. Add Question UI:

   ▶ Hint
   ▶ Solution

6. Add Choices components:

   ▶ Hint
   ▶ Solution

Final solution

View the final solution here:

▼ Solution

```
{% if user.is_authenticated %}
</br>
<button class="btn btn-primary btn-block" data-toggle="collapse" data-target="#exam">Start Exam</button>
<div id="exam" class="collapse">
    <form id="questionform" action="{% url 'onlinecourse:submit' course.id %}" method="POST">
        {% for question in course.question_set.all %}
        <div class="card mt-1">
            <div class="card-header">
                <h5>{{ question.content }}</h5>
            </div>
            {% csrf_token %}
            <div class="form-group">
                {% for choice in question.choice_set.all %}
                <div class="form-check">
                    <label class="form-check-label">
                        <input type="checkbox" name="choice_{{choice.id}}" class="form-check-input"
                            id="{{choice.id}}" value="{{choice.id}}">{{ choice.content }}
                    </label>
                </div>
                {% endfor %}
            </div>
        </div>
        {% endfor %}
        <input class="btn btn-success btn-block" type="submit" value="Submit">
    </form>
</div>
{% endif %}
```

Copied!

Run in to test:

```
python3 manage.py runserver
```

Copied! Executed!

Launch onlinecourse application

At this moment, you can not submit the exam. You will be implementing that in the next lab.

Commit your code

Committing and pushing your code to GitHub is a good practice to avoid losing it.

Assessment: Take a screen capture of `course_details_bootstrap.html` and save the screen capture as `04-course-details.png`.

Task 4: Test Data

You will now create test data for your application.

Add instructor

Add `admin` as an Instructor

Course information

Field	Value
Name	Learning Django
Image	Download from here
Description	Django is an extremely popular and fully featured server-side web framework, written in Python
Pub date	Today
Instructors	admin
Lesson #1 Title	What is Django
Lesson #1 Order	0
Lesson #1 Content	Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. Itੳurce.

Test question

Field	Value
Course	Name: Learning Django, Description: …
Content	Is Django a Python framework
Grade	100
Choice #1 Content	Yes
Choice #1 Is correct	
Choice #2 Content	No
Choice #2 Is correct	*Leave blank*

Let's open the course's front end.

Launch Application

Task 5: Submission Evaluation

Since you have created several new models, you now need to import them at the top of the `views.py`

Open **views.py** in IDE

```
from .models import Course, Enrollment, Question, Choice, Submission
```

Copied!

Submit view

You will now create a function-based view for form submission.

Create a submit view `def submit(request, course_id):` to create an exam submission record for a course enrollment. You may implement it based on the following logic:

- Get the current user and the course object, then get the associated enrollment object
  (HINT: `Enrollment.objects.get(user=..., course=...)`)
- Create a new submission object referring to the enrollment
  (HINT: `Submission.objects.create(enrollment=...)`)
- Collect the selected choices from the HTTP request object (HINT: you could use `request.POST` to get the payload dictionary and the choice id from the dictionary values. An example code snippet is also provided.)
- Add each selected choice object to the submission object
- Redirect to a `show_exam_result` view with the submission id to show the exam result
- Configure `urls.py` to route the new `submit` view such as `path('<int:course_id>/submit/', ...),`

Form submission in `views.py`

▶ Hint

▼ Solution

```
def submit(request, course_id):
    course = get_object_or_404(Course, pk=course_id)
    user = request.user
    enrollment = Enrollment.objects.get(user=user, course=course)
    submission = Submission.objects.create(enrollment=enrollment)
    choices = extract_answers(request)
    submission.choices.set(choices)
    submission_id = submission.id
    return HttpResponseRedirect(reverse(viewname='onlinecourse:exam_result', args=(course_id, submission_id,)))
```

Copied!

Route the submit view button in `urls.py`

Open **urls.py** in IDE

▶ Hint

▼ Solution

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

Copied!

Evaluation view

Create an exam result view `def show_exam_result(request, course_id, submission_id):` to check if the learner passed the exam and their question results.

You may implement the view based on the following logic:

- Get the course object and submission object based on their ids in view arguments
- Get the selected choice ids from the submission record
- For each selected choice, check if it is a correct answer or not
- Calculate the total score by adding up the grades for all questions in the course
- Add the course, selected_ids, and grade to context for rendering HTML page
- Configure `urls.py` to route the new `show_exam_result` view such as `path('course/<int:course_id>/submission/<int:submission_id>/result/', ...),`

Exam results in `views.py`

▶ Hint

▼ Solution

```
def show_exam_result(request, course_id, submission_id):
    context = {}
    course = get_object_or_404(Course, pk=course_id)
    submission = Submission.objects.get(id=submission_id)
    choices = submission.choices.all()
    total_score = 0
    for choice in choices:
        if choice.is_correct:
            total_score += choice.question.grade
    context['course'] = course
    context['grade'] = total_score
    context['choices'] = choices

    return render(request, 'onlinecourse/exam_result_bootstrap.html', context)
```

Copied!

Exam results in `urls.py`

Open **urls.py** in IDE

▶ Hint

▼ Solution

```
path('course/<int:course_id>/submission/<int:submission_id>/result/', views.show_exam_result, name="exam_result"),
```

Copied!

Assessment: Take a screen capture of `views.py` and save the screen capture as `05-views.png`.

Assessment: Take a screen capture of `urls.py` and save the screen capture as `06-urls.png`.

Task 6: Complete the Exam Result Template to Show Exam Submission Results

Complete the HTML template `exam_result_bootstrap.html` for submission results which should show if a learner passed the exam with details like the total score, the result for each question, and so on. Check the previous UI design for more details.

Stylize the updated template with `Bootstrap` to meet the UI design from the design team.

Pass output

Learners who pass the exam should be shown the final score and a congratulations message.

▶ Hint
▼ Solution

```
<b>Congratulations, {{ user.first_name }}!</b> You have passed the exam and completed the course with score {{ grade }}/100
```

Copied!

Fail output

Learners who fail the exam should be shown the final score with incorrect choices. The learner should be allowed to retake the exam and resubmit it.

▶ Hint
▼ Solution

```
<b>Failed</b> Sorry, {{ user.first_name }}! You have failed the exam with score {{ grade }}/100
```

Copied!

Exam result

You must also display the exam results so the learner can see how they did.

▶ Hint
▼ Solution

```
{% for question in course.question_set.all %}
<div class="card mt-1">
    <div class="card-header"><h5>{{ question.content }}</h5></div>
    <div class="form-group">
        {% for choice in question.choice_set.all %}
        <div class="form-check">
            {% if choice.is_correct and choice in choices %}
            <div class="text-success">Correct answer: {{ choice.content }}</div>
            {% else %}{% if choice.is_correct and not choice in choices %}
            <div class="text-warning">Not selected: {{ choice.content }}</div>
            {% else %}{% if not choice.is_correct and choice in choices %}
            <div class="text-danger">Wrong answer: {{ choice.content }}</div>
            {% else %}
            <div>{{ choice.content }}</div>
            {% endif %}{% endif %}{% endif %}
        </div>
        {% endfor %}
    </div>
</div>
{% endfor %}
```

Copied!

Assessment: Take a screen capture of a result from the mock exam and save the screen capture as `07-final.png`.

Summary

In this lab, you have gained an understanding of the requirements to obtain and test a code template for the course assessment feature. You have efficiently implemented these features, ensuring requirements alignment while maintaining good project organization and documentation.

Author(s)

- Yan Luo
- Muhammad Yahya