

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

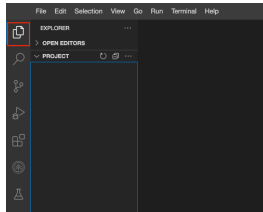
By the end of this lab you will be able to:

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

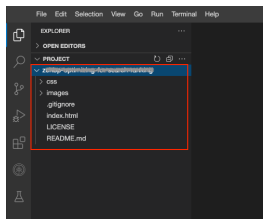
Working with Files in Cloud IDE

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.

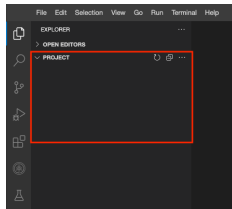
To view your files and directories inside Cloud IDE, click the file icon to reveal it.



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

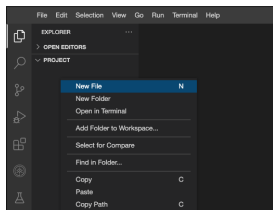


If you have not cloned and are starting with a blank project, it will look like this:

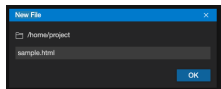


Create a New File

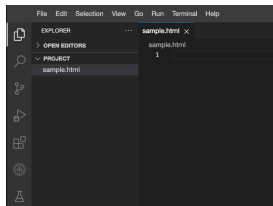
To create a new file in your project, right-click and select the New File option. You can also choose `File -> New File` to do the same.



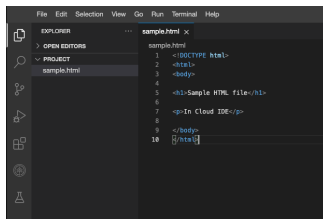
You will then be prompted to name the new file. In this scenario, let's name it `sample.html`.



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.

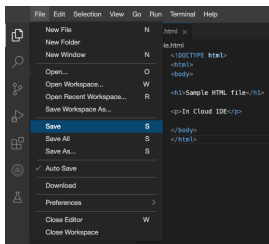


In the example below, we pasted some basic HTML code and then saved the file.



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



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
```

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
```

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
```

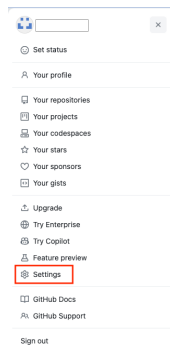
Commit the changes

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.

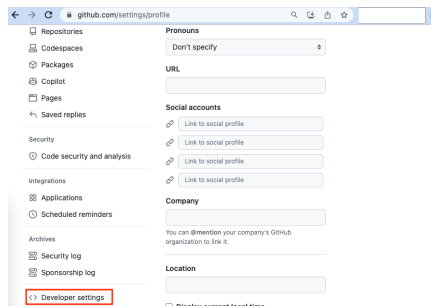
```
git commit -m 'changes made from the lab environment'
```

Git Remote

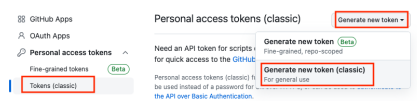
1. Create a free account on [Github](#)
2. Verify your email address, if you haven't done so already
3. Go to Settings: In the upper-right corner of any page, click your profile photo, then click Settings



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

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

cloud ide token

What's this token for?

Expiration *

30 days The token will expire on Wed, Aug 23 2023

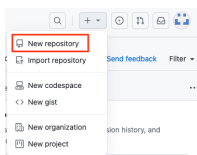
Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo:deployment	Access deployment status
<input type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<hr/>	
<input type="checkbox"/> admin:pkg_key	Full control of public user GPG keys
<input type="checkbox"/> write:pkg_key	Write public user GPG keys
<input type="checkbox"/> read:pkg_key	Read public user GPG keys
<hr/>	
<input type="checkbox"/> admin:ssh_signing_key	Full control of public user SSH signing keys
<input type="checkbox"/> write:ssh_signing_key	Write public user SSH signing keys
<input type="checkbox"/> read:ssh_signing_key	Read public user SSH signing keys

Generate token **Cancel**

7. Create a remote repository to push your code



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * **Repository name ***

my-course-repo

my-course-repo is available

Great repository names are short and memorable. Need inspiration? How about [super-duper-doodle](#) ?

Description (optional)

☐ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☒ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

☐ You are creating a private repository in your personal account.

Create repository

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_username` in the following commands with your actual GitHub username.

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

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

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

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**.

...or push an existing repository from the command line

```
git remote add origin https://github.com/_____/my-course-repo.git
git branch -M main
git push -u origin main
```

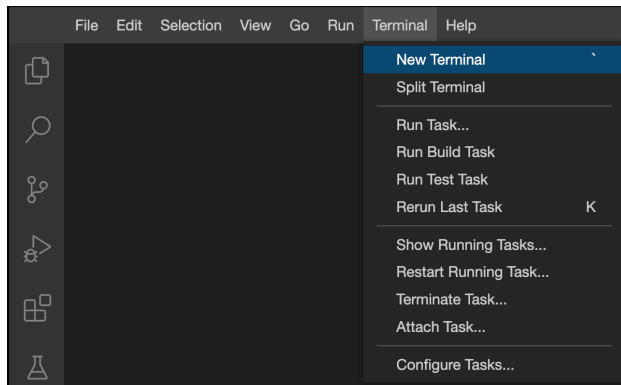
While running the below commands, you will be prompted to enter a `username`. This will be your GitHub username. And the `password` will be the access token that you generated earlier.

Then:

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

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
```

3. Fork the repository that contains the **starter code needed for this project**.

- Go to: <https://github.com/ibm-developer-skills-network/tfjz1-final-cloud-app-with-database>
- Click **Fork** and create a copy in your personal GitHub space.
- Ensure the repository remains **Public** and keep the same name: `tfjz1-final-cloud-app-with-database`

ibm-developer-skills-network / tfjz1-final-cloud-app-with-database

tfjz1-final-cloud-app-with-database Public

generated from [ibm-developer-skills-network/coding-project-template](#)

Watch 1 Fork 721 Star 2

main 1 Branch 0 Tags

lavskillup Update requirements.txt 185b623 · 2 years ago 3 Commits

myproject	initial commit	2 years ago
onlinecourse	initial commit	2 years ago
static	initial commit	2 years ago
.gitignore	Initial commit	2 years ago
LICENSE	Initial commit	2 years ago
Procfile	initial commit	2 years ago
README.md	initial commit	2 years ago
manage.py	initial commit	2 years ago
manifest.yml	initial commit	2 years ago
requirements.txt	Update requirements.txt	2 years ago
runtime.txt	initial commit	2 years ago

README Apache-2.0 license

About

final-cloud-app-with-database

- Readme
- Apache-2.0 license
- Activity
- Custom properties
- 2 stars
- 1 watching
- 721 forks

Report repository

Releases

No releases published

Packages

No packages published

Contributors 3

ibm-skills-network-bot

4. After forking, clone your **forked repository** to the lab environment. Replace `<your-username>` with your GitHub username:

```
git clone https://github.com/<your-username>/tfjz1-final-cloud-app-with-database.git
```

5. Change to the directory **tfjz1-final-cloud-app-with-database** to begin working on the lab.

```
cd tfjz1-final-cloud-app-with-database
```

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

```
ls
```

7. 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
```

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

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

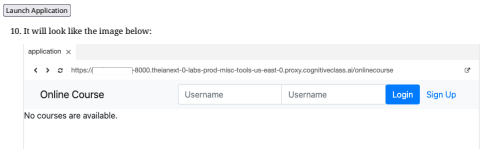
9. Create the initial migrations and generate the database schema:
Migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema. They are designed to be mostly automatic, but you will need to know when to make migrations, 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. **squigrate**, 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
```

9. Run server successfully this time.

```
python3 manage.py runserver
```



11. In your terminal, press `ctrl+c` to stop your web server.

Creating a Online Course Application

- Note:**
1. Make sure all updates to your Django project files (models.py, admin.py, views.py, urls.py, and course_details_bootstrap.html) are saved and committed to your GitHub repository.
 2. Take the required screenshots as mentioned in the lab instructions.
 3. **For Option 1: AI-Graded Submission and Evaluation:**
 - Submission requires both GitHub repository links and screenshots.
 4. **For Option 2: Peer-Graded Submission and Evaluation:**
 - Submission requires only screenshots.

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
```

• 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
```

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
```

Choice model

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

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

• Hint

```
class Choice(models.Model):  
    foreign key to question  
    Choice content as text  
    Is choice correct as boolean
```

• 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)
```

Submission Model

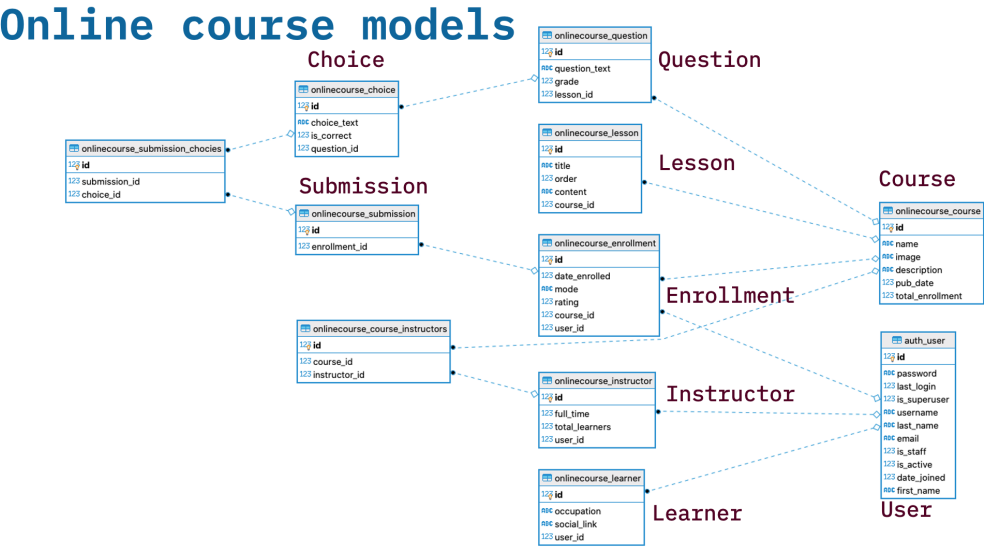
You are provided with commented out `Submission` model, which has:

- Many-to-One relationships with Exam Submissions, for example, multiple exam submissions could belong to one course enrollment.
- 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`

```
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(max_length=30, default='online course')
    image = models.ImageField(upload_to='course_images/')
    description = models.CharField(max_length=100)
    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 + ", " + 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()
    def __str__(self):
        return 'Question: ' + self.content
    def is_get_score(self, selected_id):
        all_answers = self.choice_set.filter(is_correct=True, count=1)
        selected_correct = self.choice_set.filter(is_correct=True, id__in=selected_id).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)
    submission = models.ForeignKey(Submission, on_delete=models.CASCADE)
    choices = models.ManyToManyField(Choice)
```

Run migrations

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

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

Commit your updated code to [Github repository](#)

Assessment

For Option 1: AI-Graded Submission and Evaluation

Copy and paste the [public Github repository URL](#) of the models.py file that contains the Question, Choice, and Submission models and save it in a text file.

For Option 2: Peer-Graded Submission and Evaluation

Take a the **screenshot** of your models.py file (showing the Question, Choice, and Submission models) and save it as 01-models.jpeg OR 01-models.png.

```
class Question(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    content = models.CharField(max_length=200)
    grade = models.IntegerField(default=100)

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

    def is_get_score(self, selected_id):
        all_answers = self.choice_set.filter(is_correct=True).count()
        selected_correct = self.choice_set.filter(is_correct=True, id__in=selected_id).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)
```

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

```
from models import Course, Lesson, Instructor, Learner, Question, Choice, Submission
```

Create QuestionInline and ChoiceInline

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

Hint

```
class Class_Name(admin.StackedInline):
    model = Model_Name
    extra = 2
```

Solution

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

Create QuestionAdmin class

Hint

```
class QuestionAdmin(admin.ModelAdmin):
    inlines = (QuestionSubContent)
    list_display = ('content')
```

Solution

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

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(model1, site=django.contrib.admin.site.site)

Hint

```
admin.site.register(Model1, Model2)
admin.site.register(Model3)
```

Solution

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

See the final admin.py here:

Solution

```
from django.contrib import admin
# <INFO> Import any new Models here
from models import Course, Lesson, Instructor, Learner, Question, Choice, Submission
# <INFO> 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')
# @DNT: Register Question and Choice models here
admin.site.register(Course, CourseAdmin)
admin.site.register(Lesson, LessonAdmin)
admin.site.register(Instructor, LessonAdmin)
admin.site.register(Learner, LessonAdmin)
admin.site.register(Question, QuestionAdmin)
admin.site.register(Choice, LessonAdmin)
admin.site.register(Submission)
```

Commit your updated code to Github repository

Assessment

For Option 1: AI-Graded Submission and Evaluation

Copy and save the **public Github repository URL** of your `admin.py` file that includes all seven imported classes and the implementations of `QuestionInline`, `ChoiceInline`, `QuestionAdmin`, and `LessonAdmin` and save it in a text file.

For Option 2: Peer-Graded Submission and Evaluation

Take a screenshot of your `admin.py` file and save it as `02-admin-file.jpeg` OR `02-admin-file.png`.

```
from django.contrib import admin
# @DNT: Import any new Models here
from models import Course, Lesson, Instructor, Learner, Question, Choice, Submission
# @DNT: 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')

# @DNT: Register Question and Choice models here
admin.site.register(Course, CourseAdmin)
admin.site.register(Lesson, LessonAdmin)
admin.site.register(Instructor, LessonAdmin)
admin.site.register(Learner, LessonAdmin)
admin.site.register(Question, QuestionAdmin)
admin.site.register(Choice, LessonAdmin)
admin.site.register(Submission)
```

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 ph0w0rd23

```
python3 manage.py createsuperuser
```

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
```

[Launch Django admin](#)

Assessment

For both Option 1: AI-Graded Submission and Evaluation and Option 2: Peer-Graded Submission and Evaluation

Take a screenshot of the Django admin site showing both the "Authentication and Authorization" section and the "OnlineCourse" section, and save it as `03-admin-site.jpeg` OR `03-admin-site.png`.

Django administration

WELCOME: ADMIN VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

Authentication and Authorization

Groups

+ Add

Change

Users

+ Add

Change

OnlineCourse

Choices

+ Add

Change

Courses

+ Add

Change

Instructors

+ Add

Change

Learners

+ Add

Change

Lessons

+ Add

Change

Questions

+ Add

Change

Submissions

+ Add

Change

Recent actions

My actions

None available

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).

Start Exam

What is Django?

☒ A Web framework

☐ A Movie

What is Django Model

☒ The single, definitive source of information about your data

☐ A web framework

☐ Perform ORM for developers

What is Django View

☒ Class-based view

☒ Function-based view

☐ It is a Controller

Submit

The changes will be made in `templates/onlinecourse/course_detail_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 %}
<div>
<!-- Remaining code will go here -->
{% endif %}
```

▼ Solution

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

2. Add a button to start the exam:

▼ Hint

```
<tag class="CLASS CLASS-primary CLASS-black" data-toggle="collapse" data-target="#exam">Start Exam</tag>
```

► Solution

```
<button class="btn btn-primary btn-black" data-toggle="collapse" data-target="#exam">Start Exam</button>
```

3. Add a collapsable div:

▼ Hint

```
<tag id="exam" class="collapse">
</tag>
```

► Solution

```
<div id="exam" class="collapse">
</div>
```

4. Add the Question logic inside a form:

▼ Hint

```
<div id="exam" class="collapse">
<form id="questionform" action="{% url 'onlinecourse:submit' course.id %}" method="POST">
  <!-- Question UI components will go here -->
</form>
</div>
```

► Solution

```
<div id="exam" class="collapse">
<form id="questionform" action="{% url 'onlinecourse:submit' course.id %}" method="POST">
  <!-- Question UI components will go here -->
</form>
</div>
```

5. Add Question UI:

▼ Hint

```
<div class="card mt-1">
<div class="card-header"><h5>{{ question.PROPERTY }}</h5></div>
<div class="card-body">
  <div class="form-group">
    <input type="checkbox" name="choice_{{choice.ID}}>
    <div class="form-check-label">
      {{ choice.CONTENT }}
    </div>
  </div>
</div>
```

► Solution

```
<div class="card mt-1">
<div class="card-header"><h5>{{ question.CONTENT }}</h5></div>
<div class="card-body">
  <div class="form-group">
    <input type="checkbox" name="choice_{{choice.ID}}>
    <div class="form-check-label">
      {{ choice.CONTENT }}
    </div>
  </div>
</div>
```

6. Add Choices components:

▼ Hint

```
{% for ITEM in question.FIELD.all %}
<div class="form-check">
  <input type="checkbox" name="choice_{{choice.ID}}>
  <div class="form-check-label">
    {{ choice.CONTENT }}
  </div>
</div>
{% endfor %}
```

► Solution

```
{% for choice in question.choice_set.all %}
<div class="form-check">
  <input type="checkbox" name="choice_{{choice.ID}}>
  <div class="form-check-label">
    {{ choice.CONTENT }}
  </div>
</div>
{% endfor %}
```

Final solution

View the final solution here:

Solution

```
{% if user.is_authenticated %}
</div>
<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-3">
        <div class="card-header">
          <h5>{{ question.content }}</h5>
        </div>
        <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 %}
  </div>
  <input class="btn btn-success btn-block" type="submit" value="Submit">
</form>
</div>
{% endif %}
```

Run in to test:

```
python3 manage.py runserver
```

Launch onlinecourse application

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

Commit your updated code to GitHub repository

Assessment

For Option 1: AI-Graded Submission and Evaluation

Copy and paste the public GitHub repository URL of your course_details_bootstrap.html file that displays the course name and all related lessons using Django template tags and Bootstrap and save it in a text file.

For Option 2: Peer-Graded Submission and Evaluation

Take a screenshot of your course_details_bootstrap.html page and save it as 04-course-details.jpeg OF 04-course-details.png.

```
<!-- Page content -->
<div class="container-fluid">
  <div class="card">
    <div class="card-body">
      <div class="card-columns-vertical">
        {% for lesson in course.lesson_set.all %}
          <div class="card mt-3">
            <div class="card-header">
              <h5>{{ lesson.title }}</h5>
            </div>
            <div class="card-body">
              <div class="form-group">
                <input type="checkbox" name="choice_{{choice.id}}" class="form-check-input"
                  id="{{choice.id}}" value="{{choice.id}}"/>{{ choice.content }}
                </div>
              </div>
            </div>
          </div>
        {% endfor %}
      </div>
    </div>
  </div>
  <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-3">
          <div class="card-header">
            <h5>{{ question.content }}</h5>
          </div>
          <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>
      </div>
      <input class="btn btn-success btn-block" type="submit" value="Submit">
    </form>
  </div>
</div>
```

Task 4: Test Data

You will now create test data for your application.

Add instructor

Add admin as an Instructor

Add instructor

User: admin

☒ Full time

Total learners: 0

Save

Save and add another

Save and continue editing

Course information

Field	Value
Name	Learning Django
Image	Download from link
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. tBForce .

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
```

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

```
def submit(PARAM1, PARAM2):
    course = get_object_or_404(Course, pk=PARAM2)
    user = request.user
    enrollment = Enrollment.objects.get(ARGUMENT1, ARGUMENT2)
    submission = Submission.objects.create(enrollment=enrollment)
    choices = extract_choices(request)
    submission.choices.set(choices)
    submission_id = submission.id
    return HttpResponseRedirect(reverse(viewname='onlinecourse:exam_result', args=(course_id, submission_id,)))
```

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_choices(request)
    submission.choices.set(choices)
    submission_id = submission.id
    return HttpResponseRedirect(reverse(viewname='onlinecourse:exam_result', args=(course_id, submission_id,)))
```

Route the submit view button in `urls.py`

Open **urls.py** in IDE

Hint

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

Solution

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

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, choice, 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

```
def show_exam_result(request, PARAM1, PARAM2):
    context = {}
    course = get_object_or_404(Course, pk=PARAM2)
    submission = MODEL.objects.get(id=PARAM2)
    choices = submission.choices.all()
    total_score = 0
    questions = course.RELATION_SET.all() # Assuming course has related questions
    for question in questions:
        correct_choices = question.RELATION_SET.filter(ARGUMENT1=True) # Get all correct choices for the question
        selected_choices = choices.filter(ARGUMENT2=question) # Get the user's selected choices for the question
        # Check if the selected choices are the same as the correct choices
        if set(selected_choices) == set(correct_choices):
            total_score += question.ATTRIBUTE # Add the question's grade only if all correct answers are selected
    context['REVIEW'] = course
    context['REVIEW'] = total_score
    context['REVIEW'] = choices
    return render(request, 'TEMPLATE_PATH', context)
```

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
    questions = course.question_set.all() # Assuming course has related questions
    for question in questions:
        correct_choices = question.choice_set.filter(is_correct=True) # Get all correct choices for the question
        selected_choices = choices.filter(question=question) # Get the user's selected choices for the question
        # Check if the selected choices are the same as the correct choices
        if set(correct_choices) == set(selected_choices):
            total_score += question.grade # Add the question's grade only if all correct answers are selected
    context['course'] = course
    context['grade'] = total_score
    context['choices'] = choices
    return render(request, 'onlinecourse/exam_result_bootstrap.html', context)
```

Exam results in `urls.py`

Open **urls.py** in IDE

Hint

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

```
▼ Solution

path = f'course/{int:course_id}/submission/{int:submission_id}/result/', views.show_exam_result, name='exam_result'),
```

Commit your updated code to [Github repository](#)

Assessment

For Option 1: AI-Graded Submission and Evaluation

Copy and paste the [public GitHub repository URL](#)s of your `views.py` and `urls.py` files. The `views.py` file must include the `submit` and `show_exam_result` functions and save it in a text file.

For Option 2: Peer-Graded Submission and Evaluation

Take a [screenshot](#) of your `views.py` file and save it as `85-views.jpeg` OR `85-views.png`.

```
▼ Hint

# Hint: Create a submit view to create an exam submission record for a course enrollment,
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='ollincourse:exam_result', args=(course_id, submission_id)))

# An example method to collect the selected choices from the exam form from the request object
def extract_answers(request):
    submitted_answers = {}
    for key in request.POST:
        if key.startswith('choice'):
            value = request.POST[key]
            choice_id = int(value)
            submitted_answers.setdefault(choice_id, []).append(request.POST[key])
    return submitted_answers

# Hint: Create an exam result view to check if learner passed the exam and show their question results and result for each question,
def show_exam_result(request, course_id, submission_id):
    context = {}
    course = get_object_or_404(Course, pk=course_id)
    submission = Submission.objects.get(enrollment_id=submission_id)
    choices = submission.choices.all()
    total_score = 0
    questions = course.question_set.all() # Assuming course has related questions

    for question in questions:
        correct_choices = subject_question.get_filters(correct_answer) # Get all correct choices for the question
        selected_choices = choices.filter(question=question) # Get the user's selected choices for the question

        # Check if the selected choices are the same as the correct choices
        if not correct_choices == int(selected_choices):
            total_score = question.grade # Add the question's grade only if all correct answers are selected

    context['course'] = course
    context['grade'] = total_score
    context['choices'] = choices

    return render(request, 'ollincourse:exam_result_bootstrap.html', context)
```

Assessment

For Option 1: AI-Graded Submission and Evaluation

Copy and paste the [public GitHub repository URL](#) of your `urls.py` file that includes the paths for `submit` and `show_exam_result` and save it in a text file.

For Option 2: Peer-Graded Submission and Evaluation

Take a [screenshot](#) of your `urls.py` file and save it as `86-urls.jpeg` OR `86-urls.png`.

```
ollincourse> cd app-with-database > cd ollincourse > urls.py
1 from django.urls import path
2 from django.conf import settings
3 from django.conf.urls.static import static
4 from . import views
5
6 app_name = 'ollincourse'
7 urlpatterns = [
8     # route is a string contains a url pattern
9     # view refers to the view function
10    # name the URL
11    path('routes/', views.index, name='index'),
12    path('registration/', views.registration_request, name='registration'),
13    path('login/', views.login_request, name='login'),
14    path('logout/', views.logout_request, name='logout'),
15    # ex: /ollincourse/5/
16    path('course/{id}/', views.course_detail_view.as_view(), name='course_details'),
17    # ex: /ollincourse/5/submit/
18    path('course/{id}/submit/', views.submit, name='submit'),
19
20    # Hint: Create a route for submit view
21    path('course/{id}/submit/', views.submit, name='submit'),
22    # Hint: Create a route for show_exam_result view
23    path('course/{int:course_id}/submission/{int:submission_id}/result/', views.show_exam_result, name='exam_result'),
24    # static settings.MEDIA_URL, document root=settings.MEDIA_ROOT]
```

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.

Style 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.

Home

Congratulations, John! You have passed the exam and completed the course with score 100 / 100

```
▼ Hint

<div>Congratulations, {{ USER_DETAILS }}!</div> You have passed the exam and completed the course with score {{ GRADE }}/100
```

```
▼ Solution

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

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.

Failed Sorry, Yan! You have failed exam with score 66 / 100

Re-test

Exam results

What is Django?

Correct answer: A Web framework

A Movie

What is Django Model

Correct answer: The single, definitive source of information about your data

A web framework

Not selected: Perform ORM for developers

What is Django View

Correct answer: Class-based view

Correct answer: Function-based view

It is a Controller

```
▼ Hint

<div>Failed</div> Sorry, {{ USER_DETAILS }}! You have failed the exam with score {{ GRADE }}/100
```


If you are submitting through the AI-grading Submission and Evaluation:

- 1. Submit the public GitHub URL of your repository that contains the **models.py** file with the `Question`, `Choice`, and `Submission` models.
- 2. Submit the public GitHub URL of your repository that contains the **admin.py** file with all seven imported classes along with the implementations of `QuestionInLine`, `ChoiceInLine`, `QuestionAdmin`, and `LessonAdmin`.
- 3. Upload a screenshot named **03-admin-site** showing the Django admin site with both the "Authentication and Authorization" section and the "OnlineCourse" section.
- 4. Submit the public GitHub URL of your repository that contains the **course_details_bootstrap.html** file showing the course name and all related lessons using Django template tags and Bootstrap.
- 5. Submit the public GitHub URL of your repository that contains the **views.py** file including the `submit` and `show_exam_result` functions.
- 6. Submit the public GitHub URL of your repository that contains the **urls.py** file with the paths for `submit` and `show_exam_result`.
- 7. Upload a screenshot named **07-final** showing a successful mock exam attempt with the "Congratulations" message, score, and exam results.

Option 2: Peer Graded Submission and Evaluation

If you are submitting through Peer Graded Submission and Evaluation:

- 1. Upload a screenshot named **01-models** showing the `models.py` file with the `Question`, `Choice`, and `Submission` models.
- 2. Upload a screenshot named **02-admin-file** showing the `admin.py` file.
- 3. Upload a screenshot named **03-admin-site** showing the required sections in the Django admin site.
- 4. Upload a screenshot named **04-course-details** showing the `course_details_bootstrap.html` file.
- 5. Upload a screenshot named **05-views** showing the `views.py` file.
- 6. Upload a screenshot named **06-urls** showing the `urls.py` file.
- 7. Upload a screenshot named **07-final** showing the mock exam "Congratulations" message, score, and detailed results.

Final Step

Before submitting, check that all your GitHub links open correctly and all screenshots clearly show the required content.

Author(s)

- [Yan Luo](#)
- [Muhammad Yahya](#)

© IBM Corporation. All rights reserved.