Web Development Using Application Frameworks
Coding Assignment: Forms
Instructions

Overview
The Forms coding assignment is the next in a series of assignments in which we will be developing the EZ University database system, a full C-R-U-D database application for a simplified university record keeping. In the Forms coding assignment, we add Create, Update, and Delete functionality for each of the EZ University model classes. The result is a full-featured C-R-U-D application.

Tools
I am expecting you to use the tools that are demonstrated in the tutorial videos: Anaconda and PyCharm.

Tool Versions
Use the versions of PyCharm Professional, Anaconda, and Python that we installed during Week 1 of the course when we created the *e4_trainor_django_course* virtual env. These versions are documented in *Instructions for Tool Versions, Installation, and Virtual Environments*.

Starter Files
Please note that I have provided starter files for use in the tutorial. You may download them from the Weekly Schedule:

- `starter_files_for_forms_coding_assignment.zip`

Tutorial Parts
This is a three-part tutorial.

In **Part 1**, we implement the *Create* pages. During the video, I code and test all parts required to implement the *Create* pages for the following EZU model classes:

- Instructor
- Section

While doing so, I use the following checklist:

1. Create the ModelForm subclass *ModelClass*Form in forms.py (e.g., InstructorForm).
2. Create the template for this page: *modelclass*_form.html (e.g., instructor_form.html).
3. Create the URL Pattern for this page.
4. Create the class-based view for this page: *ModelClass*Create in views.py (e.g., InstructorCreate).
5. In the *modelclass*_list.html file (e.g., instructor_list.html):
    a. Add the "Create New *ModelClass"* link code (e.g., "Create New Instructor").
6. Test.

At the end of the Part 1 tutorial video, you are instructed to perform similar coding and testing for the remaining EZU model classes on your own:

- Course
- Semester
- Student
- Registration

In **Part 2**, we implement the *Update* pages. During the video, I code and test all parts required to implement the *Update* pages for the following EZ University model classes:

- Instructor
- Section

While doing so, I use the following checklist:

1. Create the URL Pattern for this page (courseinfo_*modelclass*_update_urlpattern).
2. Add the get_update_url() method to the model class in models.py.
3. Add the *ModelClass*Update class-based view to views.py (e.g., InstructorUpdate).
4. Create the *modelclass*_form_update.html file: (e.g., instructor_form_update.html).
5. In the *modelclass* _detail.html file (e.g., instructor_detail.html):
    - add the "Edit *ModelClass* code (e.g., "Edit Instructor).
6. Test.

At the end of the Part 2 tutorial video, you are instructed to perform similar coding and testing for the remaining EZ University model classes on your own:

- Course
- Semester
- Student
- Registration

In **Part 3**, we implement the *Delete* pages. During the video, I code and test all parts required to implement the *Delete* pages for the following EZ University model classes:

- Registration (does not require a refuse_delete.html file).
- Instructor
- Section

While doing so, I use the following checklist:

1. Create the URL Pattern for this page. (courseinfo_*modelclass*_delete_urlpattern).
2. Add the get_delete_url() method to the model class in models.py .
3. Add the *ModelClass*Delete class-based view to views.py (e.g., InstructorDelete).
4. Create the *modelclass*_confirm_delete.html file (e.g., instructor_confirm_delete).
5. If appropriate (not Registration model class), create the *modelclass* _refuse_delete.html file (e.g., instructor_refuse_delete.html).
6. In the *modelclass* _detail.html file (e.g., instructor_detail.html):
    - add the "Delete *ModelClass* code (e.g., "Delete Instructor).
7. Test

At the end of the Part 3 tutorial video, you are instructed to perform similar coding and testing for the remaining EZ University model classes on your own:

- Course
- Semester
- Student

Exercises

1. **Exercise 1 (Regular)**
   Follow Parts 1, 2, and 3 of the tutorial instructions exactly.

2. **Exercise 2 (Challenge)**
   The challenge exercises for all of the EZU-based coding assignments will be based on finding ways to maximize the usefulness of the Django Admin app in the activities of the current assignment. When working on these challenge exercises, you may consult any resources that you may find helpful. I am offering the following resources as a starting point:

   1. Django Software Foundation (2025). *The Django admin site* in Django Documentation.

   2. Real Python (2025). *Customize the Django Admin With Python*.

   3. Tomazic, N. (2024). *Customizing the Django Admin* in TestDriven.io.

   This week's challenge exercise is to do the following:

   1. Write a short report that identifies ways in which we might customize the Django Admin app to help with the current week's work of writing and testing the code for this assignment.

   2. Identify 1 or 2 customizations. For each customization briefly explain how that customization might make our work easier or might make us more effective. Also, identify any coding work that would be required to implement the customization.

   3. Place your report in a text file named *forms_assignment_challenge.txt*. Place that text file directly into your Django Project directory.

   4. Format your work as a business report with proper paragraphs and sentences. Pay attention to grammar, choice of words, and spelling. You do not need to cite sources.

Code and Document Deliverables

You are expected to submit one properly organized PyCharm Django project that is ready to be tested using PyCharm. Please refer to my tutorial video for details. Even if you have decided to do Exercise 2, just submit one Django project.

Non-Code Deliverables

Please be sure that the project you submit includes the following:

1. A test user (username = “tester”, password = “{iSchoolUI}”
2. Sufficient test data present in the database to allow for testing all functions

Submission Method

Follow the process that I demonstrated in the tutorial video on submitting your work. This involves:

- Locating the properly named directory associated with your project in the file system.
- Compressing that directory into a single .ZIP file using a utility program.
- Submitting the properly named zip file to the submission activity for this assignment.

File and Directory Naming

Please use the following naming scheme for naming your PyCharm project:

surname_givenname_ezu

If this were my own project, I would name my PyCharm project as follows:

trainor_kevin_ezu

Use a zip utility to create one zip file that contain the PyCharm project directory. The zip file should be named according to the following scheme:

surname_givenname_ezu.zip

If this were my own project, I would name the zip file as follows:

trainor_kevin_ezu.zip

PLEASE NOTE: All file and directory names must be in lower case. Deductions will be made for submissions that do not conform to this standard.

Due Date

Please see the Weekly Schedule for the date and time when this assignment is due.

Last Revised
2025-03-06