Lab 2

In this exercise we will create a Django project with an app called pages that has two pages - a homepage and an about page. We will also use Django’s class-based views and templates which are the building blocks for the more complex web applications we will build later on in the module.

Open **PowerShell**, move into the directory called **djangoprojects** and create a new Django project called **lab2**

mkdir lab2

Move into the **lab2** 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\lab2>

Install Django using the following command:

(env) djangoprojects\lab2>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\ lab2>python -m pip install –upgrade pip

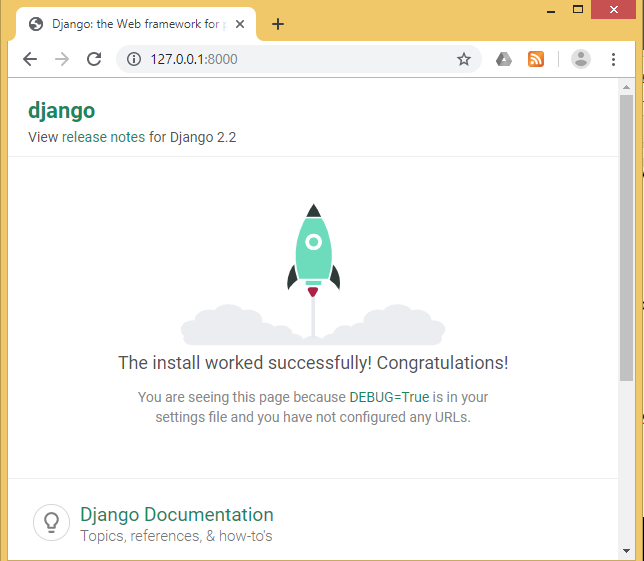
Create a new Django project called **pagesproj** with the following command. Don’t forget the period (fullstop) at the end:

(env) djangoprojects\ lab2>django-admin startproject pagesproj .

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

(env) djangoprojects\ lab2 >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\ lab2 >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 p**ages**. From the command line, quit the server with Control+BREAK. Then use the startapp command as shown below:

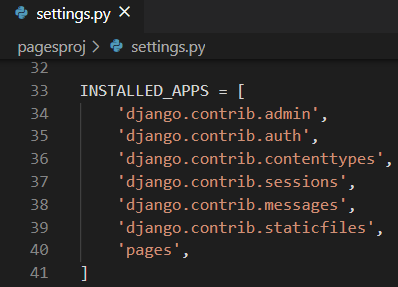
(env) djangoprojects\ lab2 >python manage.py startapp pages

**Settings.py**

Even though our new app exists within the Django project, Django doesn’t “know” about it until we explicitly add it.

1. Launch the VS Code IDE and open your project in this IDE.
2. In your text editor open the settings.py file and scroll down to INSTALLED\_APPS where you’ll see six built-in Django apps already there.
3. Add our new pages app at the bottom.

**Code**



**Templates**

In order to create HTML files in Django we use templates. A template is a text file that consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted. In Django we use templates so that individual HTML files can be served by a view to a web page specified by the URL.

In the previous lab where we created a Django project to display the message “Hello, World” on the page, we had the phrase hardcoded into a views.py file as a string. That technically works but doesn’t scale well! A better approach is to link a view to a template, thereby separating the information contained in each.

Before we create a template we have to decide where exactly we will place it in our project structure. We will create a single project-level templates directory and place all templates within there. By making a small tweak to our config/settings.py file we can tell Django to also look in this directory for templates.

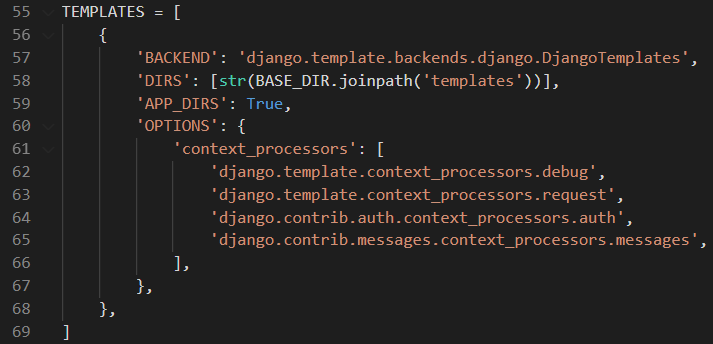
First, quit the running server with the Control+BREAK command. Then create a directory called templates and an HTML file called home.html.

(env) djangoprojects\ lab2 >mkdir templates

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

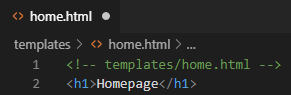
(env) djangoprojects\ lab2\templates >new-item home.html

Next we need to update **config/settings.py** to tell Django the location of our new templates directory. This is a one-line change to the setting **'DIRS'** under **TEMPLATES**.



Then we can add a simple headline to the body of our home.html file.

**Code**



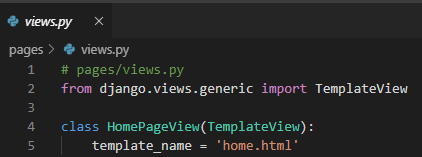
The next step is to configure our URL and view.

**Class-Based Views**

Classes are a fundamental part of Python. In our view we’ll use the built-in TemplateView to display our template.

Update the **pages/views.py** file using the code below:

**Code**



Note that we’ve capitalized our view since it’s now a Python class. Classes, unlike

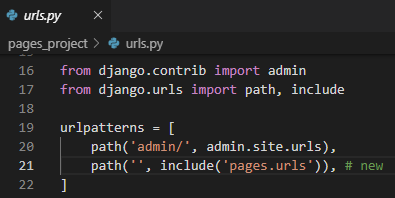
functions, should always be capitalized. The TemplateView already contains all the logic needed to display our template, we just need to specify the template’s name.

**URLs**

The last step is to update our URLConfs. Recall from Lab 1 that we need to make updates in two locations. First we update the pages\_project/urls.py file to point at our pages app and then within pages we match the views to routes.

Let’s start with the **pages\_project/urls.py** file.

**Code**

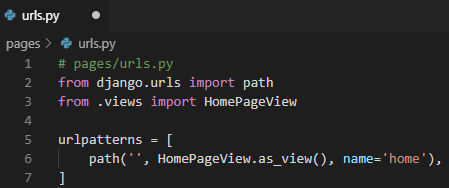


The code here should look familiar at this point. We add include on the second line

to point the existing URL to the pages app.

Next in VS Code, create an app-level urls.py file and add the following code.

**Code**



This pattern is almost identical to what we did in Lab1 with one major difference.

When using Class-Based Views, you always add as\_view() at the end of the view name.

And we’re done! If you start up the web server with python manage.py runserver and

navigate to http://127.0.0.1:8000/ you can see our new homepage.

**Add an About Page**

The process for adding an about page is very similar to what we just did. We will create

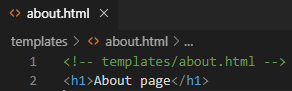
a new template file, a new view, and a new url route.

Quit the server with Control+BREAK and create a new template called **about.html**.

(env) djangoprojects\ lab2\templates >new-item about.html

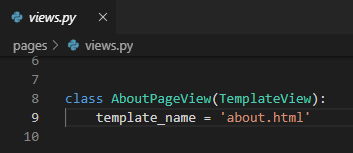
Open this new file in VS Code and add a simple headline to the body of our about.html file.

**Code**



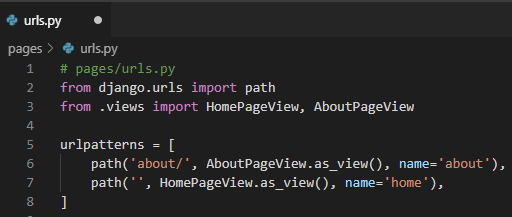
Create a new view for the page:

**Code**



And then connect it to a URL /about by adding the following code to pages/urls.py

**Code**



Start up the web server with python manage.py runserver.

Navigate to http://127.0.0.1:8000/about and you can see our new “About page”.

**Extending Templates**

One of the main advantages of using templates is their ability to be extended. If you think about most websites, there is content that is repeated on every page (header, footer, etc). In Django we can have one place for our header code that is inherited by all other templates.

Let’s create a base.html file containing a header with links to our two pages. We could name this file anything but using base.html is a common convention.

Stop the server using Control+BREAK and in PowerShell, create a new template page called base.html.

(env) djangoprojects\ lab2\templates >new-item base.html

Django has a minimal templating language for adding links and basic logic in our templates. You can see the full list of built-in template tags in the official documentation.

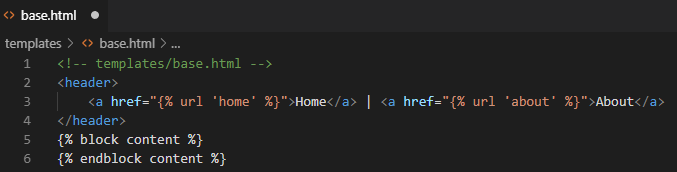
Template tags take the form of **{% something %}** where the “something” is the template tag itself.

To add URL links in our project we can use the built-in **url** template tag which takes the URL pattern name as an argument. We can use the optional URL names that we created in the urls.py file. The **url** tag uses these names to automatically create links for us.

The URL route for our homepage is called home therefore to configure a link to it we

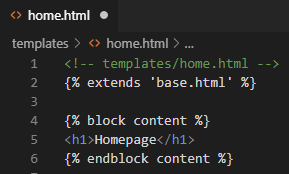
would use the following: **{% url 'home' %}**.

In VS Code, enter the following code into the file **base.html** making sure to avoid any typos.

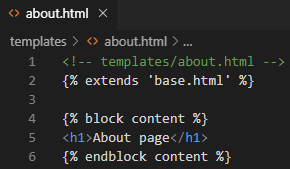
At the bottom we have added a block tag called **content**. Blocks can be overwritten by child templates via inheritance. While it is optional to name our closing endblock – you can just write **{% endblock %}** if you prefer–doing so helps with readability, especially in larger template files.

Update the files **home.html** and **about.html** to extend the **base.html** template. That means we can reuse the same code from one template in another template. The Django templating language comes with an **extends** method that we can use for this.

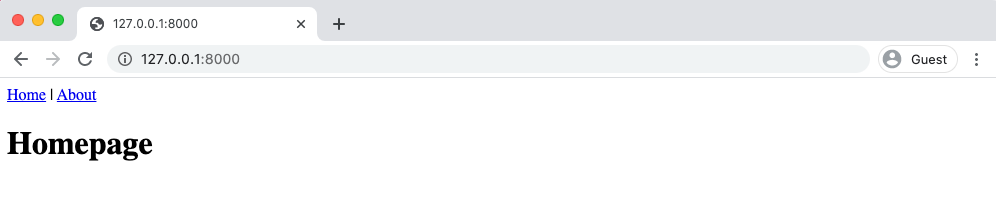
**Code**

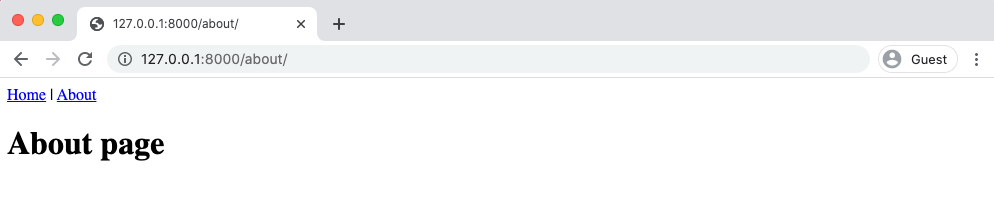


**Code**



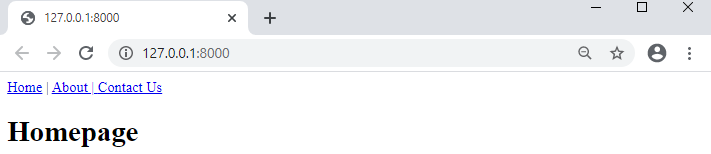
Now if you start up the server with python manage.py runserver and open up our webpages again at http://127.0.0.1:8000/ and http://127.0.0.1:8000/about you will see the header is now included in both locations.

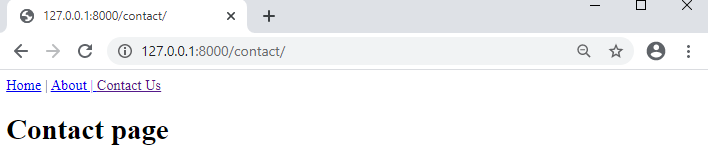




As you will see the steps for creating the home and about pages are very similar. Try to create a new html file called **contact.html** using the steps provided for creating the other pages.

When you run the server, you should then see a “Contact Us” link on the home page which when clicked brings us to **contact.html.**





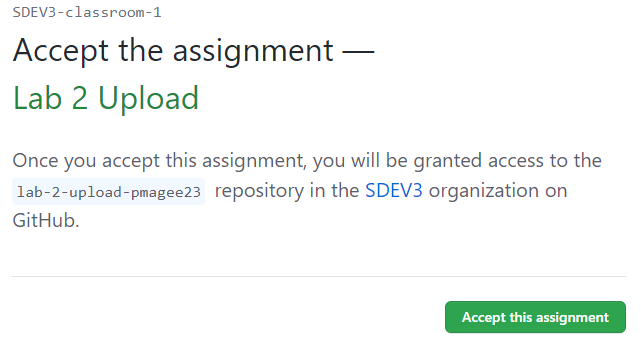
**Upload of Project to GitHub Classroom**

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

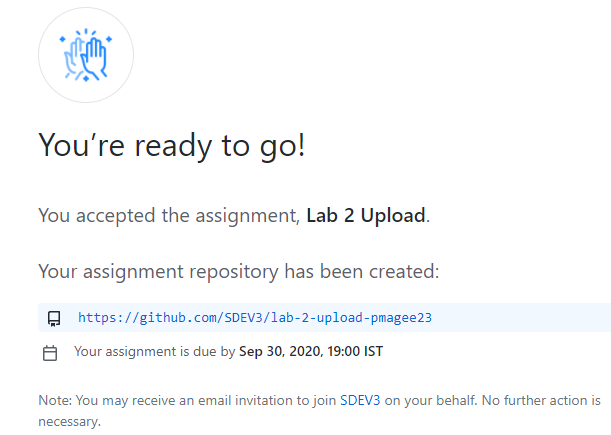
Go to Moodle and when you find the link shown below, click on it.



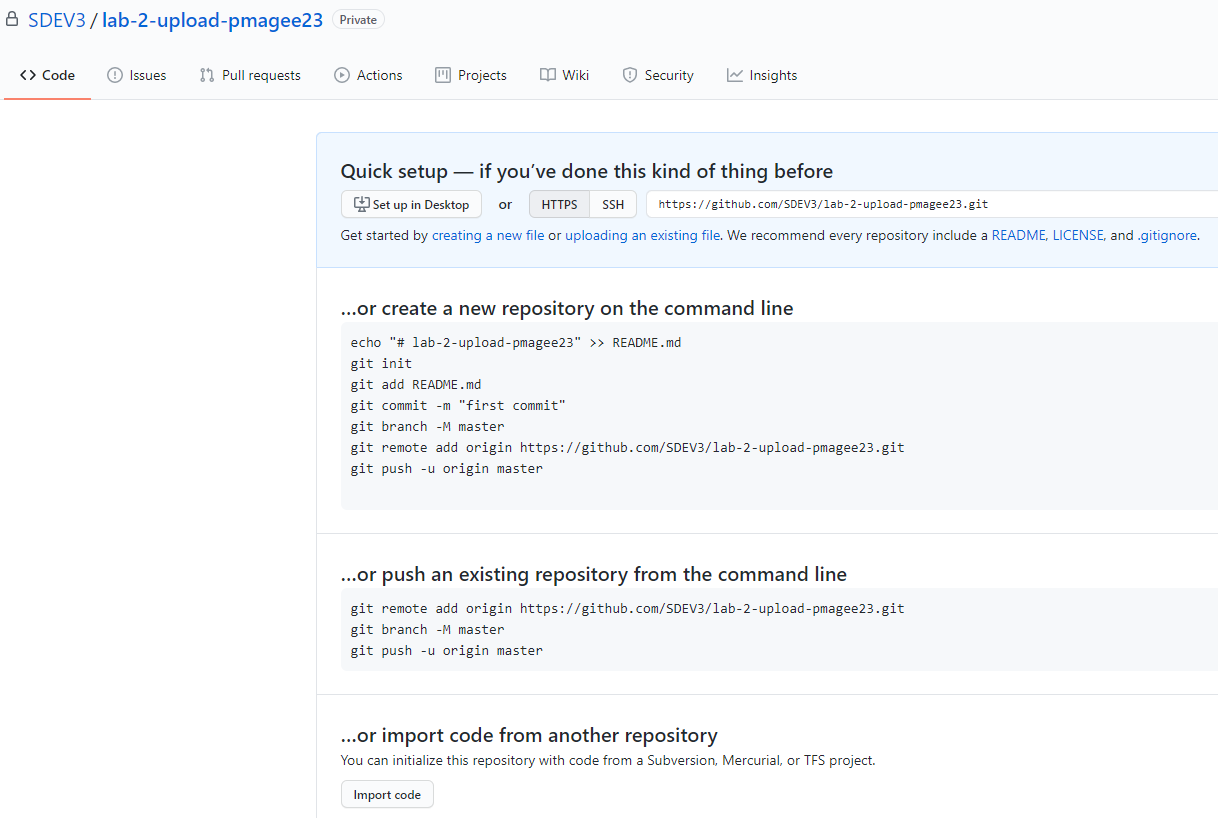
Next you should see a screen similar to the following asking you to accept the assignment.



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

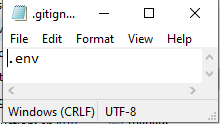


When you click on the repository link, you are taken to the repository where you see the list of commands as shown below:

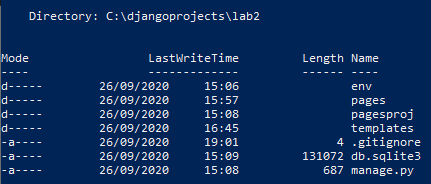


Before we initialise git we are going to look at a way to prevent the **env** folder containing all the virtual environment files from being uploaded to the repository on GitHub.

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 lab2 folder as shown below:



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

In PowerShell type the following command:

(env) djangoprojects\lab2>git init

Add all the changes by typing the following command

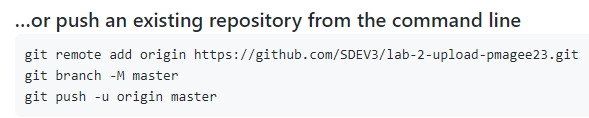
(env) djangoprojects\lab2>git add -A

Commit the changes along with a suitable message

(env) djangoprojects\lab2>git commit -m “initial commit”

The next step is to sync the local directory on your computer with the lab 2 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\lab2>git push -u origin master

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

