**Lab 3**

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**

Open **PowerShell**, move into the directory called **djangoprojects** and create a new Django project called **lab3**

mkdir lab3

Move into the **lab3** directory and run the following command to create a virtual environment

virtualenv env

You should see output indicating that the virtual environment has been created. We now need to activate the virtual environment by typing the following command:

env\scripts\activate

Note : If you get an error at this point telling you that the file activate1.ps cannot be loaded then you need to close PowerShell and run it as an administrator. Then run the following command in order to allow scripts to execute:

set-executionpolicy remotesigned

When you run this command, you are given a few options to select from. Type A and hit enter

Try the activate command again:

env\scripts\activate

You should now see parentheses around the name of your current directory on your command line which indicates the virtual environment is activated:

(env) djangoprojects\lab3>

Install Django using the following command:

(env) djangoprojects\lab3>pip install django

When Django is installed, you may get a warning suggesting that you upgrade to the latest version of pip. If so, type the following command to upgrade pip:

(env) djangoprojects\ lab3>python -m pip install –upgrade pip

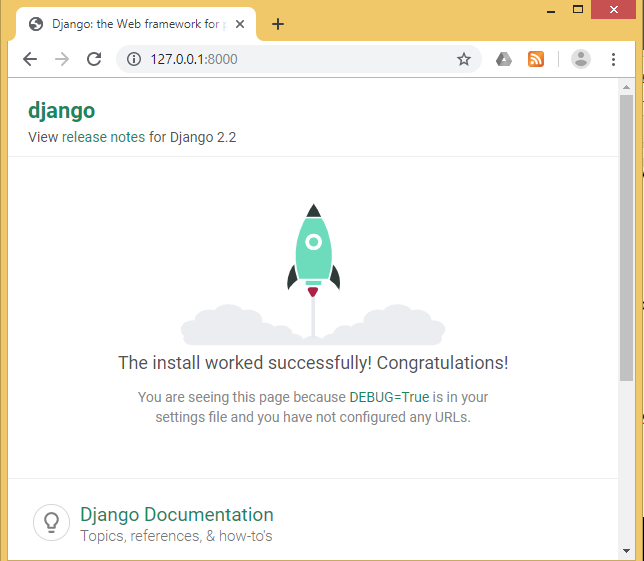
Create a new Django project called **messageboard** with the following command. Don’t forget the period (fullstop) at the end:

(env) djangoprojects\ lab3>django-admin startproject messageboard .

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

(env) djangoprojects\ lab3 >python manage.py runserver

If you visit <http://127.0.0.1:8000/> 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.

You can remove the warning by running the migrate command as shown here:

(env) djangoprojects\ lab3 >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 at a later time.

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

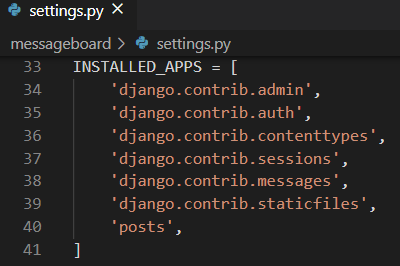
Create an app called posts. From the command line, quit the server with Control+BREAK. Then use the startapp command as shown below:

(env) djangoprojects\ lab3 >python manage.py startapp posts

**Settings.py**

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

**Code**



To confirm everything works correctly, launch the local server & and navigate to http://127.0.0.1:8000/ to see the familiar Django rocket page.

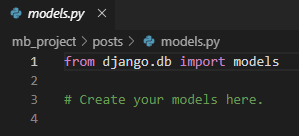
(env) djangoprojects\ lab3 >python manage.py runserver

**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.

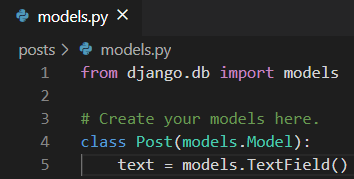
Open the **posts/models.py** file and look at the default code which Django provides:

**Code**



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

**Code**



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**

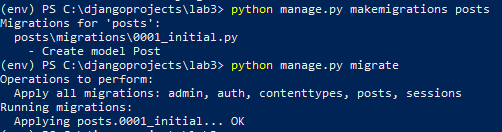
When we have the new model, we need to activate it. In future, whenever we create or modify an existing model, we will need to update Django in a two-step process.

1. We create a migration file with the **makemigrations** command which generate the SQL commands for preinstalled apps in our INSTALLED\_APPS setting. 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 Control+BREAK and then run the following two commands:

**Command Line**



The command **makemigrations** generates a set of SQL commands which is stored in a file called **001\_initial.py** which is located 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**

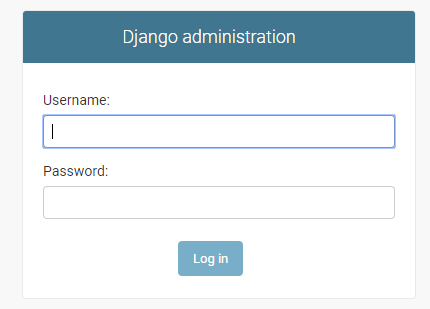
Django provides us with a very useful admin interface for interacting with our database. This is a very powerful tool that few web frameworks offer.

To use the Django admin, we first need to create a superuser who can log in. In PowerShell, type the command shown below and respond to the prompts for a username, email, and password:

(env) djangoprojects\ lab3 >python manage.py createsuperuser

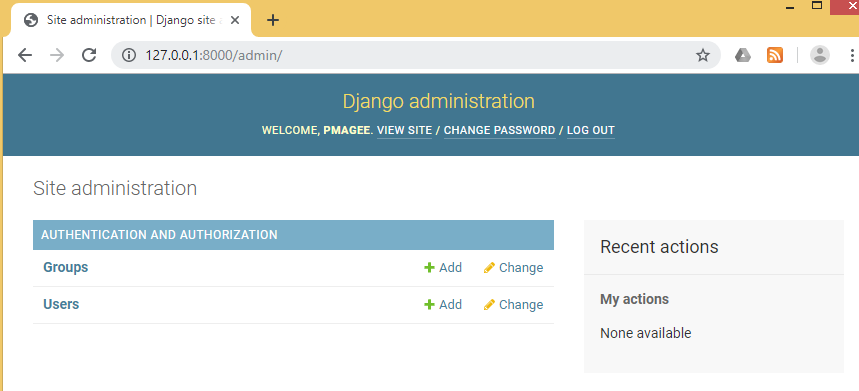
Restart the Django server with python manage.py runserver and in your web browser

go to http://127.0.0.1:8000/admin/. (**Don’t forget to type admin in the url**). You should see the admin’s log in screen:



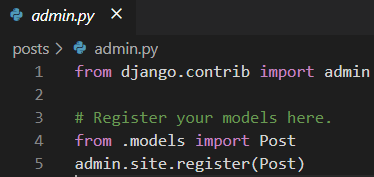
Log in by entering the username and password you just created. You will see the

Django admin homepage next:



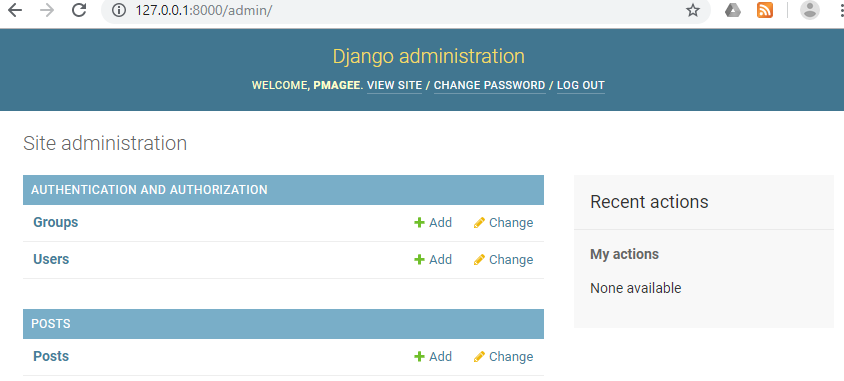
There is no sign of our **posts** app on the main admin page. We need to explicitly tell Django what to display in the admin. In VS Code open **posts/admin.py** and add the following code:

**Code**



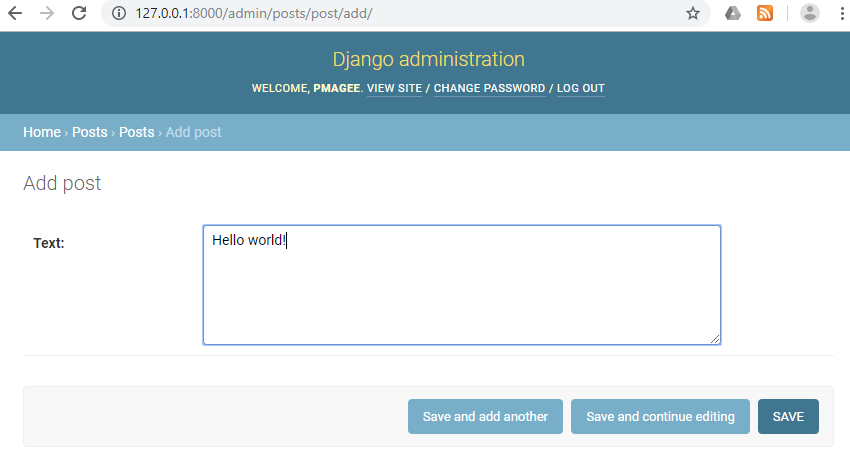
Django now knows that it should display our posts app and it’s database model **Post**

on the admin page. If you refresh your browser, you will see that it appears now:

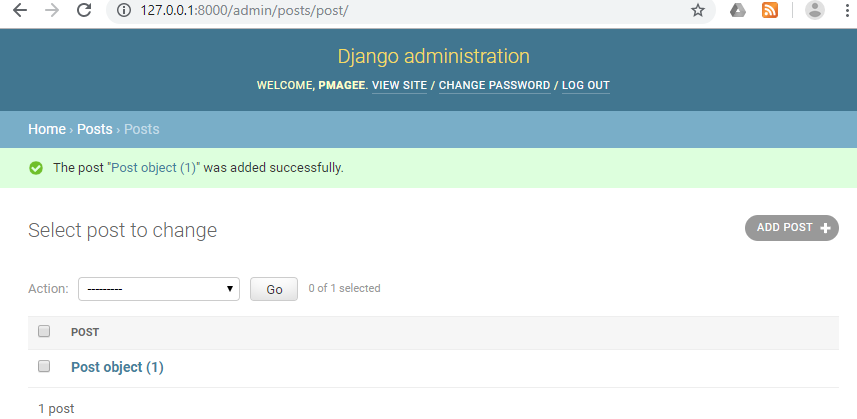


Now we can create our first message board post for our database. Click on the + Add

button opposite Posts. Enter your own text in the Text form field.

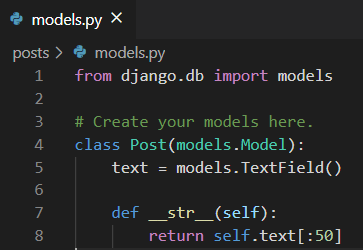


Then click the “**Save**” button, which will redirect you to the main **Posts** page. Here you will see the name “**Post object (1)**”, which isn’t very user friendly.

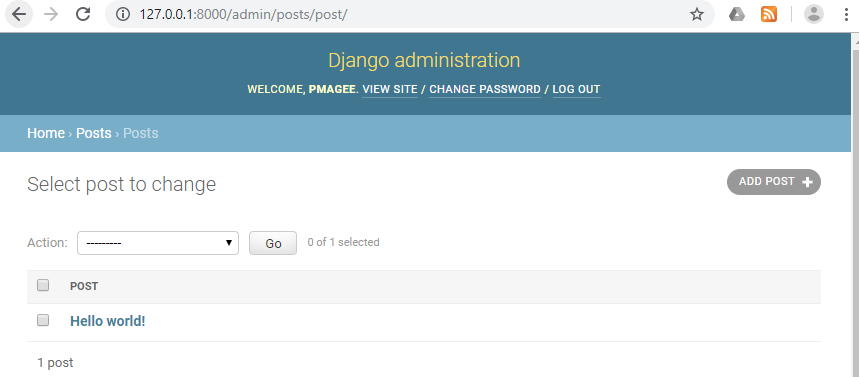


We can change the name that is displayed by writing a new function in **posts/models.py** called **\_\_str\_\_** as follows:

**Code**



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 of your models to improve their readability.

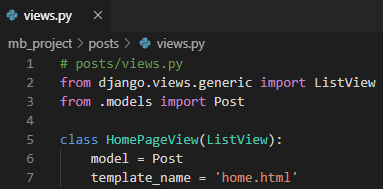
**Views/Templates/URLs**

In order to display our database content on our homepage, we need to set up our views, templates, and URLConfs. This process should start to feel familiar now.

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 the **posts/views.py** file delete the top line of code and enter the Python code below:

**Code**



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 PowerShell create a new folder called **templates:**

(env) djangoprojects\ lab3 >mkdir templates

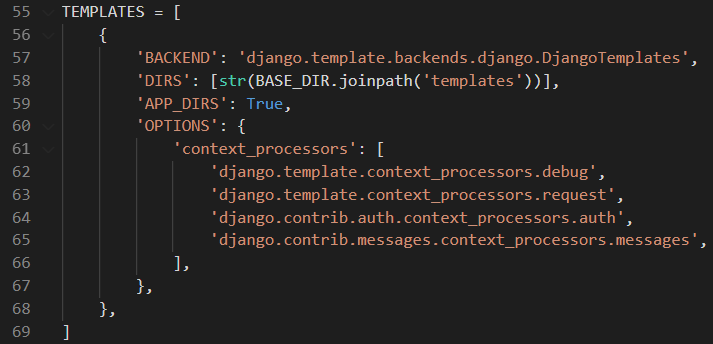
Move into the templates folder and create a new file called **home.html**:

(env) djangoprojects\ lab3\templates >new-item home.html

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

templates directory.

**Code**

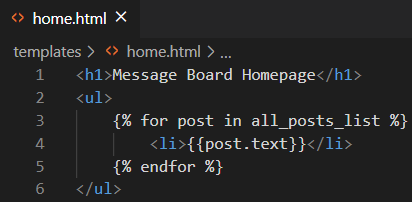


In our templates file **home.html** we can use the Django Templating Language’s **for loop** to list all the objects in **object\_list**. This is the name of the variable that **ListView** returns to us. We should specify not just the object we want, post, but the specific field we want to display which is text - we will use **post.text**.

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

**Code**

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

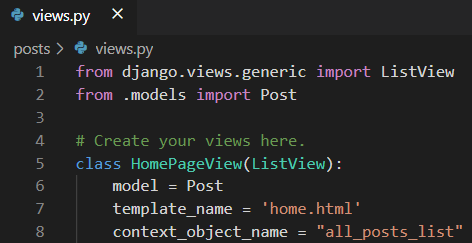


However, **object\_list** is not a very user friendly name so instead we will provide an explicit name which we can do via **context\_object\_name**.

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.

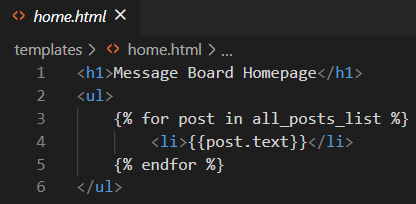
Back in our **posts/views.py** file add the following:

**Code**



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:

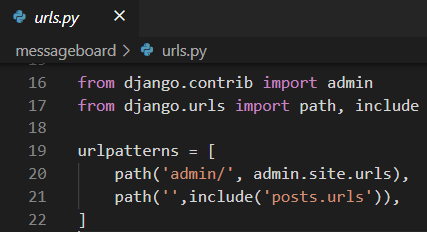
**Code**



The last step is to set up our URLConfs. Open the **messageboard/urls.py** file

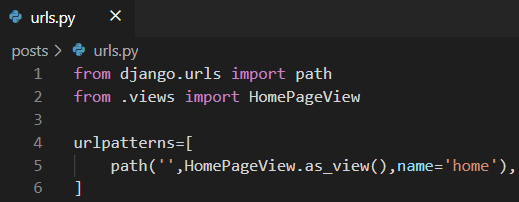
where we simply include our posts and add include on the second line.

**Code**



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

**Code**



Restart the server with **python manage.py runserver** and navigate to our homepage http://127.0.0.1:8000/ which now lists out our message board posts.



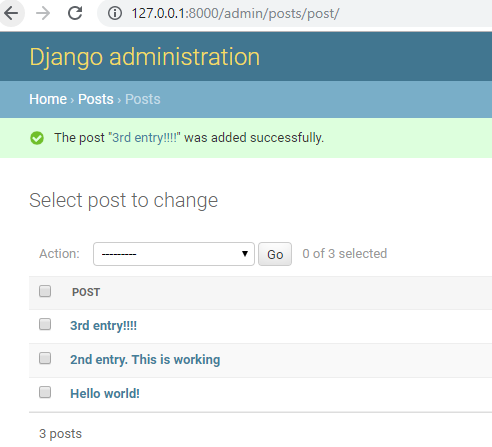
Create a few more message board posts in the Django admin to confirm that they will display correctly on the homepage.

**Adding new posts**

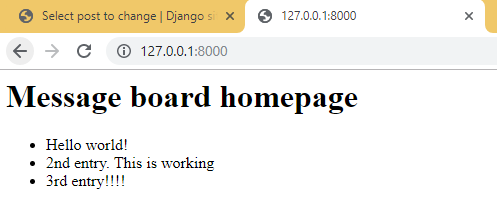
To add new posts to our message board, go back into the Admin:

http://127.0.0.1:8000/admin/

Create two more posts similar to those shown below (Go back to page 8 to see how we added our first post) :



If you return to the homepage you will see that it automatically displays our formatted posts:



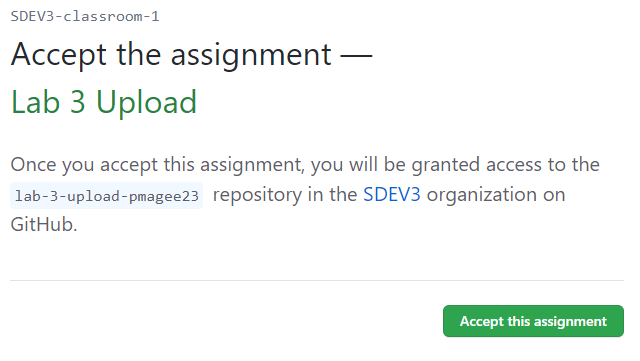
**Upload of Project to GitHub Classroom**

Go to [www.github.com](http://www.github.com) and log in to your GitHub account

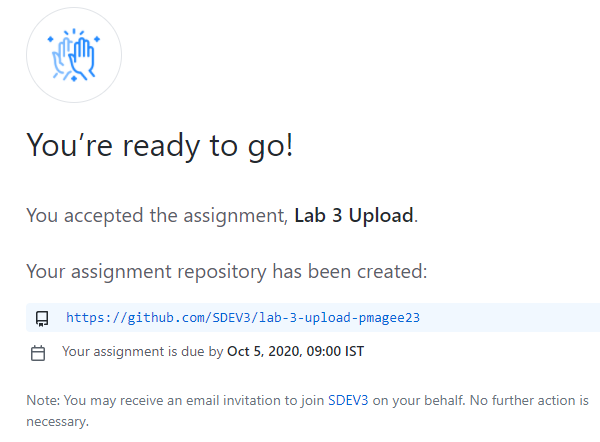
Go to Moodle and when you find the link shown below, click on it.



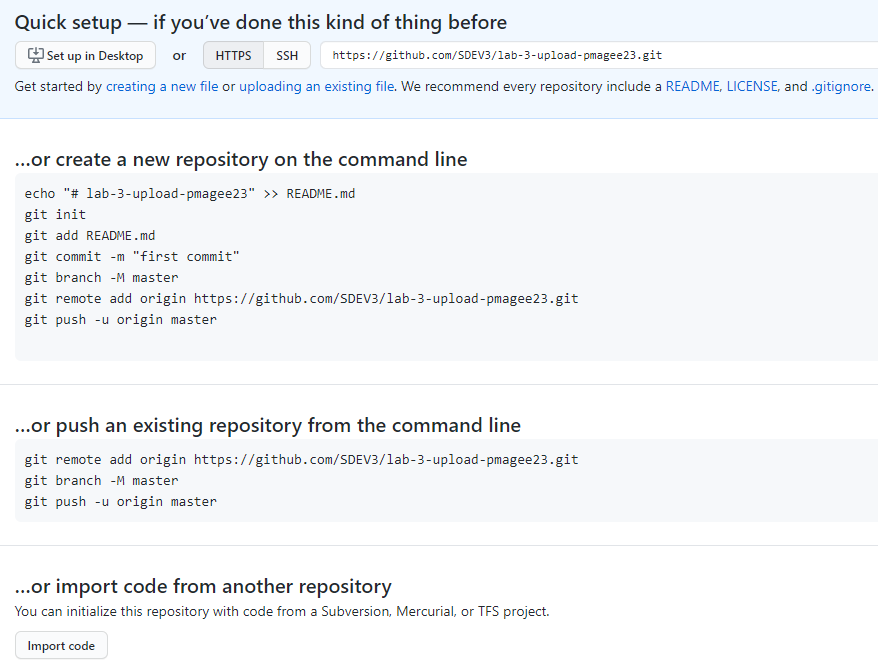
Next you should see a screen similar to the following asking you to accept the assignment.



Once you have accepted the assignment you are presented with a link to your repository for lab 2 as shown below:

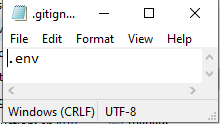


When you click on the repository link, you are taken to the repository where you see the list of commands as shown below:

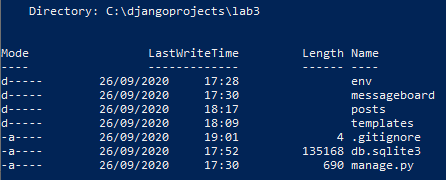


Before we initialise git we are going to look at a way to prevent the **env** folder containing all the virtual environment files from being uploaded to the repository on GitHub.

In order to do this, you need to create a file called .gitignore in the root directory of your project using Notepad and add .env on its own line in the file as shown below.



Make sure this file is sitting in the right location i.e. in the lab3 folder as shown below:



Now you can go ahead and initialise git to your repository.

In PowerShell type the following command:

(env) djangoprojects\lab3>git init

Add all the changes by typing the following command

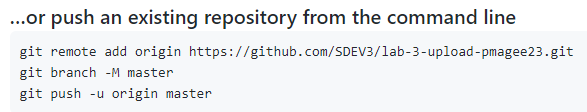
(env) djangoprojects\lab3>git add -A

Commit the changes along with a suitable message

(env) djangoprojects\lab3>git commit -m “initial commit”

The next step is to sync the local directory on your computer with the lab 3 repository.

Go to the page for your repository that contains the list of commands and copy the line of code similar to the one shown below and paste it into the command line:



Push your code to GitHub by typing the following command:

(env) djangoprojects\lab3>git push -u origin master

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

