PLTW COMPUTER SCIENCE

Activity 3.2.4

Web Templates: Image Gallery

goals

- Incrementally develop a web app using the Django framework and *Python*
- Learn how web pages are connected and accessed

description of web App

Image Gallery: Create an app as part of a website framework.

Essential Questions

1. What challenges have you faced in decomposing a large project and how have you managed it?
2. How is abstraction managing complexity in my program?
3. How is iteration managing program flow?

essential Concepts

- Django Framework
- Cascading Style Sheets
- HTML
- Databases
- Conditionals and Modulo

Create the Gallery and Discussion Apps

Using the notes in your PLTW Developer's Journal, work from previous activities, and the list below, you will work to add an image gallery and discussion app to your site. The image gallery will allow people to upload images and have discussions on a detailed view of the images.

1. Log in to your Cloud9 account.
2. Create a new workspace using the naming convention set forth by your teacher. Example: *gallerydiscussionsitelastnamefirstinitial*
3. Add a description that will help you identify this workspace among the others you have created. This will also help other developers who may share the workspace with you.
4. Choose a template: **Django**.
5. Add a gallery and discussion app to your site.
    - Use *manage.py* in bash to start the new app. Create *gallery* first.
    - Add necessary imports to the view file for the app, such as the *datetime*, *template loader*, and *HttpResponse*.
    - Define an *index* view for the app with some test output that identifies the app name.
    - Create a *urls.py* file in your app.
    - Add the *url import* and *general* views

      ```
      from . import views
      ```

    - Register the URL for your app.
    - Repeat steps a–f for an app called *discussion*.
6. In the *settings.py* file:
    - Add both apps to your list of *INSTALLED_APPS*; use previous apps to help.

      Important: Make sure the capitalization and punctuation patterns remain the same.

    - Update the *TIME_ZONE* to match your time zone.
    - Add a login redirect URL, that will direct a user to the discussion view after they have logged in.

      Hint: Use *LOGIN_REDIRECT_URL* as the variable name.

    - Add a logout redirect URL (based on the login redirect) that will direct a user to the discussion view after they log out.
7. Add the URL patterns to the site for the gallery and discussion apps:
    - Add the *url* patterns for each app.
    - If you get an error message, read the error to see what you need to add to the file.
8. To check your apps, navigate to the URL for each app.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Setting Up the Discussion App

The first app you will focus on is the discussion app. Similar to the bulletin board app, users need to log in before they can post. After you build some of the basic functions in this app, you will set up the gallery app. After they are both set up, you will connect the two apps to work together so that users can discuss a given image in the gallery.

The following outline provides the high-level steps to set up the discussion app. Use notes from your PLTW Developer's Journal and previous activities to help you. You will need to allow users to create an account, log in and log out, post only while logged in, and display the posts with date, author, title, and message. You will need a superuser to manage the app's users and posts.

Discussion Models

Set up the discussion app models with a *Post* class, as you have done in previous activities.

9. Import the *User* model from *django.contrib.auth.models*.
10. Include the following fields as you have done in previous apps:
    - post_title
    - post_text
    - pub_date
    - user
11. Create a migration for the models in your discussion app.
    - Check to see how you did migrations previously, if you need help.
    - If the models do not migrate, read and follow any errors to fix your code.
12. Run the migrations in bash for the model you just created.
13. Register your model with the app admin file.
    - Import *Post* from the models file.
    - Register the *Post* model.

Discussion Forms

To use the forms that Django already has in the framework, you need to import them. You will create and add to the forms class the built-in title and message fields from the Django forms database.

14. In the discussion app, add a *forms.py* file with the following parts:
    - Import forms from *django*.
    - *PostForm* class with the title and message fields.

Django Admin Portal

In the admin portal you can delete posts and users. You will use the admin portal to set up two users with one post each to help test the views as you set them up. It is important to keep track of the superuser login, so you can delete your test posts later.

15. Create a superuser account for your admin site.
16. Log in using the credentials that you just created.
17. Create a new User object with a different username than your admin account.
18. Create two new post objects, one for the superuser admin and one for the non-admin user object. This way, you will be able to verify that the author code is working to identify who is posting.

    Important: If you get a table error message while trying to save posts, it is possible that your migrations were out of order. To create tables, use the following code in the appropriate place to sync databases:

    ```
    python manage.py migrate --run-syncdb
    ```

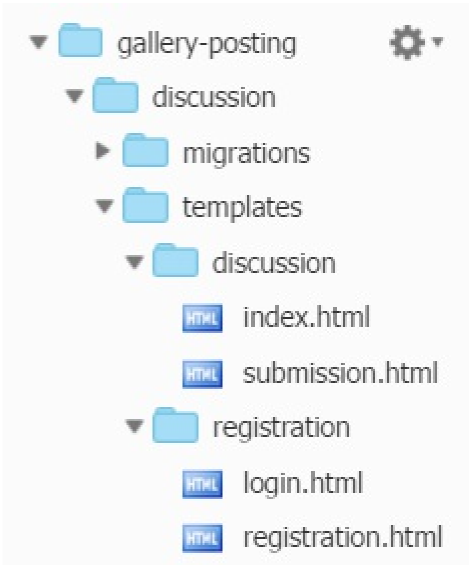
Templates for Authenticated Posting

Setting up templates helps to make your website and apps appear the way you want. It is important that the directory structure follows certain patterns, so that Django can find your folders. If you have a problem with accessing a template, check both the URL path you provided in the *Python* files, as well as the directory structure and naming.

19. Add the import for the login and logout views to the site's url file so that they can be accessed:

    ```
    url(r'^', include('django.contrib.auth.urls')),
    ```

20. Create a *templates* folder in the *discussion* folder.
21. Create a *discussion* folder in the *templates* folder.
22. Create an *index.html* file and a *submission.html* file in the discussion folder.
23. Create a *registration* folder in the *templates* folder.
24. In the registration folder, create a *login* file and a *registration* file.



25. Fill out the templates based on what you did in the bulletin board app. Make sure to include links to existing views including the login page and logout page.

26. If you get an error message, hover over the error to see what the problem is. Use the error and your previous app experience to fix any errors.

27. Testing the app now will show the login page and the logout page, but not the others. Take a few minutes to discuss with your partner what the next step is to direct the code to use the templates.

28. Implement the necessary steps so your index view uses the index template when you navigate to that URL.

- Be sure the index view shows the posts that you added in the admin portal, including the author, date, message title, and text.
- If necessary, troubleshoot and fix your code. Read and follow any errors that are provided.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Submission View

You may set up the submission view as you have done in the past, or you can make your code more efficient by using logical operators to combine conditionals in the submission view. If you choose do the same as you did in the past, use your resources to develop that submission view. Otherwise, use the following directions to combine conditionals for your submission view.

29. Using complex conditionals:
    - Register the submission view as a URL pattern in the correct *urls.py* file so you can check whether your code is working as you develop it.
    - Add import statements for *PostForm*, *Post*, *User*, and *UserCreationForm* to the *views.py* file.
    - Add a submission template.
    - Add a submission view function.
    - Add a conditional that checks if a user is authenticated AND is posting.

      ```
      request.user.is_authenticated()
      ```

    - Add a nested conditional that checks if the form is valid.
        - If true, create a new *PostForm*, get the form data, and save the title, message, date, and user. Create a context and template that will then take them to the index view.
        - If false, return a response so the user knows their form had problems, such as with the following code:

          ```
          return HttpResponse(form.errors.__str__())
          ```

        - Else if, the user is authenticated, but the post method does not equal POST, provide the submission view and load the form context.
        - Else if, the user is not authenticated, load the login template.
    - Add a return statement to the submission view function. Look at your index view function for help.
    - Add navigation links to the template views to make navigation easier while you are testing the site.

30. Test and verify that:

- ☐ You can navigate to the logout page and log out.
- ☐ While logged out, the submission page takes you to the login page.
- ☐ You can log in with one of the user accounts.
- ☐ From the submission page, making a post that has fields left blank provides an error message stating that you left fields blank and that they are required.
- ☐ After filling out the form correctly, you are able to post and are automatically directed to the discussion index view.

Important: If you run into errors, read the errors and determine where the problem is. It could be the URL pattern is not being registered properly, the submission template is not redirecting or displaying properly, or in the submission function in the view file.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

User Signup

To avoid having to use the admin portal to add new users, set up user registration. Use the previous activities to help with this process. Keep in mind that copying and pasting alone will not work. Make sure the code reflects the names used in this app and site. Below is a general outline of the steps to do in the process.

31. Create a success template that notifies a user when they have successfully registered their account.
32. Create a success view that selects the success template.
33. Register the success URL.
34. Navigate to the success URL to verify it is displaying properly.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

35. Create a registration view that uses a conditional to check whether the user is posting or not, to provide the correct view or save the inputs.
36. Create a conditional that checks whether the form is valid. If it is not, it should provide the feedback of why it is not valid.
37. Add a return statement at the end.
38. Register the registration URL.
39. Update all the navigation links so users can select a view to display.
40. Test the site’s registration and success views. Keep in mind that entering a password that is less than eight characters should return an error view.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

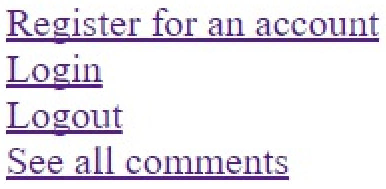
41. Test the discussion app:
    - ☐ Verify that you can use the interface to create a new user.
    - ☐ After adding the user, log in with the new user.
    - ☐ Make a post on the submission page and verify that the index view shows the new post with the new user.

Links Template

At this point, you have multiple views and multiple templates that need links for users to navigate. You are about to add even more. It is not practical to go into each template each time you add a new view and update all the links. In fact, you may already be tired of updating all the links.

Instead of adding to the amount of updating you need to do with the addition of each view, you will create a template just for links. Then you will have all the other templates reference the links template. By doing this, you only have to update the one links template, and the rest of the templates will update,   because they just reference what is in the links template.

42. Add a template called *links.html*. Which folder should you add it to—discussion or registration?
43. Copy and paste all the links into the template file.
44. Add HTML tags to format the links in a user friendly way. Below is an example, but yours may look different based on the html tags you add.


Register for an account
Login
Logout
See all comments

Important: You can now update the links in this one template file and call this template into all the other templates. Use the following base code to add the specific navigation direction for your directory. Then, you can add it to all your other templates and have all the links.

```
{% include "_____ /______.html" %}
```

45. Replace the navigation links in all the discussion app templates to use this one file.
46. Use the navigation in the views to verify that each view has the navigation and that the links take you where you think you are going when you select a link.
47. With each additional template you add, be sure to add the line of code to keep consistent navigation in all your website views.
48. Test your app to make sure it is still functioning properly and that you can navigate between pages without modifying the URL address directly in the browser navigation bar.

- ☐ Navigate with links through all the pages of your site.
- ☐ Register a new user.
- ☐ Log out and back in with a different user account.
- ☐ Posts only work while you are logged in.

If all of these function as intended, continue on. If parts of the app do not work, it is important to fix it now before you move to the second app. As you build the gallery app, you will connect it to your discussion app. Troubleshooting is easier when you know everything up until this point is working properly.

Gallery App

Now that the discussion app is set up, it is time to focus on the new gallery app. In the gallery app, a user will be able to upload an image with a title to display. Other users will be able to select an image and have a discussion about specific images that have been uploaded in a detailed view of the images.

Gallery Models

You will add some similar fields from previous app models, but you will also now add new fields to an Image class model: file and comments. These fields will allow a user to upload a file and then make comments about that specific image.

49. Update your gallery models to include the following fields in an Image class:
    - pub_date
    - image_title
    - author - whatever user is logged in

    - Create and name a field “file”.
    - To use the file upload system embedded in the Django database models, use the *FileField* class from the *django.db* models.
    - Include an argument field *upload_to* with the value 'images/' paired to the *upload_to* argument. The argument will identify where in the directory structure the file should be uploaded. The path needs to be specific, so that later, it will be easy to identify where the program can find the uploaded images.
    - Create a "comments" field.
    - Use the *ManyToManyField* class from the *django.db* models.
    - Include the string *'discussion.Post'* as an argument.

    ```
    comments = _____._______('________.__________')
    ```

    -
    -
    -

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

In the models Image field, you specified the directory location as ‘images/’. You need to make sure this location exists in the directory. Part of the built-in Django framework is to look for a ‘media’ folder under the main site directory.

50. Create a ‘media’ folder in the top level directory of your site.
51. Create another folder inside the media folder named ‘images’ to take advantage of the built-in framework of uploading images to the place you specified in the model.

52. Now that you have created the Image model, add it to your app’s admin file.

```
from .models import Image
admin.site.register(Image)
```

Gallery Forms

Much like previous apps, the models are one part that expresses what the classes are, and the forms file is what allows you access to the built-in Django forms. You need to set up a form file that will use the built-in features to simplify uploading an image.

53. Create a *forms* file in the gallery app.

54. Import forms from django.

55. Import the UserCreationForm from django.contrib.auth.forms.

56. Set up an ImageUploadForm class:

```
class ________(______.______):
```

57. Set up a title field:

```
title = ______.CharField(_____='____', max_length=___)
```

58. Set up an image field forms.FileField() :

```
image = _____.FileField(____='_____')
```

Updating the models and forms requires you to set up additional database tables to handle these new fields. Trying to run your website now might get you the following message:

Your models have changes that are not yet reflected in a migration, and so won’t be applied. Run 'manage.py makemigrations' to make new migrations, and then re-run 'manage.py migrate' to apply *them.*

Important: If you get error messages that suggest you do something, it is best to follow the advice given. If you receive the error message above, run a migration so space is set up for each of the fields you have added to the gallery app.

Upload View Function

Now that all the fields in your *Image* class and *ImageUploadForm* class are outlined, add an *upload* view in your gallery app. The upload view should allow a user to type in a title and select the path to the image that they want to upload.

59. Create the upload view and function. The following prompting questions may be helpful:
    - How do you make the upload view a valid URL address?
    - What imports do you need to add to access the Image class model in the views file?
    - What import do you need to add to access the User information from the *django.contrib.auth.models*?
    - What import do you need to add to access the *ImageUploadForm*?
    - Do you have the imports for loading and rendering templates?
    - What import do you need to get the date and time?
60. In the upload function, a conditional should identify if the user is authenticated and posting. If that conditional is true:
    - A form variable should call upon the *ImageUploadForm* constructor with two arguments: *request.Post* and *request.FILE*
    - A conditional should check if the form is valid. If the form is valid, the code should:
        - ☐ Add the fields for author, file, title, and date and time.
        - ☐ Load the index template for the gallery app.

    Important: To access files in request.FILES: If the file in your Form class was named my_file, use request.FILES['my_file']. Make sure you are using the correct file name and pathway from your website directory.

    Else if, the user is not posting, the function should store an empty *ImageUploadForm* and load the upload template, passing in the form as a context item.

61. Add other conditional alternatives in the proper places to let the user know why the form is not valid, or redirect them to the login page.
62. Test your app. If an error occurs, read the error message to determine why you are getting it now.

    Hint: If you have not yet defined a template, look at the error message to find where it expresses that this is the problem. After you identify where the error message is providing this feedback, continue with the steps to set up your templates.

63. Set up the templates folder in your gallery app.
64. Set up the gallery and registration folders inside the template folder.
65. Set up a blank index template in the correct folder.
66. Copy and paste the registration and login templates from your discussion app.
67. Create an upload template in the correct folder.

68. With your elbow partner, discuss the parts of the following content for the upload template:

```
<form method="post" action="{% url 'upload' %}" enctype="multipart/for

<div>
    <td>{{ form.title.label_tag }}</td>
    <td>{{ form.title }}</td>
</div>

<div>
    <td>{{ form.image.label_tag }}</td>
    <td>{{ form.image }}</td>
</div>

<div>
    <input type="submit" value="Upload" />
    <input type="hidden" name="next" value="{{ next }}" />
</div>

</form>
```

69. Navigate to your upload view. At this time, you should see an empty form with a title, a Choose File button, and an Upload button. If you have not already, include the link to the links template.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

70. Test to make sure you can enter a title and select an image file.
    - ☐ Clicking the **Upload** button should take you to a different view. However, you cannot see any images.
    - ☐ You should be able to upload an image. However, the image does not show up, because you have not directed any part of the website to display it.
71. Update the gallery index view to show the image uploads by date, as you did with the posts in the discussion app.
72. Use the following return statement in your *view ()* for the index view function:

```
return render(request, 'gallery/index.html', {'image_list':image_list}
```

73. With your elbow partner, discuss the parts of the following content for the index template. Pay special attention to the conditionals and the loop built into this template. What is each part doing when?

```
{% if image_list %}
<div>
```

```
{% for image in image_list %}
{% if image.file == None %}
Invalid image file
{% else %}
<a href="{{image.id}}/"><img src="{{ image.file.url }}" width=100/></a
{% endif %}
{% if forloop.counter0|divisibleby:4 and forloop.counter0 > 1 %}
</div><div>
{% endif %}
{% endfor %}
</div>
{% else %}
<p>No posts are available.</p>
{% endif %}
```

PLTW DEVELOPER'S JOURNAL Not all code requires indention like *Python* does. You saw the same code with and without indention. Record some information about the advantages of indention, even if the language does require it. What else could you do to help make your code more readable?

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

74. Navigate to the homepage for your gallery app. The image does not show up, only an image box.

    Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

75. To update the settings file for the website to access the media folder you added, add the following to the end of the file:

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

76. Add imports to the site's URL patterns to access the server and display the images:

```
from django.views.static import serve
import settings
```

77. To register the URL pattern based on the imports and settings updates you have done, use:

```
url(r'^media/(?P!path>.*)$', serve, {'document_root': settings.MEDIA_F
```

78. Add links to navigate between the gallery app views and the discussion app views, including the login, logout, and registration pages.

When you navigate back to your gallery index, it should display the pictures that have been uploaded. If while testing the form, you selected to **Upload** without selecting an image first, you might see some blank boxes still. Upload a new image to verify that the app is working as expected. If it is not showing images, verify that you updated all the parts as directed and saved all the files before you refresh the index page.

Putting the Apps Together

You now have an app where a user can upload and display images and a second app where a user can make posts. You also have navigation set up to allow users to navigate both apps in the website.

You are going to combine the functionality of both apps into one detail view. You will add a detailed view of the images, so users can make posts specific to the image they are looking at. In each step, discuss with your partner what each part is doing.

79. Add a template for this detail view in the gallery app. Discuss the following code and add it to the detail template:

```
<ul>
<img src="{{image.file.url}}" width=200/>
<p>Author: {{image.author}}</p>
<p>Date of upload: {{image.pub_date}}</p>
<p>Title: {{image.image_title}}</p>

<h3><a href = "{% url 'submission' image.id %}">Submit a Comment</a></
{% for comment in image.comments.all %}
    <li>
        <p>{{comment.post_title}} {{comment.user.username}} {{comment.
        <p>{{comment.post_text}}</p>
    </li>
{% endfor %}
</ul>
```

80. In the gallery views, add the *Http404* import from *django.http*.

    The detail view will try to display the images. If the image does not exist or cannot be displayed, the detail view will show an error message that it cannot display that image.

81. To add a detail view function, use the following code:

```
def detail(request, image_id):
    try:
        image = Image.objects.get(pk=image_id)
    except Image.DoesNotExist:
        raise Http404("Image does not exist")
    return render(request, 'gallery/detail.html', {'image':image
```

82. Register the *url* pattern with the gallery app:

```
url(r'^(?P!image_id>[0-9]+)/$', views.detail, name='detail'
```

83. Add to the *discussion urls* a pattern that combines the *image id* with the *submission* to create a submission thread specific to each picture, add:

```
url(r'^(?P!image_id>[0-9]+)/submission/$', views.submission, name='sub
```

84. In the discussion view that handles the submission of posts, add an *Image* import from *gallery.models*.

85. Add to the submission function an argument of *image_id* with the request.

86. Update the submission view function to include saving the image id and the post associated with it before the post is saved to the database:

```
image = Image.objects.get(id=image_id)
image.comments.add(post_to_submit)
```

87. Add to the context of the submission post *'image_id' : image_id*:

```
context = {'post_list' : post_list, 'image_id' : image_id}
```

88. Update the submission template to handle the *image_id*. Add *image_id* in the action section of the submission template.

```
<form method="post" action="{% url 'submission' image_id %}">
```

Now, when you click on an image from your gallery home page, you should be redirected to the detailed view for that image.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

PLTW DEVELOPER'S JOURNAL CSRF was handled differently in this app than it has been in the past. Work with your partner to figure out how and document it in your journal.

89. Test to be sure your app has full functionality:

    - ☐ Upload an image.
    - ☐ Verify that the new image is in the gallery index view.
    - ☐ Select the new image to go to the detail view that shows the information about that picture.
    - ☐ Click the **Submit a Comment** link. The link should take you to a URL that shows you are submitting to the index number of that image.
    - ☐ Add a title and a post in the fields and store the post.
    - ☐ You should be rerouted to the discussion index view after submitting the post.
    - ☐ The discussion view should display all the posts that have been made, not just the posts for the one image.
    - ☐ Navigate back to the detail view of the image you posted a comment for. You should see only the comment you posted for that image.
    - ☐ Try posting a comment to another image and verify that all posts can be seen on the discussion homepage, but that each image only displays the posts made for that image.
    - ☐ Try to submit a post now. You will get an error, because regular posts do not have an image value associated with them. Remove from all views the navigation link to submitting a post. If you used the links template, you only need to delete it from there.
    - ☐ Update the links template to navigate to all the views (except making a post), so that the words displayed to the user make sense and allow them to easily navigate the site.

Conclusion

1. Describe a way in which you used basic programming constructs differently in this activity than you have in previous activities.
2. How did you interpret and respond to the essential questions? Capture your thoughts for future conversations.