

Lab 3 – Databases & Django Admin

In this exercise we will use a database for the first time to build a basic Message Board application where users can post and read short messages. We'll explore Django's very useful built-in admin interface which provides a visual way to make changes to our data.

Django provides built-in support for various types of database backends. With just a few lines in our **settings.py** file it can support PostgreSQL, MySQL, Oracle, or SQLite. But the easiest one to use is SQLite because it runs off a single file and requires no complex installation. Django uses SQLite by default for this reason and it is a perfect choice for our small projects.

Initial Set Up

Go to www.github.com and log in to your GitHub account

Go to Moodle and when you find the link shown below, click on it.



Lab 3 GitHub Classroom Upload

Next you should see a screen like the following asking you to accept the assignment. Click the green button to accept the assignment.

TUD-SDEV-classroom-SDEV3-2023

Accept the assignment —

Lab 3 Message Board

Once you accept this assignment, you will be granted access to the `lab-3-message-board-pmagee` repository in the **TUD-SDEV** organization on GitHub.


Accept this assignment

Once you have accepted the assignment you are asked to refresh the page, and you are then presented with a link to your repository for lab 3 as shown below:


You're ready to go!

You accepted the assignment, **Lab 3 Message Board** .

Your assignment repository has been created:

 <https://github.com/TUD-SDEV/lab-3-message-board-pmagee>

We've configured the repository associated with this assignment ([update](#)).


 Your assignment is due by **Oct 2, 2023, 18:00 IST**


Note: You may receive an email invitation to join [TUD-SDEV](#) on your behalf. No further action is necessary.


When you click on the repository link, you are taken to the repository where you see a readme file has already been added:


lab-3-message-board-pmagee Private Edit Pins Unwatch 1

main 1 branch 0 tags Go to file Add file Code

 github-classroom[bot] Add assignment deadline url aba1146 1 minute ago 1 commit

 **README.md** Add assignment deadline url 1 minute ago

README.md 

 Review the assignment due date

In Windows Command line make sure that you are in the **django** projects folder and use the following command to activate the virtual environment:

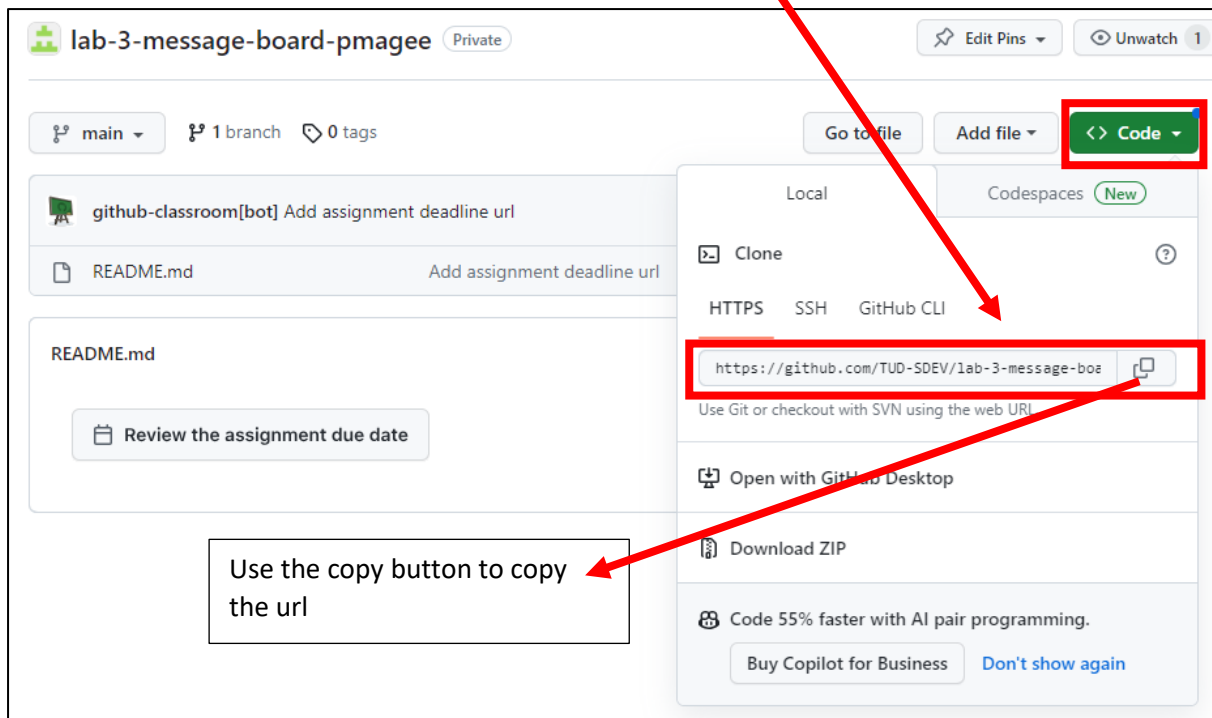
```
env\scripts\activate.bat
```

Use the following command to clone this repo to your local computer:

```
git clone <repo name>
```

Replace this part with your
repo name.

You can get the repo details by clicking on the Code button as shown below:



After the clone command is finished you have a new folder called **lab-3-messageboard-<username>** inside the **django projects** folder.

Move into this new folder using the **cd** command.

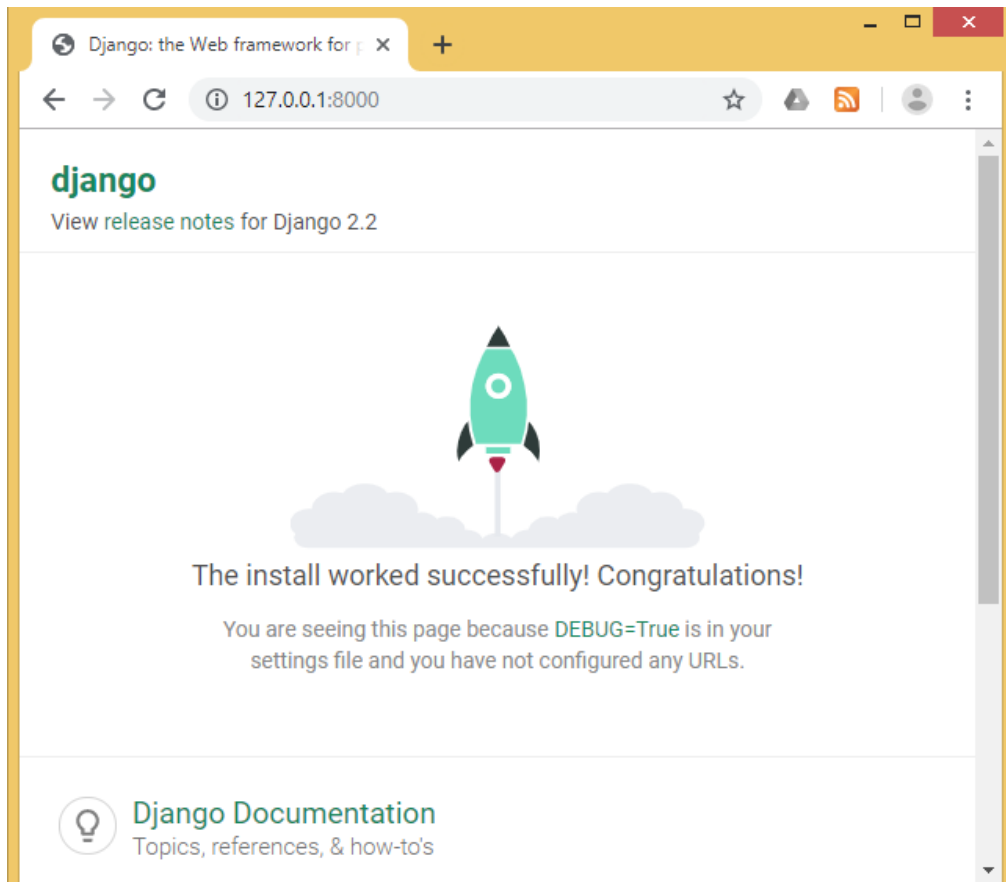
Create a new Django project called **messageboard** with the following command. Don't forget the period (fullstop) at the end:

```
django-admin startproject messageboard .
```

You can verify that the Django project works by typing in the following command:

```
python manage.py runserver 8080
```

If you visit <http://127.0.0.1:8080/> you should see the familiar Django welcome page



The output in the command line shows a warning about “18 unapplied migrations” although this warning has no effect on the project at this point. Django is letting us know that we have not yet “migrated” or configured our initial database. Since we don’t use a database in this exercise, the warning won’t affect the result.

Stop the server using **Ctrl+c** and remove the warning by running the migrate command as shown here:

```
python manage.py migrate
```

When you run this command, you will see in the output that 18 migrations are applied. We will look at the meaning of these migrations later.

If you execute the **python manage.py runserver 8080** again, the warning message is gone.

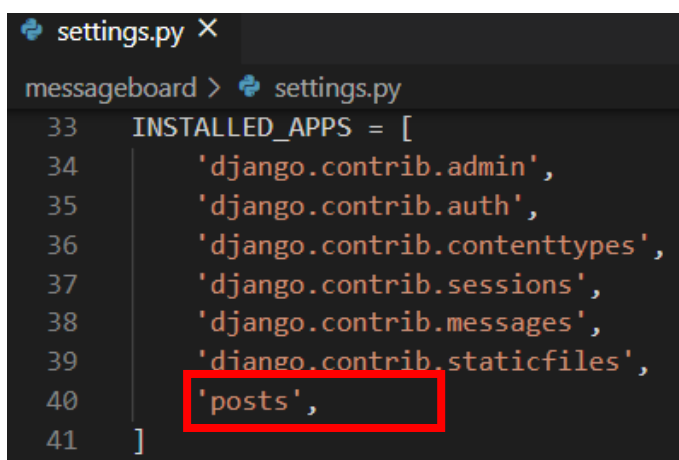
From the command line, quit the server with **Ctrl+c**. Then use the **startapp** command as shown below to create a new app called **posts**:

```
python manage.py startapp posts
```

Settings.py

1. In **VS Code**, open the **lab3** folder, locate and open **settings.py** and register your new app as shown below:

Code



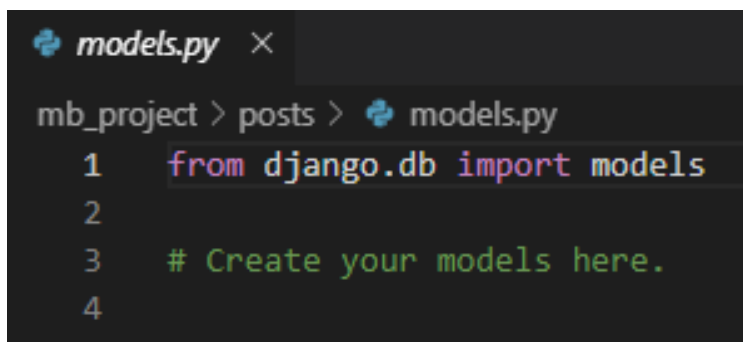
```
settings.py ×
messageboard > settings.py
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'posts',
41 ]
```

Create a database model

Our first task is to create a database model where we can store and display posts from our users. Django will turn this model into a database table for us. In real-world Django projects, it is often the case that there will be many complex interconnected database models but in our simple message board app we only need one.

Open the **posts/models.py** file and look at the default code which Django provides:

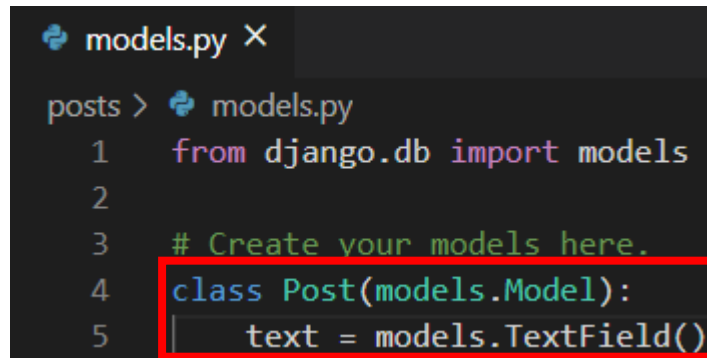
Code



```
models.py ×
mb_project > posts > models.py
1  from django.db import models
2
3  # Create your models here.
4
```

Django imports a module called **models** to help us build new database models, which will “model” the characteristics of the data in our database. To create a model to store the textual content of a message board post, add the code highlighted below:

Code



```
models.py X
posts > models.py
1  from django.db import models
2
3  # Create your models here.
4  class Post(models.Model):
5      text = models.TextField()
```

We have just created a new database model called **Post** which has the database field `text`. We have also specified the type of content it will hold, `TextField()`. Django provides many model fields supporting common types of content such as characters, dates, integers, emails, and so on.

Activating models

When we have the new model, we need to activate it. In future, whenever we create or modify an existing model, we will need to update Django in a two-step process.

1. We create a migration file with the **makemigrations** command which creates the migrations file that instructs Django on how to create the database tables for the models defined in project. Migration files do not execute those commands on our database file, rather they are a reference of all new changes to our models. This approach means that we have a record of the changes to our models over time.
2. We build the actual database with the **migrate** command which executes the instructions in our migrations file.

Make sure the local server is stopped using **Ctrl+c** and then run the following two commands:

python manage.py makemigrations posts

After you run this command, you should see the following output:

```
Migrations for 'posts':
  posts\migrations\0001_initial.py
    - Create model Post
```

Next run the following command:

python manage.py migrate

After you run this command, you should see the following output:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, posts, sessions
Running migrations:
  Applying posts.0001_initial... OK
```

The command **makemigrations** generates a set of SQL commands which is stored in a file called **001_initial.py** which is in the migrations folder. The **migrate** command then executes the SQL commands in this file which creates a table in our database called **Posts**.

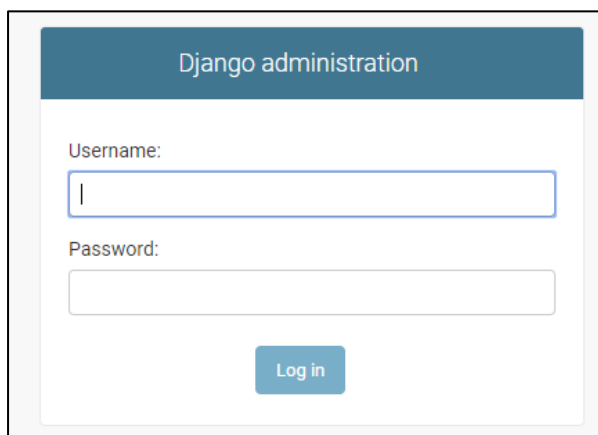
Django Admin

Django provides us with a very useful admin interface for interacting with our database. This is a very powerful tool that few web frameworks offer.

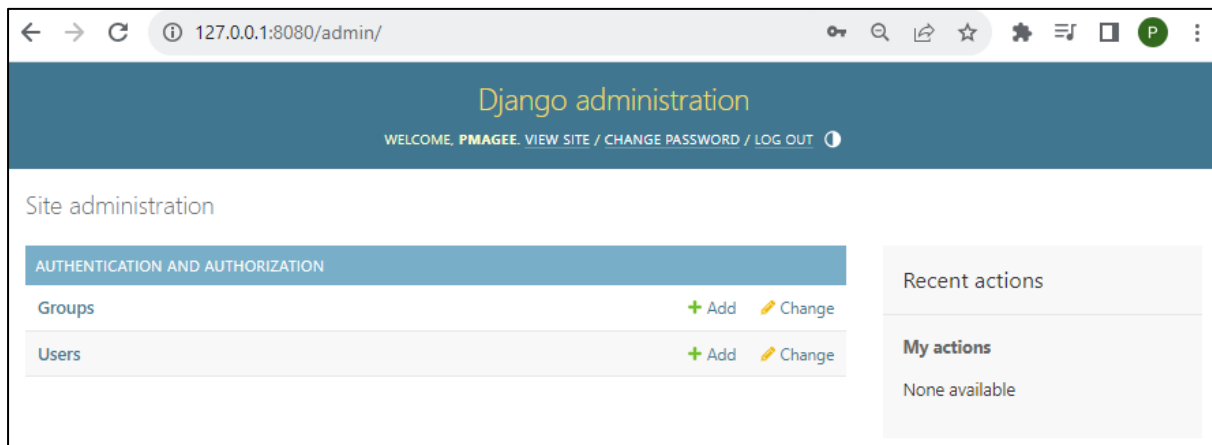
To use the Django admin, we first need to create a superuser who can log in. In **Windows Command Line**, type the command shown below and respond to the prompts for a username, email, and password. Note that the password does not display as you type it.

python manage.py createsuperuser

Restart the Django server with **python manage.py runserver 8080** and in your web browser go to **http://localhost:8080/admin/**. (Don't forget to type admin in the url). You should see the admin's log in screen:

A screenshot of the Django administration login interface. It features a dark blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". A blue "Log in" button is positioned below the password field.

Log in by entering the username and password you just created. You will see the Django admin homepage next:

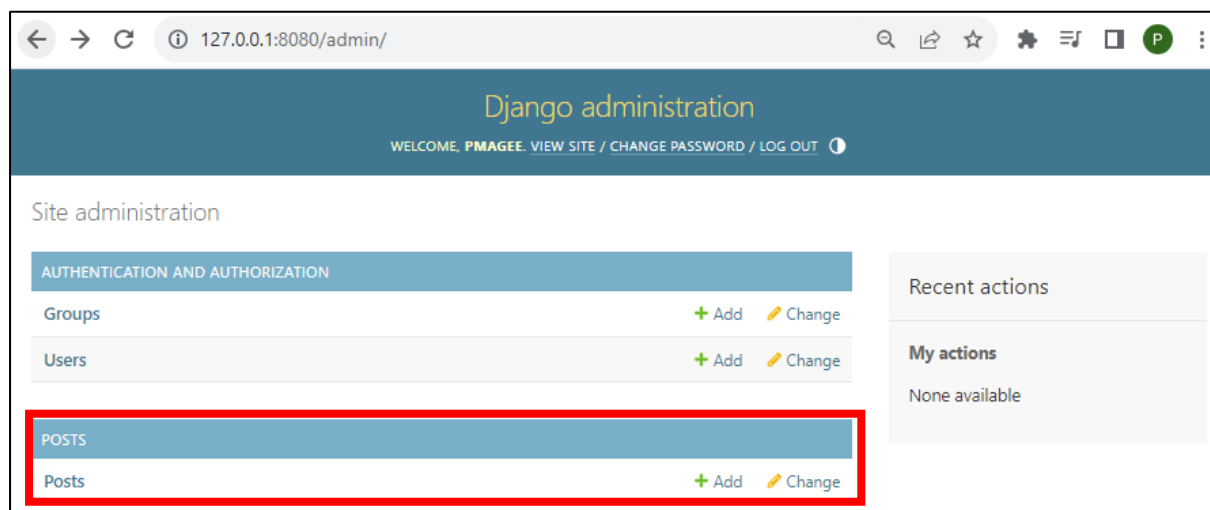
A screenshot of the Django administration homepage in a web browser. The browser's address bar shows "127.0.0.1:8080/admin/". The page has a dark blue header with "Django administration" in yellow and "WELCOME, PMAGEE. VIEW SITE / CHANGE PASSWORD / LOG OUT" in white. The main content area is titled "Site administration" and contains a table with two rows: "Groups" and "Users". Each row has "+ Add" and "Change" links. To the right of the table, there are two sections: "Recent actions" and "My actions", both showing "None available".

There is no sign of our **posts** app on the main admin page. We need to explicitly tell Django what to display in the admin. In VS Code open **posts/admin.py** and add the following code:

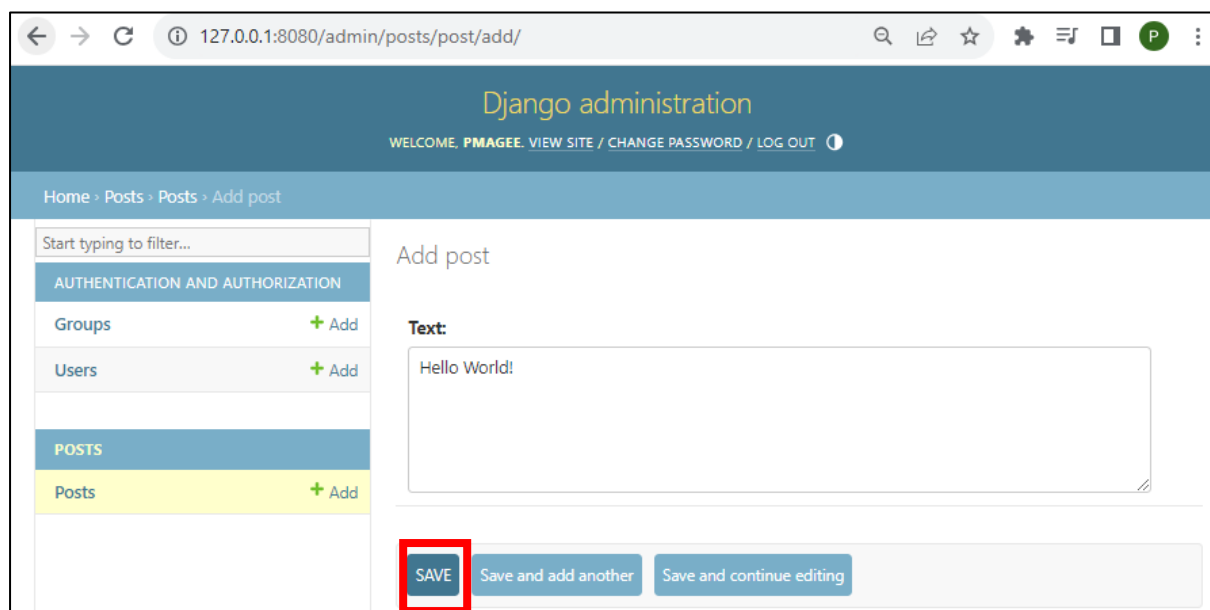
Code

```
posts > admin.py
1  from django.contrib import admin
2  from .models import Post
3  # Register your models here.
4
5  admin.site.register(Post)
```

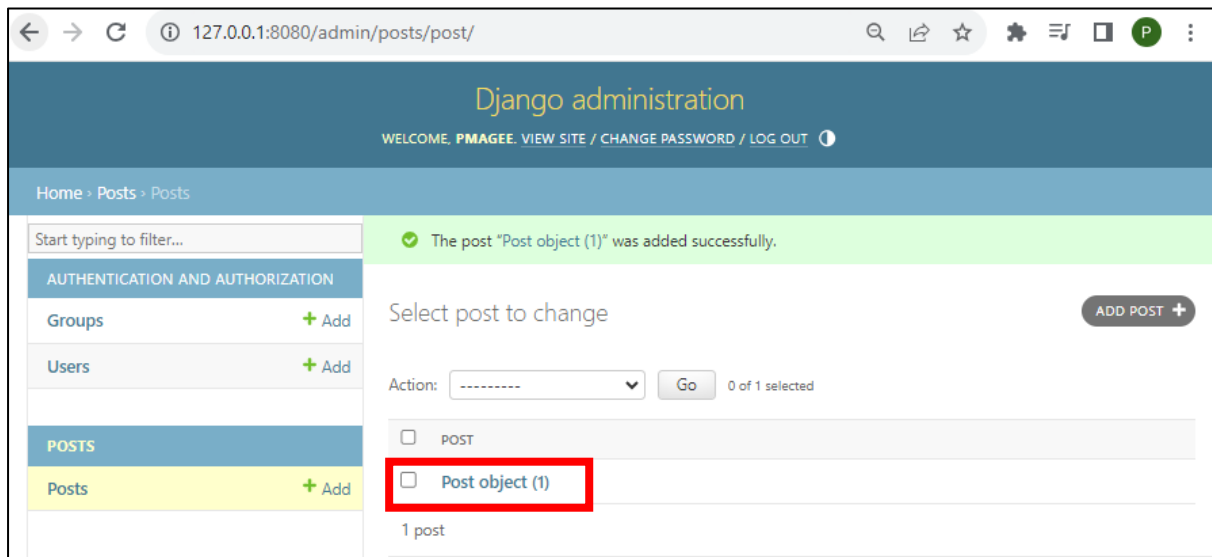
Django now knows that it should display our posts app and its database model **Post** on the admin page. If you refresh your browser, you will see that it appears now:



Now we can create our first message board post for our database. Click on the + Add button opposite Posts. Enter your own text in the Text form field.



Then click the **“Save”** button, which will redirect you to the main **Posts** page. Here you will see the name **“Post object (1)”**, which isn’t very user friendly.

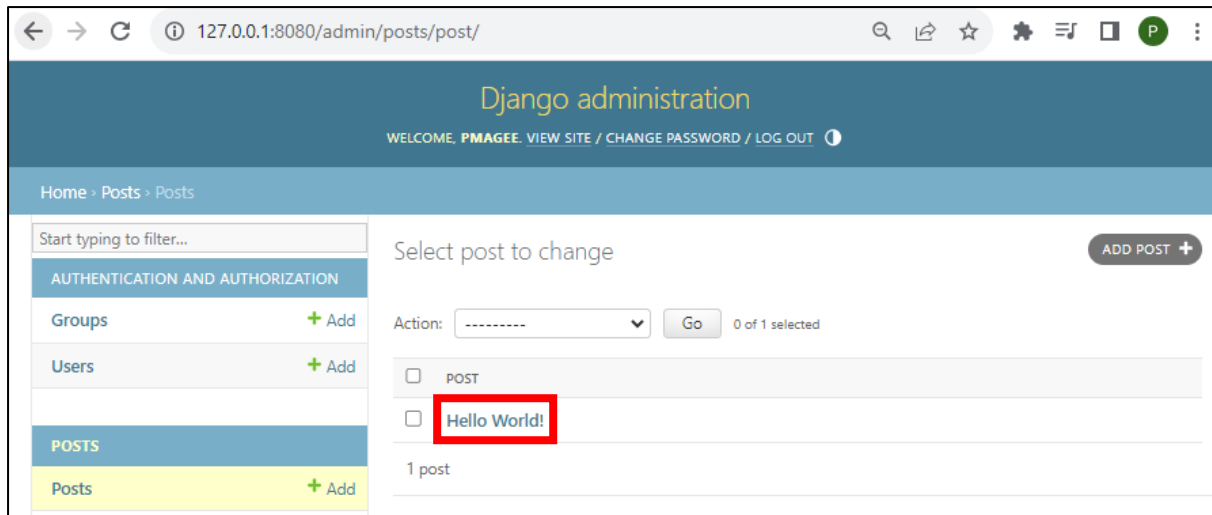


We can change the name that is displayed by writing a new function in **posts/models.py** called **__str__**. Add the following code:

Code

```
models.py ×
posts > models.py
1  from django.db import models
2
3  # Create your models here.
4  class Post(models.Model):
5      text = models.TextField()
6
7      def __str__(self):
8          return self.text[:50]
```

This will display the first 50 characters of the text field. If you refresh your Admin page in the browser, you will see it has changed to a much more descriptive and helpful representation of our database entry.



It is best practice to add **str()** methods to all your models to improve their readability.

Views/Templates/URLs

To display our database content on our homepage, we need to set up our views, templates, and URLConfs.

We will start with the view. In the previous exercise we used the built-in generic **Template- View** to display a template file on our homepage. Now we want to list the contents of our database model. This is a very common task in web development and Django comes equipped with the generic class-based **ListView**.

In **VS Code**, move into the **posts/views.py** file delete the top line of code and enter the Python code below:

Code

```
views.py ×
mb_project > posts > views.py
1  # posts/views.py
2  from django.views.generic import ListView
3  from .models import Post
4
5  class HomePageView(ListView):
6      model = Post
7      template_name = 'home.html'
```

On the first line we are importing `ListView` and on the second line we need to explicitly define which model we are using. In the view, we subclass **`ListView`**, specify our model name and specify our template reference. Internally **`ListView`** returns an object called **`object_list`** that we want to display in our template.

Our view is complete which means we still need to configure our URLs and make the template. Let's start with the template.

In **Windows command line**, stop the server and create a new folder called **templates**:

```
mkdir templates
```

In **VS Code**, move into the **templates** folder and create a new file called **home.html**:

Update the **`DIRS`** field in our **`settings.py`** file so that Django knows to look in this **templates** directory.

Code

```
55  TEMPLATES = [  
56      {  
57          'BACKEND': 'django.template.backends.django.DjangoTemplates',  
58          'DIRS': [str(BASE_DIR.joinpath('templates'))],  
59          'APP_DIRS': True,  
60          'OPTIONS': {  
61              'context_processors': [  
62                  'django.template.context_processors.debug',  
63                  'django.template.context_processors.request',  
64                  'django.contrib.auth.context_processors.auth',  
65                  'django.contrib.messages.context_processors.messages',  
66              ],  
67          },  
68      },  
69  ]
```

In our templates file **home.html** we can use the Django Templating Language's **for loop** to list all the objects in **`object_list`**. This is the name of the variable that **`ListView`**

returns to us. We should specify not just the object we want which is post, but the specific field we want to display which is text - we will use **post.text**.

In **VS Code** open the file **home.html** and enter the following code:

Code

```
templates > <> home.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <title>Home Page</title>
6  </head>
7
8  <body>
9      <h1>Message Board Homepage</h1>
10     <ul>
11         {% for post in object_list %}
12         <li>{{post.text}}</li>
13         {% endfor %}
14     </ul>
15 </body>
```

- The code in this HTML **template** file will be called by the Django function **render()**. You must write the python code in between `{% %}` so that Django recognizes that it is **Python** code
- To display a variable in the HTML template, you must use `{{ }}`. You must close the **for loop** by writing **endfor**

However, **object_list** is not a very user-friendly name so instead we will manually set the name of the context variable. The `context_object_name` attribute on a generic view specifies the context variable to use. Adding an explicit name in this way makes it easier for other members of a team, for example a designer, to understand and reason about what is available in the template context.

Add the following line of code to **posts/views.py**:

Code

```
views.py X
posts > views.py
1  from django.views.generic import ListView
2  from .models import Post
3
4  # Create your views here.
5  class HomePageView(ListView):
6      model = Post
7      template_name = 'home.html'
8      context_object_name = "all_posts_list"
```

We now need to update our template, to use this new name. In **home.html**, change the code in the for loop to that shown below:

Code

```
templates > home.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5  |   <title>Home Page</title>
6  </head>
7
8  <body>
9  |   <h1>Message Board Homepage</h1>
10 |   <ul>
11 |       {% for post in all_posts_list %}
12 |       <li>{{post.text}}</li>
13 |       {% endfor %}
14 |   </ul>
15 </body>
```

The last step is to set up our URLConfs. Open the **messageboard/urls.py** file and add the code shown below:

Code

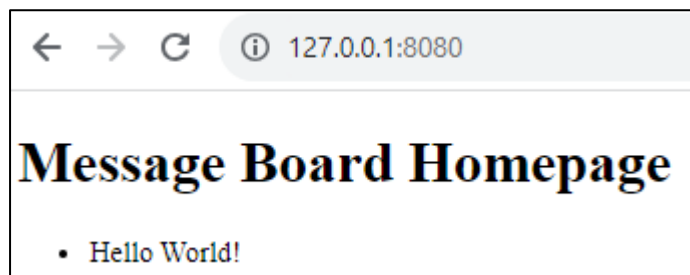
```
urls.py ×
messageboard > urls.py
16  from django.contrib import admin
17  from django.urls import path, include
18
19  urlpatterns = [
20  |   path('admin/', admin.site.urls),
21  |   path('', include('posts.urls')),
22  ]
```

Then in **VS Code** create a file called **urls.py** in the **posts** folder and update it with the following code:

Code

```
urls.py
posts > urls.py
1  from django.urls import path
2  from .views import HomePageView
3
4  urlpatterns=[
5      path('', HomePageView.as_view(), name='home'),
6  ]
```

Restart the server with `python manage.py runserver 8080` and navigate to our homepage `http://127.0.0.1:8080/` which now lists out our message board posts.

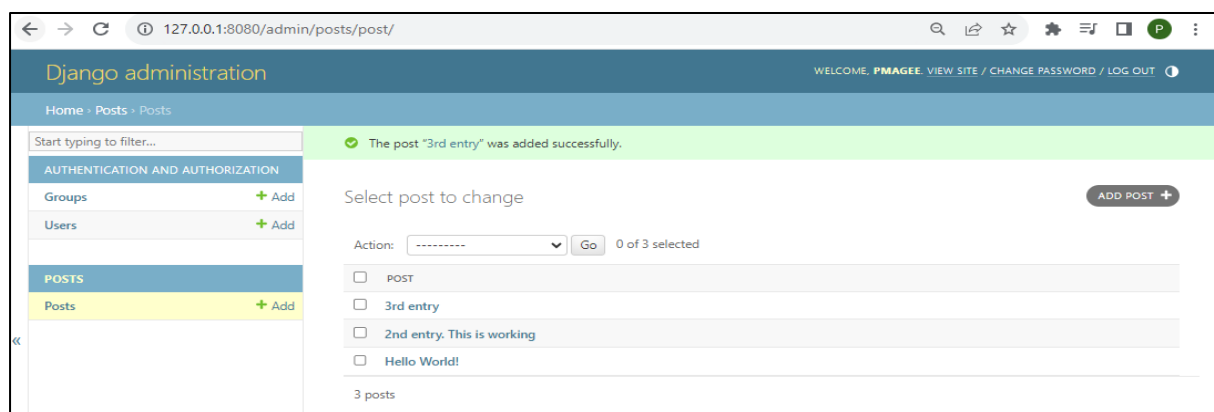


Adding new posts

Create a few more message board posts in the Django admin to confirm that they will display correctly on the homepage. To add new posts to our message board, go back into the Django Admin:

`http://127.0.0.1:8080/admin/`

Create two more posts like those shown below:



If you refresh the homepage, you will see that it automatically displays our formatted posts:



Open the **README.md** file in Vs Code and replace the contents with your name and id number.

Save a snapshot of the current project state with the following command:

git add -A

Commit the changes along with a suitable message:

git commit -m "lab 3 commit"

Update the remote repository with the local commits:

git push -u origin main

Go to your GitHub page and refresh the page to see your local code now hosted online.

To deactivate the virtual environment type, deactivate as shown below:

deactivate

To exit out of Windows Command Line type exit:

exit