The first thing we want to do is get our Django server up and running with all the basics:

* Install Django inside of a virtual environment
* Start the project
* Start the app
* Add the app
* Create an empty model so we can create the first migration
* Make the migration, and Migrate
* Create the super user for the admin area
* Create an urls.py file in the app directory
* Create the templates directory
* Create the template inside the app’s template folder
* Create the function inside the views.py file
* Create the URL that links the function in views.py with the template

Like that, we have our very basic Django server. For this example, we are going to learn how to create users inside of the Django app, not creating them since the beginning but using the Django baked-in methods for user registration, log-in, log-out, and deleting the user.

This is not meant to be complicated is just meant to be a simple re-usable app for future Django projects and practice purposes only. The page is only going to go to a page where you can see the user’s name, and there the user will have access to his data and be able to log out, log in, and delete his account.

**First, We must begin with a form. What is a Form?**

We have two types of requests when we are requesting a web page, or two methods, one is called “GET” and the other one “POST”. But How do we know when is the user requesting or submitting to the page? Well, is very simple because this is defined in the HTML file. This happens when we create the submit form, and there we say that the method to get to the page is equal to “POST”, very simple. This is how it looks in the HTML file:

<form action="" method=POST>

            {% csrf\_token %}

            <div class="mb-3">

              <label for="exampleInputEmail1" class="form-label">Username</label>

              <input type="text" class="form-control" name="username" aria-describedby="emailHelp">

            </div>

            <div class="mb-3">

              <label for="exampleInputPassword1" class="form-label">Password</label>

              <input type="password" class="form-control" name="password">

            </div>

            <input type="submit" value = "Submit" class="btn btn-primary">

          </form>

This way, you can define a function in the views.py file, or in the forms.py file that we are about to create, where this function takes the request, but, differently, as we did before because now we can use some simple logic to determine if the user is trying to just get the page, or if he is trying to submit some information. This information can be anything: The registration form, authentication, anything. The deal is that if the user is “posting” to the page then the method is defined as a post. So, this is what we will do:

We must change the way we access the URL since the URL that we are using will be Django’s. We must create two paths with the same name because Django has to deal with all the authentication things and we do that with the first URL. And this we must change it in the **main urls.py**.

Father\_Dir:

>Hashimete:

| >Hashimete:

| | §\_\_init\_\_.py

| | §settings.py

| | §urls.py

| | §wsgi.py

| |

| §manage.py

| ≡db.sqlite3

≡Pipfile

{}Pipfile.lock

*§urls.py*

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('django.contrib.auth.urls')),

    path('', include('user\_registration\_app.urls')),

]

Now that we have all that, let’s create a Log- in and registration Template, so our users can log in. This one won’t be complex, just a Welcome text, a login, and a register button. We will only build the functionalities.

We do this the same way as always, we create the login template, with its function in views.py, and of course our URL in the urls.py file.

Now is the time to create the form that will take in that information and pass it to the tables that we already have built inside Django.

For that, we are going to use Bootstrap because is super easy.

<form action="" method=POST>

        {% csrf\_token %}

        <div class="mb-3">

          <label for="exampleInputEmail1" class="form-label">Username</label>

          <input type="text" class="form-control" name="username" aria-describedby="emailHelp">

        </div>

        <div class="mb-3">

          <label for="exampleInputPassword1" class="form-label">Password</label>

          <input type="password" class="form-control" name="password">

        </div>

        <input type="submit" value = "Submit" class="btn btn-primary">

      </form>

**Log-in:**

This functionality might be more complex than we’ll see today. Even though is not that complicated we will use Django’s built-in function classes and methods to it, by creating forms and applying python logic to determine what data will be shown to the user, depending on if he is authenticated or not

**IMPORTANT:** If you don’t set the method in the form to post, every time that you hit submit nothing will happen because the method is set to GET by default. And notice the name that is being passed to the variables.

All right now that we have the exterior, let’s build the actual functionality of the webpage, by using Django’s authentication system and log in, and logout tools.

For that, we must type in the Google bar “Django Authentication”, and go to our Official documentation which will be very helpful for this application. Next, we go to the link “Authentication in web requests” and then we copy the code that they already have over there and paste it into our login\_user function in our views.py file.

This is what we want to copy:

**from** **django.contrib.auth** **import** authenticate, login

**def** my\_view(request):

username = request.POST['username']

password = request.POST['password']

user = authenticate(request, username=username, password=password)

**if** user **is** **not** **None**:

login(request, user)

*# Redirect to a success page.*

...

**else**:

*# Return an 'invalid login' error message.*

...

But before, we have to make sure that the person is not logged in already, so it won’t send a logged-in user to the login link. We can do that with a little bit of python logic using variables and “if else” logic.

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login, logout

from django.contrib import messages

def home(request):

    return render (request, 'ur\_app/home.html')

def login\_user(request):

    if request.method == "POST":

        username = request.POST['username']

        password = request.POST['password']

        user = authenticate(request, username=username, password=password)

        if user is not None:

            login(request, user)

            return redirect ('home')# Redirect to a success page.

        else:

            messages.success(request, "There was an error, try again")

            return redirect('login')

    return render (request, 'ur\_app/login.html', {})

**IMPORTANT:** A very important detail to be noticed, is that the variable **request** is referencing just a string from which data is obtained. With this data, we can get variables from it, and create unique ‘paths’ for unique requests. That’s why all those functions always use the request variable as the first argument because is necessary to fetch the information that is being held by this string.

To be imported:

\* render, redirect, authenticate, login, logout

Defining a function to log in, with a different name than login, because django.contrib.auth already has one with that name

Here we create a simple python logic to determine if the method used to access the web page is POST or GET

These variables are referencing the variables that are passed to the backend when we make the request, the name of the variables has already been established in the HTML file, and Those variables are obtained by marking them with a ? symbol, and separated by a & symbol and giving it the value with an = symbol:

http://127.0.0.1:8000/login\_user/?username=alice&password=\*\*\*\*\*\*\*\*

This can be also represented in a Python dictionary like this:

{‘username’: ‘alice’, ‘password’:’\*\*\*\*\*’}

If we see it this way, what we are doing in these lines of code is just getting this data, and turning it into variables, from this point, is just python.

The next green line of code is just authentication of the username and password that we got from the POST dictionary, and this return something to that variable, which is the username and password data, and if after being authenticated and validated this variable is not NONE:

Python Logic

If the user is not an empty string or empty data, then we will use the login function from Django which will set this user, as the user that is logged in rn

Also, we use the redirect that we imported to redirect us to another page, right now we are using the home, but it cud be any other page,

Python Logic >>>Is triggered by user = None

We use the messages function we imported to return a message to our user so they know that something is going on and that he needs to fix something.

Also, we redirect to the same page we are in, this way all the fields will be empty again

Python Logic>>>Triggered by the method being GET, this means that the form is not submitted, that we are only requesting to get to the page.

All right, now let’s finish the login part by adding the message functionality to our HTML in our bame.html file, which is the one that the rest of the pages get information on.

{% if messages %}

    {% for message in messages %}

    <div class="alert alert-danger alert-dismissible fade show" role="alert">

      <strong>Error</strong> {{ message }}

      <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>

    </div>

    {% endfor %}

    {% endif %}

Not complicated, just some python logic with a whole lot of Bootstrap wrapping it. This means that if there is any messages variable being passed into the front end of the project, this variable will be True, so this will trigger the rest of the command blocks.

**Log-Out:**

To log out the first thing we want to do is create the logout function inside of our views.py file the same way we did with our login function. This means that we must give it a different name than logout because log out is already a function that comes baked in with Django.

*§views.py*

def logout\_user(request):

    logout(request)

    messages.success(request, ("Logged Out"))

    return redirect('home')

As for the view that is all that there is to it. Now, we can’t use the word **logout** or **logout\_user** in the href in the HTML file because it will redirect you to the log out of Django, another solution would be to add another path before it but I just changed the name

{% if user.is\_authenticated %}

      <a class="navbar-brand" href="l\_out">Log-out</a>

      {% endif %}

Also, we can see that I use a python logic block to determine if the user is authenticated, or logged in, so I won’t show a logout button to a not logged-in user.

**user.is\_authenticaded** is the syntaxis that we use because we have the user class, object, or module, I don’t know, which has a function or method inside itself that controls that. Also, we can access the username by calling **user.username** .

**Register:**

For this example, we will just let Django manage the user created for us, so it’s going to be easy. First, let’s create a view or a function that will handle user creation.

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login, logout

from django.contrib import messages

from django.contrib.auth.forms import UserCreationForm

>>>Here we import a built-in function that comes with Django, so we don’t have to create the form from scratch.

Now let’s create the **register\_user**  function:

def register\_user(request):

    if request.method == 'POST':

        form = UserCreationForm(request.POST)

        if form.is\_valid():

            form.save()

            username = form.cleaned\_data['username']

            password = form.cleaned\_data['password1']

            user = authenticate( username=username, password=password)

            login(request, user)

            messages.success(request, "Registration Successfull")

            return redirect('home')

    else:

        form = UserCreationForm()

    return render (request, 'ur\_app/register\_user.html',{

    'form': form,

    })

First, we create an instance of the UserCreationForm class, passing it the request, being modified by the POST method so is easier to harvest info from the string.

\*is\_valid, save, cleaned\_data are UserCreationForm class’s methods:

**Is\_valid**: makes sure that info is valid

**save**: saves to form, adds its contents to the table

**cleaned\_data:** it cleans the data

All these functionalities are from the login\_user function

If the user is just requesting the page, meaning that the method is GET, we return the form to be filled out.

**IMPORTANT:** Is very easy to forget the comma in the past in the dictionary if it only has one element in, SO, don’t forget.

**Extra Fields:**

Django also has those extra fields functionality so It’s going to be very easy for us to add them. The first thing we will do is go to our forms.py file and create a completely new form, with the UserCreationForm as a base.

Then once we have created it, we just must replace it in the view, as simple as that.

*§forms.py*

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

from django import forms

class RegisterUserForm(UserCreationForm):

    email = forms.EmailField()

    first\_name = forms.CharField(max\_length=100)

    last\_name = forms.CharField(max\_length=100)

    class Meta:

        model = User

        fields = ('username','first\_name','last\_name','email','password1', 'password2',)

Here we import all the tools that we are going to need to extend this form:

**UserCreationForm:** The original form to be extended

**User:** This is the model that we are using, the one that comes with Django

**forms:** this is the forms module from django.

Next, we create the extended class using the UserCreationForm as a father so we inherit all of its functionalities. Also, we add the fields to be added using the **forms** module’s methods to make it easier.

The class meta is the class that we create for the fields, is not something that is explained, is just something that we do. Inside of the Meta class, we define the model or table to be filled, and the fields to be shown and created.

**IMPORTANT:** For now, those are the fields that we can add because those fields are already defined in the User model or table that comes baked in with Django

**Bootstrapping it:**

For the fields that we added is easy to add bootstrap since we can give it a bootstrap class when creating the field using the widgets. This is the same for the username and password fields, but since they are built-in, we must open the \_\_init\_\_ and add the attributes to their widgets in the \_\_init\_\_ function.

The first thing we want to do is use bootstrap with the fields that we can use without modifying the class:

*§forms.py*

class RegisterUserForm(UserCreationForm):

    email = forms.EmailField(widget= forms.EmailInput( attrs= {'class': 'form-control'}))

    first\_name = forms.CharField(max\_length=100, widget= forms.TextInput( attrs= {'class': 'form-control'}))

    last\_name = forms.CharField(max\_length=100, widget= forms.TextInput( attrs= {'class': 'form-control'}))

Here there is not too much to it, the fields that we added can be easily bootstrapped by using a widget, which has a type of input that depends on the type of field that it is(if it is **EmailField** then the widget’s name will be **EmailInput**). Once we apply the widget, we give it its **attrs**, or attributes, which are going to be loaded from the HTML, which is loading this info from the bootstrap website.

Now let’s see how we give a widget to an attribute that was set by default to not have a widget at the moment of its creation.

*§forms.py*

def \_\_init\_\_(self, \*args, \*\*kwargs):

        super(RegisterUserForm, self).\_\_init\_\_(\*args, \*\*kwargs)

        self.fields['username'].widget.attrs['class'] = 'form-control'

        self.fields['password1'].widget.attrs['class'] = 'form-control'

        self.fields['password2'].widget.attrs['class'] = 'form-control'

Is not complicated. We only need to redefine the init method that was already defined at the moment of the class’s creation, by redefining the \_\_init\_\_ itself.

The first line of code is super basic when treating with classes and redefining an \_\_init\_\_, which is just the convention. Then what we want to do here is define the superclass, which is **RegisterUserForm,** and also pass the **self**, we do this because we need to refer to the instance from which we don’t know its name. Then we use the Method \_\_init\_\_ which is the constructor method, so we sing the \_\_init\_\_ to modify the \_\_init\_\_.

Now, we see why we defined the “**self”** earlier, and i. We will use it now to modify the attribute fields, which I can see that is a dictionary because we are accessing it by key.

Once we access the fields, **username**, for example, we modify its widget’s **attrs,** which again, I can see is a dictionary, by giving to its key **class** a value of **‘form-control’**.

Comparison:

class RegisterUserForm(UserCreationForm):

    email = forms.EmailField(widget= forms.EmailInput( attrs= {'class': 'form-control'}))

def \_\_init\_\_(self, \*args, \*\*kwargs):

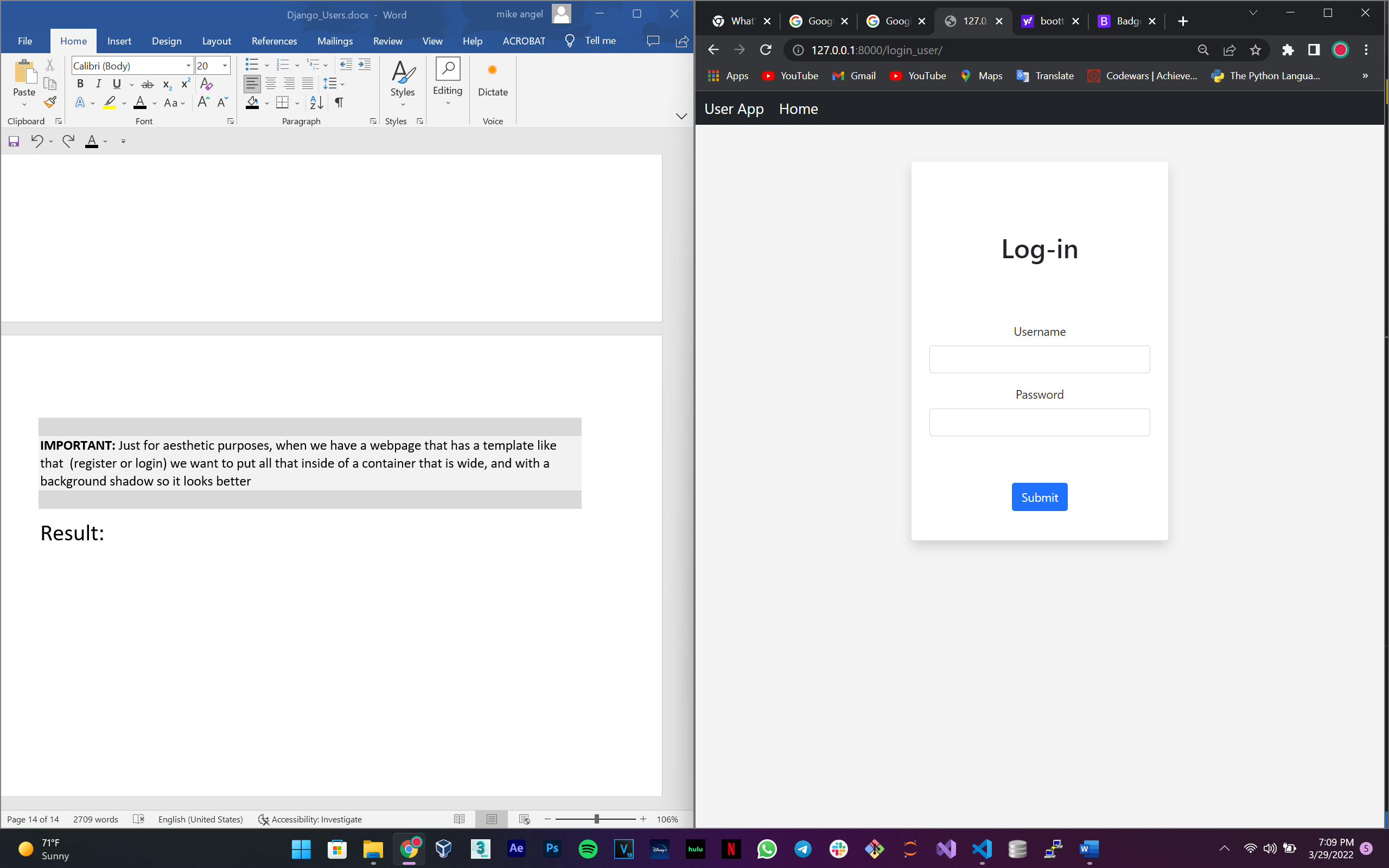
        super(RegisterUserForm, self).\_\_init\_\_(\*args, \*\*kwargs)

        self.fields['username'].widget.attrs['class'] = 'form-control'

**IMPORTANT:** Just for aesthetic purposes, when we have a webpage that has a template like that (register or login) we want to put all that inside of a container that is wide, and with a background shadow so it looks better

Result:

Log-in



Register:

Graphical user interface, application, Word

Description automatically generated

**Authentication-based Permission:**

I already learned this on my own, is easy to do so by modifying the HTML so it will only show the info that we want to show

There are two important things to consider here:

* IF, the user is authenticated.
* WHICH user is authenticated?

I already did that by using the authenticated line of code to show navbar items, and the WHICH user, to show the name of the user. Now to make this more complex I need to learn how to link tables so I can create relationships between tables.

**IMPORTANT:** Django’s baked-in functions make all this very easy by verifying which user is authenticated, so when we, for example, try to show the user’s name, we can do so easily by accessing the Django’s user baked-in table

Graphical user interface, text, application

Description automatically generated

As we can see, this is the baked-in table, or model that Django offers to create, modify, or delete a user.