MWC3 Web Development Workshop with Django

The contents of this repository is the finished version of the MWC3 2024 Django workshop.

This README that you are reading is a step-by-step written guide to what we did in the workshop.

Commits of this project should line up and be in sync with the steps listed in this README.

Create The Project

Open up the Windows Command prompt (Terminal) `WINDOWS KEY` and search for `CMD`

1. Ensure you have python installed and on the path. In the terminal type:

   ```
   python --version
   ```

   Which should return something like:

   ```
   Python 3.11.8
   ```

2. Navigate to where you would like to start the project. EX:

   ```
   cd Documents
   ```

3. Create a folder for the new project and change directories to it.

   ```
   mkdir mwc3_ticket_tracker
   cd mwc3_ticket_tracker
   ```

4. Create a new virtual environment for the project and activate it.

   ```
   python -m venv .venv
   .venv\Scripts\activate.bat
   ```

5. Update pip if needed

```
python -m pip install --upgrade pip
```

6. Install the required packages. NOTE: django must be version 4.2

```
python -m pip install django~=4.2
python -m pip install django-adminlte2-pdq
```

7. Dump the requirements so that the packages can easily be installed again later if needed.

```
python -m pip freeze > requirements.txt
```

8. Start the new Django project. NOTE: Do NOT forget the period on the end.

```
django-admin startproject django_project .
```

9. Verify that the install worked by running the test server and pulling up the default landing page for a Django site.

```
python manage.py runserver
```

10. Then in a browser visit `http://localhost:8000` and verify the site came up. You should see a rocket ship.

    You can stop the server by pressing `CTRL-BREAK` or `CTRL-C` If using `CTRL-C`, you sometimes have to press it a couple of times.

11. Every Django project comes with some default database tables that can be used for User Authentication. In order to utilize that, we need to have Django create the database tables for us. This is called migrating.

```
python manage.py migrate
```

12. The migration process will take python code and use it to generate a database for us. The database will be a simple SQLite, file based database, located in the root of our project called `db.sqlite`.

Open The Project Code And Set Up AdminLTE2-PDQ

1. Open the project in VSCode. If VSCode is added to the path, you can just type the following on the command line to open it. If it is not, you need to manually open VSCode and then open the correct folder.

```
code .
```

2. Follow the Quickstart instructions for Django-AdminLTE2-PDQ located at the following link: https://django-adminlte2-pdq.readthedocs.io/en/latest/quickstart.html

3. Re-run the development server and verify that AdminLTE2-PDQ is working.

```
python manage.py runserver
```

4. Once that is confirmed to be working, we can add some customization to how AdminLTE2-PDQ behaves by editing some lines in the `settings.py` file of the `django_project` folder.

`django_project/settings.py`

```
# AdminLTE2 settings
ADMINLTE2_USE_LOGIN_REQUIRED = True
ADMINLTE2_LOGO_TEXT = "Ticket Tracking"
ADMINLTE2_LOGO_TEXT_SMALL = "TT"

ADMINLTE2_INCLUDE_ADMIN_HOME_LINK = True
ADMINLTE2_INCLUDE_MAIN_NAV_ON_ADMIN_PAGES = True
ADMINLTE2_INCLUDE_ADMIN_NAV_ON_MAIN_PAGES = True
```

5. Since we now turned on Login Required, it will be impossible to get to the site without a user. So, let's make a superuser that is able to login to the site. In the terminal:

```
python manage.py createsuperuser
```

6. Re-run the development server again. Verify that you can get to the login page and that you can log in as the newly created superuser.

```
python manage.py runserver
```

7. In the browser, Visit: `http://localhost:8000/` which should redirect to `http://localhost:8000/accounts/login/`. Then login which should then take you to the main Dashboard.

Great! We now have the stock adminLTE2-pdq site up and running. Time to start making our actual ticket tracking app.

Create Django App for Ticket Tracking

1. Apps are small plugable parts of a Django project. There will always be one single project, but potentially many apps. We will only have one app. But there could be more. Create the one new app that we plan to use. In the terminal:

   ```
   python manage.py startapp tickets
   ```

2. Register the new app in the `INSTALLED_APPS` list of the Django project settings `settings.py`

   `django_project/settings.py`

   ```
   INSTALLED_APPS = [
       "tickets.apps.TicketsConfig",
       "adminlte2_pdq",
       "django.contrib.admin",
       "django.contrib.auth",
       "django.contrib.contenttypes",
       "django.contrib.sessions",
       "django.contrib.messages",
       "django.contrib.staticfiles",
   ]
   ```

Create Database Models

1. Create the required database model in the app's `models.py` file

`tickets/models.py`

```python
from django.db import models

# Choices for Priority
PRIORITY_CHOICES = (
    (1, "High"),
    (2, "Medium"),
    (3, "Low"),
)


class Ticket(models.Model):
    """Represents a ticket for project."""

    assigned_to = models.ForeignKey(
        "auth.User",
        on_delete=models.CASCADE,
        related_name="tickets",
        blank=True,
        null=True,
    )

    name = models.CharField(max_length=250)
    description = models.TextField()
    priority = models.IntegerField(choices=PRIORITY_CHOICES)
    complete_by = models.DateField(blank=True, null=True)
    completed = models.BooleanField(default=False)

    def __str__(self):
        """str conversion"""
        return self.name
```

2. Now that we have our model created, we need to migrate that work into the database. It is a two step process. 1. Create a migration that can do the DB work for us. 2. Apply the migration to actually do that DB work. In the terminal:

```
python manage.py makemigrtaions tickets
python manage.py migrate
```

3. Django comes with a very powerful part of the site that lets certain users interact with the database. The only requirement is that the programmer register the models with the admin.

   `tickets/admin.py`

   ```
   from django.contrib import admin

   from .models import Ticket

   admin.register(Ticket)
   ```

4. In the browser navigate to the following URL and see what is available for interacting with our new model. `http://localhost:8000/admin/`

Create Ticket URL Endpoints

1. Create a `urls.py` file to store the end points for our app.

   Right click on the `tickets` folder and select `new file`. Type `urls.py` for the name of the new file and press `enter`.

2. Register the new `urls.py` file with the projects `urls.py` file.

   `django_project/urls.py`

   ```
   urlpatterns = [
       path("accounts/", include("django.contrib.auth.urls")),
       path("admin/", admin.site.urls),
       path("", include("tickets.urls")),
       path("", include("adminlte2_pdq.urls")),
   ]
   ```

3. Create the first URL endpoint for the app that will show all of the open tickets.

   NOTE: We have not created the OpenTicketListView class yet. As a result, the import may have issues for now but should be fixed in one of the next steps.

`tickets/urls.py`

```python
from django.urls import path

from .views import (
    OpenTicketListView,
)

urlpatterns = [
    path("", OpenTicketListView.as_view(), name="open_ticket_list"),
]
```

4. Update `settings.py` and set the AdminLTE2-PDQ home route to the newly created URL endpoint.

`django_project/settings.py`

```python
# AdminLTE2 settings
ADMINLTE2_HOME_ROUTE = "open_ticket_list"
```

Create Open Ticket List View For First URL Endpoint

1. Update the `views.py` file to contain our first view for the url endpoint we set up in the previous step.

   NOTE: Some of the import statements will not be used right now. But we are adding them now because we will need them later on.

   `tickets/views.py`

```python
from django.contrib import messages
from django.views.generic import ListView, TemplateView
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from django.urls import reverse

from .models import Ticket


class OpenTicketListView(TemplateView):
    """Show lists of open tickets by priority"""

    template_name = "ticket_list_open.html"

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.extra_context = {
            "open_high_tickets": Ticket.objects.filter(priority=1,
completed=False),
            "open_medium_tickets": Ticket.objects.filter(priority=2,
completed=False),
            "open_low_tickets": Ticket.objects.filter(priority=3,
completed=False),
        }
```

Create Folder To Store App Templates

1. In the VSCode sidebar, right click below the file tree and select `New Folder`. Name the new folder `templates`. This should put the new folder into the root of the project. It needs to be in the root of the project to work correctly.

2. Update the `settings.py` file to tell Django to look in our new `templates` directory for our templates.

   You just need to find the setting called `TEMPLATES` in `settings.py` and update it to contain a new directory in the `DIRS` part of that setting.

`django_project/settings.py`

Change

```
"DIRS": [],
```

To

```
"DIRS": [BASE_DIR / "templates"]
```

The full altered setting should look like the below when done.

`django_project/settings.py`

```
TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [BASE_DIR / "templates"],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ],
        },
    },
]
```

Create Open Ticket List View Template And Partial Templates

1. Create all of the template files that we will need for the whole app. Create all of them right now so we are done with it and don't have to keep creating them as we need them.

   For each of the filenames listed below, right click on the `templates` directory and choose `New File` and then create each file.

   ```
   _ticket_box.html
   _ticket_breadcrumb.html
   _ticket_form.html
   ticket_create.html
   ticket_delete.html
   ticket_list_closed.html
   ticket_list_open.html
   ticket_update.html
   ```

2. Template partials are small snippets that are meant to be reused multiple times when creating templates. They are prefixed with an underscore to denote that they are not a full page.

   Create the `_ticket_box.html` partial.

   `templates/_ticket_box.html`

```
<div class="box box-solid box-{{ color }}">
  <div class="box-header with-border">
    <h3 class="box-title">{{ priority }} Tickets</h3>
  </div>
  <div class="box-body">
    {% if tickets.exists %}
      <table class="table">
        <tr>
          <th>Name</th>
          <th>Assigned To</th>
          <th>Complete By</th>
          <th class="text-right">Actions</th>
        </tr>
        {% for ticket in tickets.all %}
          <tr>
            <td>{{ ticket.name }}</td>
            <td>{{ ticket.assigned_to.username }}</td>
            <td>{{ ticket.complete_by }}</td>
            <td class="text-right">
              <a href="{% url "ticket_update" ticket.pk %}" class="btn btn-
success">
                <i class="fa fa-pencil"></i> Update
              </a>
              <a href="{% url "ticket_delete" ticket.pk %}" class="btn btn-
danger">
                <i class="fa fa-times"></i> Delete
              </a>
            </td>
          </tr>
        {% endfor %}
      </table>
    {% else %}
      <h3>No Tickets For This Priority Level</h3>
    {% endif %}
  </div>
</div>
```

3. Create the `_ticket_breadcrumb.html` partial.

`templates/_ticket_breadcrumb.html`

```
<ol class="breadcrumb">
  <li>
    <i class="fa fa-dashboard"></i>
    <a href="{% url 'open_ticket_list' %}">Home</a>
  </li>
  {% if icon and action %}
    <li class="active"><i class="fa fa-{{ icon }}"></i> {{ action }}
Ticket</li>
  {% endif %}
</ol>
```

4. Create the `_ticket_form.html` partial.

`templates/_ticket_form.html`

```
{% load adminlte_tags %}
<div class="box box-solid box-{{ color }}">
  <div class="box-header with-border">
    <h3 class="box-title">{{ action }} Ticket</h3>
  </div>
  <div class="box-body">
    <form method="post">
      {% csrf_token %}
      {% if action == "Delete" %}
        <p>
          <h4>Are you sure you want to delete this ticket?</h4>
          - {{ ticket.name }}
        </p>
      {% else %}
        {% render_form form %}
      {% endif %}
      <button type="submit" class="btn btn-{{ color }}">
        <i class=""></i> {{ action }}
      </button>
    </form>
  </div>
</div>
```

5. Use two of the above partials to create the `ticket_list_open.html` page.

`templates/ticket_list_open.html`

```
{% extends "adminlte2/base.html" %}

{% block page_name %}
Open Tickets
{% endblock page_name %}

{% block breadcrumbs %}
  {% include "_ticket_breadcrumb.html" with icon="ticket" action="Open" %}
{% endblock breadcrumbs %}

{% block content %}

  <a href="{% url "ticket_create" %}" class="btn btn-primary pull-right">
    <i class="fa fa-plus"></i> New Ticket
  </a>

  <br><br>

  {% include "_ticket_box.html" with tickets=open_high_tickets color="danger"
priority="High Priority" %}
  {% include "_ticket_box.html" with tickets=open_medium_tickets
color="warning" priority="Medium Priority" %}
  {% include "_ticket_box.html" with tickets=open_low_tickets color="info"
priority="Low Priority" %}

{% endblock content %}
```

6. Test out the site and verify that we can see the boxes. Run the server if it is not already running and visit `http://localhost:8000`

```
python manage.py runserver
```

Create Remaining Templates Utilizing Created Partials

1. Use partials to create the `ticket_list_closed.html` page.

`templates/ticket_list_closed.html`

```
{% extends "adminlte2/base.html" %}

{% block page_name %}
Closed Tickets
{% endblock page_name %}

{% block breadcrumbs %}
  {% include "_ticket_breadcrumb.html" with icon="ticket" action="Closed" %}
{% endblock breadcrumbs %}

{% block content %}
  {% include "_ticket_box.html" with tickets=closed_tickets color="purple"
priority="Closed" %}
{% endblock content %}
```

2. Use partials to create the `ticket_create.html` page.

`templates/ticket_create.html`

```
{% extends "adminlte2/base.html" %}

{% block page_name %}
Create Ticket
{% endblock page_name %}

{% block breadcrumbs %}
  {% include "_ticket_breadcrumb.html" with icon="plus" action="Create" %}
{% endblock breadcrumbs %}

{% block content %}
  {% include "_ticket_form.html" with color="primary" action="Create" %}
{% endblock content %}
```

3. Use partials to create the `ticket_update.html` page.

`templates/ticket_update.html`

```
{% extends "adminlte2/base.html" %}

{% block page_name %}
Update Ticket
{% endblock page_name %}

{% block breadcrumbs %}
  {% include "_ticket_breadcrumb.html" with icon="pencil" action="Update" %}
{% endblock breadcrumbs %}

{% block content %}
  {% include "_ticket_form.html" with color="success" action="Update" %}
{% endblock content %}
```

4. Use partials to create the `ticket_delete.html` page.

`templates/ticket_delete.html`

```
{% extends "adminlte2/base.html" %}

{% block page_name %}
Delete Ticket
{% endblock page_name %}

{% block breadcrumbs %}
  {% include "_ticket_breadcrumb.html" with icon="times" action="Delete" %}
{% endblock breadcrumbs %}

{% block content %}
  {% include "_ticket_form.html" with color="danger" action="Delete" %}
{% endblock content %}
```

Create Remaining Views For The Templates We Just Created

1. Update the `views.py` file by appending the following Views that will complete our site.

   NOTE: By the end of this, all import statements should be used.

   `tickets/views.py`

```python
class ClosedTicketListView(ListView):
    """Show list of closed tickets"""

    model = Ticket
    template_name = "ticket_list_closed.html"

    def get_queryset(self):
        return Ticket.objects.filter(completed=True)


class TicketCreateView(CreateView):
    """Create a new ticket"""

    model = Ticket
    template_name = "ticket_create.html"
    fields = "__all__"

    def get_success_url(self):
        """Where to send user on success"""
        messages.success(self.request, "Ticket Created Successfully")
        return reverse("open_ticket_list")


class TicketUpdateView(UpdateView):
    """Update a ticket"""

    model = Ticket
    template_name = "ticket_update.html"
    fields = "__all__"

    def get_success_url(self):
        """Where to send user on success"""
        messages.success(self.request, "Ticket Updated Successfully")
        return reverse("open_ticket_list")
```

```python
class TicketDeleteView(DeleteView):
    """Delete a ticket"""

    model = Ticket
    template_name = "ticket_delete.html"

    def get_success_url(self):
        """Where to send user on success"""
        messages.success(self.request, "Ticket Deleted Successfully")
        return reverse("open_ticket_list")
```

Create the Remaining URLs For The Views We Just Created

1. Import the views that we just made and then create the remaining URL endpoints needed to complete the app.

   NOTE: The below code is what the finished file looks like.

   `tickets/urls.py`

```python
from django.urls import path

from .views import (
    OpenTicketListView,
    ClosedTicketListView,
    TicketCreateView,
    TicketUpdateView,
    TicketDeleteView,
)

urlpatterns = [
    path("", OpenTicketListView.as_view(), name="open_ticket_list"),
    path("tickets/closed/", ClosedTicketListView.as_view(),
name="closed_ticket_list"),
    path("tickets/create/", TicketCreateView.as_view(), name="ticket_create"),
    path("tickets/<int:pk>/update/", TicketUpdateView.as_view(),
name="ticket_update"),
    path("tickets/<int:pk>/delete/", TicketDeleteView.as_view(),
name="ticket_delete"),
]
```

Update The AdminLTE2-PDQ Menu To Contain Links For Our Site

1. We now need to tell AdminLTE2-PDQ what URL route we want to be our home page. Additionally, we want to define the sidebar menu. Both things are done by creating a setting in the `settings.py` file. Append the following lines to the bottom of the `settings.py` file.

`django_project/settings.py`

```python
# AdminLTE2 menu
ADMINLTE2_MENU = [
    {
        "text": "Home",
        "nodes": [
            {
                "route": "open_ticket_list",
                "text": "Open Tickets",
                "icon": "fa fa-dashboard",
                "active_requires_exact_url_match": True,
            },
            {
                "route": "closed_ticket_list",
                "text": "Closed Tickets",
                "icon": "fa fa-ticket",
            },
        ],
    },
    {
        "text": "Profile",
        "nodes": [
            {
                "route": "password_change",
                "text": "Change Password",
                "icon": "fa fa-lock",
            },
        ],
    },
]
```

Test Out The Entire Site

Assuming everything went well, you should not be able to run the server again and verify that everything works as intended.

```
python manage.py runserver
```

In a browser visit:

```
http://localhost:8000
```