Building a Classified Ad Web Site

In this assignment, you will build a web site that is roughly equivalent to

https://chucklist.dj4e.com/m1

This web site is a classified ad web site. People can view ads without logging in and if they log in, they can create their own ads. It uses a social login that allows logging in using github accounts.

You will build this application by borrowing parts and pieces from the code that runs

https://samples.dj4e.com/

and combining them into a single application.

Make sure to get the correct version of dj4e-samples.

Getting the Right Version of the Sample Code

Important: During Fall 2023 the course is making a transition from Django 4.0 to 4.2 and from Bootstrap 3 to Bootstrap 5. Depending on when you started, you might be using Django 4.0 or 4.2 at this point in the course.

There is no need to switch Django versions in the middle of the course - all we need to do is make sure the provided sample code you are using corresponds to the Django version you are using.

First check what version of Django is running in your virtual environment:

```
workon django4
python -m django --version
```

This should be 4.0.n or 4.2.n You need to checkout the correct version of the sample code that corresponds to your Django version.

If you have never checked out the sample code on PythonAnywhere you can:

```
cd ~
git clone https://github.com/csev/dj4e-samples
```

Once you have the samples checked out (either before or just now), check out the right version:

```
cd ~/dj4e-samples

git checkout main        # For Django 4.2

git checkout django40    # For Django 4.0
```

Once you have checked out the correct version, continue with:

```
git pull
pip install -r requirements4.txt
python manage.py check
```

If `python manage.py check` fails with a traceback, stop and get some help. If `python manage.py check` works without a traceback, you should be ready to continue with this assignment and the rest of the course.

Using the MySQL Database

To switch to using the MySQL database outside of the PythonAnywhere, you might need to install the MySQL client and possibly the server on your system before `pip install` will work using instructions at https://pypi.org/project/mysqlclient/ - once you have the MySQL prerequisites installed or are using PythonAnywhere, you can run:

```
pip install mysqlclient==1.4.6
```

So your Python code can connect to MySQL databases. If you are having installation problems, you can keep using the SQLite database but it will mean that your application will start to run much more slowly as we add complexity to the application. It is especially to switch to MySQL on PythonAnywhere.

Important Note: If you find you have a problem saving files in the PythonAnywhere system using their browser-based editor, you might need to turn off your ad blocker for this site - weird but true.

Important Note: If you are using PowerPoint slides for sample code, *never* copy and paste from Powerpoint or Word into PythonAnywhere. Microsoft products use *weird* characters for quotes and dollar signs and they can break your code but look perfect. A good rule is to never copy code from any Microsoft product.

Pulling In Code From Samples

In this section, we will break and then fix your `settings.py` and `urls.py` . When this is done, the autos, cats, dogs, etc will stop working unless you add them back to these two files. It is OK for these applications not to be working. The autograder will just look at /ads.

(1) Copy the `settings.py` and `urls.py` files and the entire `home` folder from the `dj4e-samples` project:

```
cp ~/dj4e-samples/dj4e-samples/settings.py ~/django_projects/mysite/mysite
cp ~/dj4e-samples/dj4e-samples/urls.py ~/django_projects/mysite/mysite
cp -r ~/dj4e-samples/home/* ~/django_projects/mysite/home
```

(2) Edit `~/django_projects/mysite/mysite/settings.py` and then delete all the `INSTALLED_APPLICATIONS` after `home` . You will have to search and replace `dj4e-samples` with `mysite` in a few places. Also set the name of your application in the `settings.py` file to something other than `ChucksList` :

```
# Used for a default title
APP_NAME = 'ChucksList'
```

This shows up in default page titles and default page navigation.

(3) Edit your `django_projects/mysite/mysite/urls.py` and remove all of the `path()` calls to the sample applications. Make sure to keep the `path()` to `home.urls` . Also keep the `site` and `favicon` rules in your `urls.py` .

(4) Edit the file `django_projects/mysite/home/templates/home/main.html` and replace the contents with this:

```
{% extends "base_bootstrap.html" %}
{% block content %}
    <h1>Welcome to {{ settings.APP_NAME }}</h1>
    <p>
    Hello world.
    </p>
{% endblock content %}
```

(5) At this point, you should be able to run:

```
python manage.py check
```

Keep running `check` until it does not find any errors.

If you get an error like `Could not import github_settings.py for social_django` when running `manage.py` or restarting your PythonAnywhere webapp, don't worry - you will see this warning until you set up the social login.

If you are running on your local computer, (i.e not using PythonAnywhere) you can skip to step 9, otherwise continue with these steps. Steps 6-8 are for PythonAnywhere.

(6) We are going to switch your application on PythonAnywhere from using an SQLite database to a MySQL database for the rest of this course. If you keep running SQLite and your

application stores too much data it will start to slow down. If you are running locally, you can keep using SQLite.

(7) To use MySQL, first go to the `Databases` tab in PythonAnywhere. Make a MySQL database named `ads` and choose a name and password and write them down.

(8) Edit `~/django_projects/mysite/mysite/settings.py` and find the existing value for the `DATABASES` variable and comment it out.

```
# DATABASES = {
#     'default': {
#         'ENGINE': 'django.db.backends.sqlite3',
#         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
#     }
# }
```

Add an entry to point Django at your newly created MySQL database. In this example, your PythonAnywhere account is `drchuck` and the database you created is `ads` and the password you set for the database is `phone_8675309` . Change the sample values below to the values for your database.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'drchuck$ads',
        'USER': 'drchuck',
        'PASSWORD': 'phone_8675309',
        'HOST': 'drchuck.mysql.pythonanywhere-services.com',
         'OPTIONS': {
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
        },
    }
}
```

(9) Once `check` works you will need to run your migrations and make a new administrator account:

```
cd ~/django_projects/mysite
python manage.py makemigrations      # Might say "no changes"
python manage.py migrate
python manage.py createsuperuser
```

If the `makemigrations` works and `migrate` fails, you may have an error in the `DATABASE` section of your `settings.py` . You can edit your `settings.py` and rerun the `migrate` until it works.

Somtimes `migrations` or `makemigrations` takes up to a few minutes - if they are running and not showning any errors - please be patient. If the migrations process is interrupted - you might need to drop your MySQL tables and run the migrations again - instructions are shown below to drop the databases tables so you can run the migrations on a fresh database.

(10) If you restart your web application, there won't be many working urls. Try these two to see if you have the home code working properly:

```
https://your-account.pythonanywhere.com/
https://your-account.pythonanywhere.com/favicon.ico
https://your-account.pythonanywhere.com/accounts/login
```

Look at how pretty the login form looks :). Don't worry about the social login yet. We will get to that later. Favicons are shown in the browser tabs. We will get to favicons later too :)

Building the Ads Application

In this section, you will pull bits and pieces of the sample applications repository and pull them into your `ads` application.

Important Note: If you find you have a problem saving files in the PythonAnywhere system using their browser-based editor, you might need to turn off your ad blocker for this site - weird but true.

(1) Create a new `ads` application within your `mysite` project:

```
cd ~/django_projects/mysite
python manage.py startapp ads
```

Then add the application to your `mysite/mysite/settings.py` and `mysite/mysite/urls.py` .

(2) Use this in your `ads/model.py` :

```
from django.db import models
from django.core.validators import MinLengthValidator
from django.conf import settings

class Ad(models.Model) :
    title = models.CharField(
            max_length=200,
            validators=[MinLengthValidator(2, "Title must be greater than 2 characters")]
    )
    price = models.DecimalField(max_digits=7, decimal_places=2, null=True)
    text = models.TextField()
    owner = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # Shows up in the admin list
    def __str__(self):
        return self.title
```

(3) Copy the `owner.py` from `myarts` to your ads application. This is the one file you **do not** have to change at all (thanks to object orientation 😊).

(4) The files `admin.py`, `views.py`, `urls.py`, and the `templates` in the `myarts` folder will require significant adaptation to be suitable for a classified ad application and the above model. A big part of this assignment is to use the view classes that are in `owner.py` and used in `views.py`. The new `owner` field should not be shown to the user on the create and update forms, it should be automatically set by the classes like `OwnerCreateView` in `owner.py`. If you see an "owner" drop down in your create screen the program is not implemented correctly and will fail the autograder.

(5) Adapt the templates in `myarts/templates/myarts` as a starting point to create the needed templates in `ads/templates/ads`.

(6) When you are implementing the update and delete views, make sure to follow the url patterns for the update and delete operations. They should be of the form `/ad/<int:pk>/update` and `/ad/<int:pk>/delete`. Something like the following should work in your `urls.py`:

```
from django.urls import path, reverse_lazy
from . import views

app_name='ads'
urlpatterns = [
    path('', views.AdListView.as_view(), name='all'),
    path('ad/<int:pk>', views.AdDetailView.as_view(), name='ad_detail'),
    path('ad/create',
        views.AdCreateView.as_view(success_url=reverse_lazy('ads:all')), name='ad_create'),
    path('ad/<int:pk>/update',
        views.AdUpdateView.as_view(success_url=reverse_lazy('ads:all')), name='ad_update'),
    path('ad/<int:pk>/delete',
        views.AdDeleteView.as_view(success_url=reverse_lazy('ads:all')), name='ad_delete'),
]
```

(7) As you build the application, use `check` periodically as you complete some of the code.

```
python manage.py check
```

(8) Once your application is mostly complete and can pass the `check` without error, add the new models to your migrations and database tables:

```
python manage.py makemigrations
python manage.py migrate
```

Adding the Bootstrap menu to the top of the page

Important: Follow these instructions closely because during Fall 2023, we are switching from Bootstrap 3 to Bootstrap 5. If you started the course with Django 4.0 and Bootstrap 3, you should finish the course with Bootstrap 3. There is no need to switch. In this section, we show you how to check your Bootstrap version and provide code for both Bootstrap 3 and Bootstrap 5.

Next we will add the bootstrap navigation bar to the top of your application as shown in:

https://chucklist.dj4e.com/

This top bar includes a "Create Ad" navigation item and the login/logout navigation with gravatar when the user logs in.

(1) Edit all four of the `ads_` files in `ads/templates/ads` to change them so they extend `ads/base_menu.html` . Change the first line of each file from:

```
{% extends "base_bootstrap.html" %}
```

to be:

```
{% extends "base_menu.html" %}
```

(2) **Important:** We need to create `home/templates/base_menu.html` depending on the Bootstrap version you are using. Open your in your `home/templates/base_bootstrap.html` and look for a `<link` line that is loading Bootstrap. You need to choose one of the following two versions of `base_menu.html` .

If the line in `base_bootstrap.html` is including Bootstrap 5:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
```

Then you are using Bootstrap 5.0 and use the following code for your `base_menu.html` :

```
{% extends 'base_bootstrap.html' %}
{% load app_tags %} <!-- see home/templatetags/app_tags.py and dj4e-samples/settings.py -->
{% block navbar %}
<nav class="navbar navbar-expand-lg navbar-dark bg-dark" style="border-radius:10px !important">
  <div class="container-fluid">
    <a class="navbar-brand" href="{% url 'ads:all' %}">{{ settings.APP_NAME }}</a>
    <ul class="navbar-nav">
      {% url 'ads:all' as x %}
      <li {% if request.get_full_path == x %}class="active"{% endif %}>
          <a class="nav-link" href="{% url 'ads:all' %}" role="button">Ads</a></li>
    </ul>
    <ul class="navbar-nav">
      {% if user.is_authenticated %}
      <li>
         <a class="nav-link" href="{% url 'ads:ad_create' %}">Create Ad</a>
      </li>
      <li class="nav-item dropdown">
         <a class="nav-link dropdown-toggle" href="#" id="rightnavDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
            <img style="width: 25px;" src="{{ user|gravatar:60 }}"/><b class="caret"></b>
        </a>
        <ul class="dropdown-menu dropdown-menu-end" aria-labelledby="rightnavDropdown">
            <li><a class="dropdown-item" href="{% url 'logout' %}?next={% url 'ads:all' %}">Logout</a></li>
        </ul>
       </li>
       {% else %}
       <li class="nav-item"><a class="nav-link" href="{% url 'login' %}?next={% url 'ads:all' %}">Login</a></li>
       {% endif %}
    </ul>
  </div>
</nav>
{% endblock %}
```

If the line in `base_bootstrap.html` is including Bootstrap 3:

```
<link ... href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap-theme.min.css" ...
```

Then you are using Bootstrap 3.0 and use the following code for your `base_menu.html`:

```
{% extends "base_bootstrap.html" %}
{% block navbar %}
{% load app_tags %}
<nav class="navbar navbar-default navbar-inverse">
  <div class="container-fluid">
    <div class="navbar-header">
        <a class="navbar-brand" href="/">{{ settings.APP_NAME }}</a>
    </div>
    <!-- https://stackoverflow.com/questions/22047251/django-dynamically-get-view-url-and-check-if-its-the-current-page -->
    <ul class="nav navbar-nav">
      {% url 'ads' as ads %}
      <li {% if request.get_full_path == ads %}class="active"{% endif %}>
          <a href="{% url 'ads:all' %}">Ads</a></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
        {% if user.is_authenticated %}
        <li>
        <a href="{% url 'ads:ad_create' %}">Create Ad</a>
        </li>
        <li class="dropdown">
            <a href="#" data-toggle="dropdown" class="dropdown-toggle">
                <img style="width: 25px;" src="{{ user|gravatar:60 }}"/><b class="caret"></b>
            </a>
            <ul class="dropdown-menu">
                <li><a href="{% url 'logout' %}?next={% url 'ads:all' %}">Logout</a></li>
            </ul>
        </li>
        {% else %}
        <li>
        <a href="{% url 'login' %}?next={% url 'ads:all' %}">Login</a>
        </li>
        {% endif %}
    </ul>
  </div>
</nav>
{% endblock %}
```

(3) Find the line in your `base_bootstrap.html` that looks like this:

```
    <meta name="dj4e-code" content="99999999">
```

and change the `9999999` to be "42169769777706" Note that there will be two meta tags, one for dj4e-code and one for dj4e - keep both in this file.

Make sure to check the autograder for additional markup requirements.

When you are done, you should see an 'Ads' menu on the left and a 'Create Ad' link on the right just like the sample implementation.

Manual Testing

It is always a good idea to manually test your application before submitting it for grading. Here are a set of manual test steps:

- Make two accounts if you have not already done so
- Log in to your application on the first account
- Make sure the menu bar shows at the top of all of the screens - the autograder gets grumpy about a missing menu on a page
- Create an ad
- Try to submit an add with no title - make sure that it complains
- Create an ad
- In the all ads list make sure that the edit / delete button shows correctly
- View its details - make sure the edit / delete button shows up correctly
- Update the ad, check that the details are correct after the update
- Delete the ad - just to make sure it works
- Create two more ads
- Log in on the second account - make sure you **do not** see edit / delete buttons on the existing ads
- Go into one the detail for the ad created by the other user - make sure you **do not** see edit / delete buttons
- Create a new ad on the second account
- Make sure that in the "all ads list" the edit / delete buttons are only present on the ads the second user "owns"
- Delete the ad

Fun Challenges

(1) Make yourself a gravatar at https://en.gravatar.com/ - it is super easy and you will see your avatar when you log in in your application and elsewhere with gravatar enabled apps. The gravatar can be anything you like - it does not have to be a picture of you. The gravatar is associated an email address so make sure to give an email address to the user you create with `createsuperuser`.

(2) Change your `home/static/favicon.ico` to a favicon of your own making. I made my favicon at https://favicon.io/favicon-generator/ - it might not change instantly after you update the favicon because they are cached extensively. Probably the best way to test is to go right to the favicon url after you update the file and press 'Refresh' and/or switch browsers. Sometimes the browswer caching is "too effective" on a favicon so to force a real reload (command/ctrl + shift + r) to check if the new favicon is really being served you can add a GET parameter to the URL to force it to be re-retrieved:

```
https://chucklist.dj4e.com/favicon.ico?x=42
```

Change the `x` value to something else if you want to test over and over.

(3) To make the social login work. Take a look at github_settings-dist.py, copy it into `mysite/mysite/github_settings.py` and go through the process on github to get your client ID and secret. The documentation is in comments of the file. Also take a look at dj4e-samples/urls.py and make sure that the "Switch to social login" code is correct and at the end of your `mysite/mysite/github_settings.py` .

You can register two applications with github - one on localhost and one on PythonAnywhere. If you are using github login on localhost - make sure that you register `http://127.0.0.1:8000/` instead of `http://localhost:8000/` and use that in your browser to test your site. If you use localhost, you probably will get the `The redirect_uri MUST match the registered callback URL for this application.` error message when you use social login.

Working with Ambiguity

This assignment is more vague than previous assignments - on purpose. The goal is to get closer to the development model of actual applications. You know what you want to build and start with a mostly blank slate. You look at sample code, reuse some code from stuff you built earlier, do some online searching and glue pieces of what you find together to make your application. Of course as you are gluing bits from various places together, they always break and you have to adjust things so they fit in your application.

So this is kind of like the real world - when you have to build your own first application for someone else.

It is not tricky on purpose. We want you to succeed in this assignment. But we do want you to do less cutting-and-pasting and more writing Django applications.

Debugging: Searching through all your files in the bash shell

If you have errors, you might find the `grep` tool very helpful in figuring out where you might find your errors. For example, lets say after you did all the editing, and went to the ads url and got this error:

```
NoReverseMatch at /ads
'myarts' is not a registered namespace
```

You *thought* you fixed all the instances where the string "myarts" was in your code, but you must have missed one. You can manually look at every file individually or use the following command to let the computer do the searching:

```
cd ~/django_projects/mysite
grep -ri myarts *
```

You might see output like this:

```
ads/templates/ads/ad_list.html:<a href="{% url 'login' %}?next={% url 'myarts:all' %}">Login</a>
```

The `grep` program searches files in the current folder and subfolders for any lines in any file that have the string "myarts" in them and shows you the file name and the line it is mentioned.

The `grep` command is the "Generalized Regular Expression Parser" and is one of the most useful Linux commands to know. The 'r' means 'recursive' and the 'i' means 'ignore case. The `grep` program will save you so much time 😊.

Some Common Errors in This Assignment

Since you are in effect starting with a brand new `mysite/settings.py` and `mysite/urls.py`, you might find a few problems when you are running `python manage.py check` - I will keep a list of the common problems and their solutions here:

(1) If you see an error message "TypeError: 'module' object is not iterable" when you are running `python manage.py check`, this maybe because you mistakenly edited the `ROOT_URLCONF` value in `settings.py` - your value should be:

```
ROOT_URLCONF = 'mysite.urls'
```

More will be added as the problems are identified.

(2) If you have a problem running `migrate` or `makemigrations` in step 9 above, you might want to start with a fresh MySQL database. Since we are using a MYSQL server, we can't just delete the SQLite file and start over - but it is not much more difficult.

First go into `Consoles` and start a `MySQL` console. You should go into a shell and see a prompt like this - type the command `SHOW DATABASES;` to find your database:

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dj4e$chucklist     |
| dj4e$default       |
+--------------------+
3 rows in set (4.05 sec)
mysql>
```

Note - **never touch** the `information_schema` database - if you mess with this you break your entire MySQL installation and may need to create a completely new PythonAnywhere account. Leave `information_schema` alone.

Pick the database you are using (in your `settings.py`) and issuer the `USE` command to select the database and run the `SHOW TABLES;` command:

```
mysql> use dj4e$chucklist;
Database changed
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_dj4e$chucklist   |
+----------------------------+
| ads_ad                     |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
| social_auth_association    |
| social_auth_code           |
| social_auth_nonce          |
| social_auth_partial        |
| social_auth_usersocialauth |
+----------------------------+
10 rows in set (0.00 sec)
mysql>
```

Then we will get rid of the `ads_ad` table and its associated migration records:

```
mysql> DROP TABLE ads_ad;
mysql> DELETE FROM django_migrations WHERE app='ads';
```

Then, in the bash shell, you can remove and re-make the migrations

```
cd ~/django_projects/mysite
rm ads/migrations/00*
```

Then go back to step 9 and pick up with the `makemigrations` and `migrate` steps as well as `createuser` is needed.