PARULINSTITUTEOF ENGINEERING &TECHNOLOGY
FACULTY OF ENGINEERING & TECHNOLOGY
PARULUNIVERSITY

# 303105257 - Programming in Python with Full Stack Development

## Functions.

Computer Science & Engineering

Prabhat Parashar(Asst. Professor. PIET-CSE)

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

# Table of Contents:

# Introduction to Django Framework

Django is a Python framework that makes it easier to create web sites using Python.

Django takes care of the difficult stuff so that you can concentrate on building your web applications.

Django emphasizes reusability of components, also referred to as DRY (Don't Repeat Yourself), and comes with ready-to-use features like login system, database connection and CRUD operations (Create Read Update Delete).

Django is especially helpful for database driven websites.

Parul® University

NAAC ACCREDITED UNIVERSITY A++

DIGITAL LEARNING CONTENT

# Introduction to Django Framework

Django follows the MVT design pattern (Model View Template).

- Model - The data you want to present, usually data from a database.
- View - A request handler that returns the relevant template and content - based on the request from the user.
- Template - A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.

# Introduction to Django Framework

## Model

The model provides data from the database.

In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases.

The most common way to extract data from a database is SQL. One problem with SQL is that you have to have a pretty good understanding of the database structure to be able to work with it.

Django, with ORM, makes it easier to communicate with the database, without having to write complex SQL statements.

The models are usually located in a file called `models.py`.

# Introduction to Django Framework

## View

A view is a function or method that takes http requests as arguments, imports the relevant model(s), and finds out what data to send to the template, and returns the final result.

The views are usually located in a file called `views.py`.

# Introduction to Django Framework

## Template

A template is a file where you describe how the result should be represented.

Templates are often .html files, with HTML code describing the layout of a web page, but it can also be in other file formats to present other results, but we will concentrate on .html files.

Django uses standard HTML to describe the layout, but uses Django tags to add logic:

```
<h1>My Homepage</h1>

<p>My name is {{ firstname }}.</p>
```

The templates of an application is located in a folder named `templates`.

# Installation of Virtual Environment

To install Django, you must have Python installed, and a package manager like PIP.

PIP is included in Python from version 3.4.

## Django Requires Python

To check if your system has Python installed, run this command in the command prompt:

```
python --version
```

If Python is installed, you will get a result with the version number, like this

```
Python 3.9.2
```

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Installation of Virtual Environment

If you find that you do not have Python installed on your computer, then you can download it for free from the following website: https://www.python.org/

## PIP

To install Django, you must use a package manager like PIP, which is included in Python from version 3.4.

To check if your system has PIP installed, run this command in the command prompt:

```
pip --version
```

If PIP is installed, you will get a result with the version number.

For me, on a windows machine, the result looks like this:

```
pip 20.2.3 from c:\python39\lib\site-packages\pip (python 3.9)
```

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Installation of Virtual Environment

## Virtual Environment

It is suggested to have a dedicated virtual environment for each Django project, and one way to manage a virtual environment is venv, which is included in Python.

The name of the virtual environment is your choice, in this tutorial we will call it `myworld`.

Type the following in the command prompt, remember to navigate to where you want to create your project:

Windows:

```
py -m venv myworld
```

Unix/MacOS:

```
python -m venv myworld
```

This will set up a virtual environment, and create a folder named "myworld" with subfolders and files, like this:

# Installation of Virtual Environment

```
myworld
  Include
  Lib
  Scripts
  pyvenv.cfg
```

Then you have to activate the environment, by typing this command:

Windows:

```
myworld\Scripts\activate.bat
```

Unix/MacOS:

```
source myworld/bin/activate
```

Once the environment is activated, you will see this result in the command prompt:

# Installation of Virtual Environment

Windows:

```
(myworld) C:\Users\Your Name>
```

Unix/MacOS:

```
(myworld) ... $
```

**Note:** You must activate the virtual environment every time you open the command prompt to work on your project.

# Installation of Virtual Environment

Windows:

```
(myworld) C:\Users\Your Name>
```

Unix/MacOS:

```
(myworld) ... $
```

**Note:** You must activate the virtual environment every time you open the command prompt to work on your project.

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Installation of Virtual Environment

## Install Django

Now, that we have created a virtual environment, we are ready to install Django.

**Note:** Remember to install Django while you are in the virtual environment!

Django is installed using pip, with this command:

Windows:

```
(myworld) C:\Users\Your Name>py -m pip install Django
```

Unix/MacOS:

```
(myworld) ... $ python -m pip install Django
```

Which will give a result that looks like this (at least on my Windows machine):

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Installation of Virtual Environment

## Install Django

Now, that we have created a virtual environment, we are ready to install Django.

**Note:** Remember to install Django while you are in the virtual environment!

Django is installed using pip, with this command:

Windows:

```
(myworld) C:\Users\Your Name>py -m pip install Django
```

Unix/MacOS:

```
(myworld) ... $ python -m pip install Django
```

Which will give a result that looks like this (at least on my Windows machine):

# Installation of Virtual Environment

Which will give a result that looks like this (at least on my Windows machine):

```
Collecting Django
  Downloading Django-4.0.3-py3-none-any.whl (8.0 MB)
     |████████████████████████████████| 8.0 MB 2.2 MB/s
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata; sys_platform == "win32"
  Downloading tzdata-2021.5-py2.py3-none-any.whl (339 kB)
     |████████████████████████████████| 339 kB 6.4 MB/s
Installing collected packages: sqlparse, asgiref, tzdata, Django
Successfully installed Django-4.0.3 asgiref-3.5.0 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 20.2.3; however, version 22.3 is available.
You should consider upgrading via the 'C:\Users\Your Name\myworld\Scripts\python.exe -m pip install
--upgrade pip' command.
```

That's it! Now you have installed Django in your new project, running in a virtual environment!

# Phases of Django Project Creation

Once you have come up with a suitable name for your Django project, like mine: `my_tennis_club`, navigate to where in the file system you want to store the code (in the virtual environment), I will navigate to the `myworld` folder, and run this command in the command prompt:

```
django-admin startproject my_tennis_club
```

Django creates a `my_tennis_club` folder on my computer, with this content:

```
my_tennis_club
    manage.py
    my_tennis_club/
        __init__.py
        asgi.py
        settings.py
        urls.py
        wsgi.py
```

# Phases of Django Project Creation

These are all files and folders with a specific meaning, you will learn about some of them later in this tutorial, but for now, it is more important to know that this is the location of your project, and that you can start building applications in it.

## Run the Django Project

Now that you have a Django project, you can run it, and see what it looks like in a browser.

Navigate to the `/my_tennis_club` folder and execute this command in the command prompt:

```
py manage.py runserver
```

Which will produce this result:

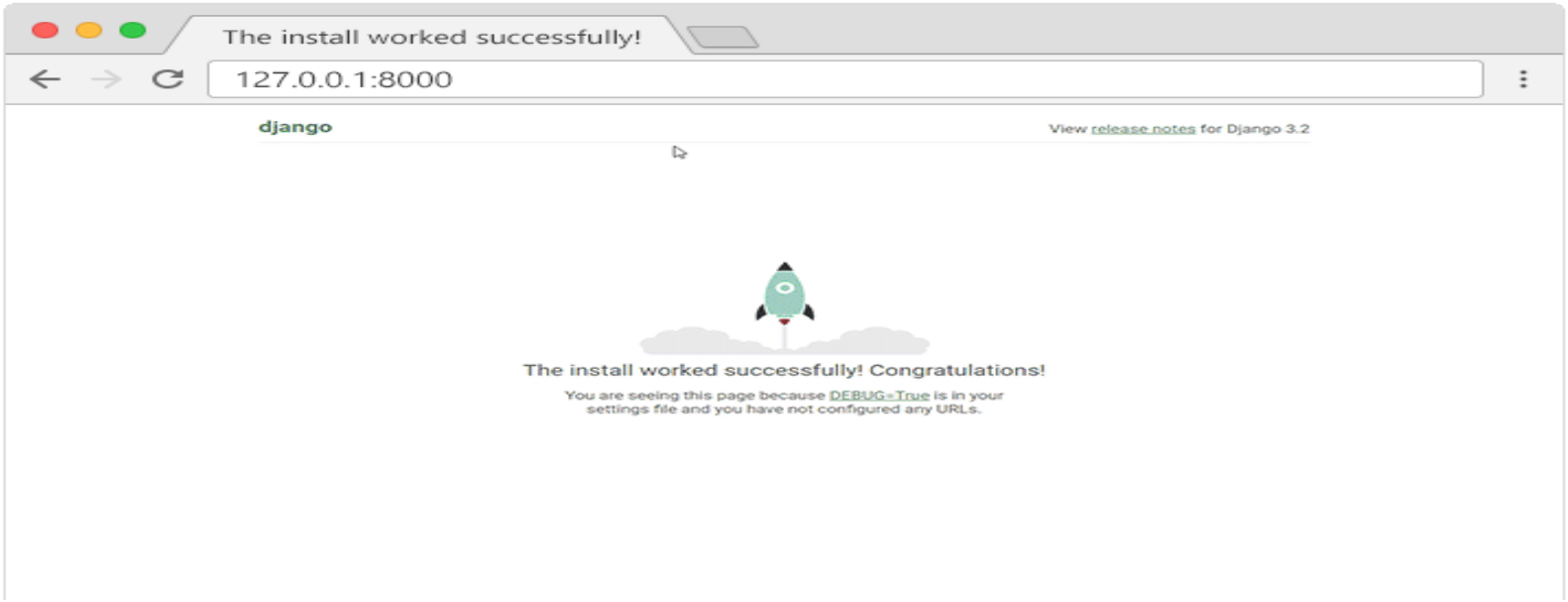
# Phases of Django Project Creation

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the
migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
October 27, 2022 - 13:03:14
Django version 4.1.2, using settings 'my_tennis_club.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

# Phases of Django Project Creation

Open a new browser window and type `127.0.0.1:8000` in the address bar.

The result:



Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Creation of App and their structure

## What is an App?

An app is a web application that has a specific meaning in your project, like a home page, a contact form, or a members database.

In this tutorial we will create an app that allows us to list and register members in a database.

But first, let's just create a simple Django app that displays "Hello World!".

## Create App

I will name my app `members`.

Start by navigating to the selected location where you want to store the app, in my case the `my_tennis_club` folder, and run the command below.

If the server is still running, and you are not able to write commands, press [CTRL] [BREAK], or [CTRL] [C] to stop the server and you should be back in the virtual environment.

```
py manage.py startapp members
```

# Creation of App and their structure

Django creates a folder named `members` in my project, with this content:

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        migrations/
            __init__.py
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

These are all files and folders with a specific meaning. You will learn about most of them later in this tutorial.

# Creation of App and their structure

## Views

Django views are Python functions that take http requests and return http response, like HTML documents.

A web page that uses Django is full of views with different tasks and missions.

Views are usually put in a file called `views.py` located on your app's folder.

There is a `views.py` in your `members` folder that looks like this:

`my_tennis_club/members/views.py` :

```
from django.shortcuts import render

# Create your views here.
```

# Creation of App and their structure

Find it and open it, and replace the content with this:

my_tennis_club/members/views.py :

```
from django.shortcuts import render
from django.http import HttpResponse

def members(request):
    return HttpResponse("Hello world!")
```

**Note:** The name of the view does not have to be the same as the application.

I call it `members` because I think it fits well in this context.

This is a simple example on how to send a response back to the browser.

# Creation of App and their structure

## URLs

Create a file named `urls.py` in the same folder as the `views.py` file, and type this code in it:

`my_tennis_club/members/urls.py` :

```
from django.urls import path
from . import views

urlpatterns = [
    path('members/', views.members, name='members'),
]
```

The `urls.py` file you just created is specific for the `members` application. We have to do some routing in the root directory `my_tennis_club` as well. This may seem complicated, but for now, just follow the instructions below.

# Creation of App and their structure

There is a file called `urls.py` on the `my_tennis_club` folder, open that file and add the `include` module in the `import` statement, and also add a `path()` function in the `urlpatterns[]` list, with arguments that will route users that comes in via `127.0.0.1:8000/`.

Then your file will look like this:

`my_tennis_club/my_tennis_club/urls.py`:

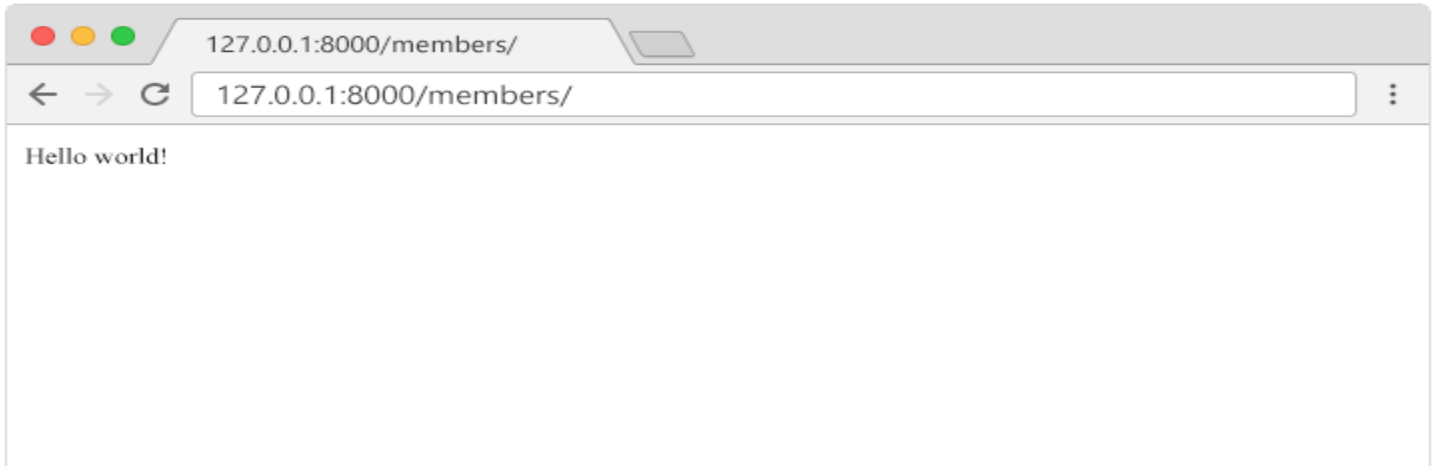
```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('', include('members.urls')),
    path('admin/', admin.site.urls),
]
```

# Creation of App and their structure

If the server is not running, navigate to the `/my_tennis_club` folder and execute this command in the command prompt:

```
py manage.py runserver
```

In the browser window, type `127.0.0.1:8000/members/` in the address bar.



# Creation of App and their structure

## Templates

In the Django Intro page, we learned that the result should be in HTML, and it should be created in a template, so let's do that.

Create a `templates` folder inside the `members` folder, and create a HTML file named `myfirst.html`.

The file structure should be like this:

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        templates/
            myfirst.html
```

# Creation of App and their structure

Open the HTML file and insert the following:

my_tennis_club/members/templates/myfirst.html :

```
<!DOCTYPE html>
<html>
<body>

<h1>Hello World!</h1>
<p>Welcome to my first Django project!</p>

</body>
</html>
```

# Creation of App and their structure

## Modify the View

Open the `views.py` file and replace the `members` view with this:

`my_tennis_club/members/views.py`:

```
from django.http import HttpResponse
from django.template import loader

def members(request):
  template = loader.get_template('myfirst.html')
  return HttpResponse(template.render())
```

# Creation of App and their structure

## Change Settings

To be able to work with more complicated stuff than "Hello World!", We have to tell Django that a new app is created.

This is done in the `settings.py` file in the `my_tennis_club` folder.

# Creation of App and their structure

Look up the `INSTALLED_APPS[]` list and add the `members` app like this:

my_tennis_club/my_tennis_club/settings.py :

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'members'
]
```

Then run this command:

```
py manage.py migrate
```

# Creation of App and their structure

Which will produce this output:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
```

# Creation of App and their structure

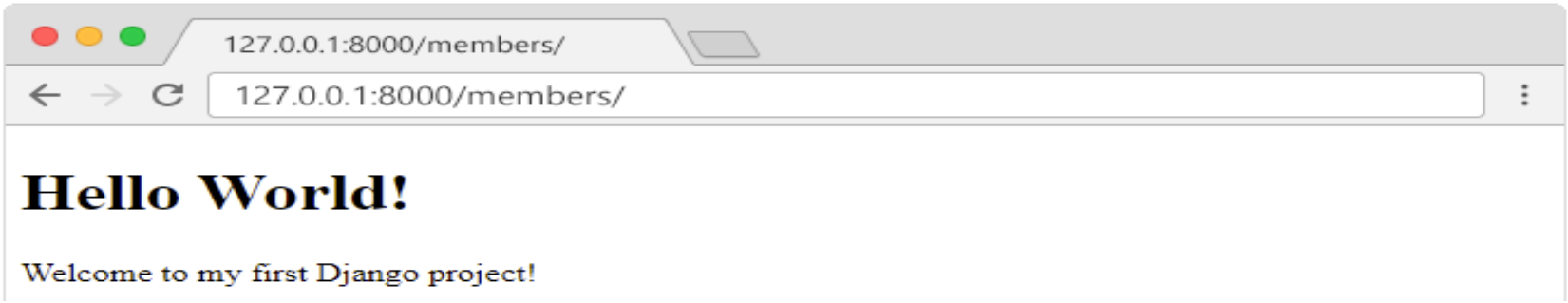
```
(myworld) C:\Users\Your Name\myworld\my_tennis_club>
```

Start the server by navigating to the `/my_tennis_club` folder and execute this command:

```
py manage.py runserver
```

In the browser window, type `127.0.0.1:8000/members/` in the address bar.

The result should look like this:



Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Creation of App and their structure

A Django model is a table in your database.

## Django Models

Up until now in this tutorial, output has been static data from Python or HTML templates.

Now we will see how Django allows us to work with data, without having to change or upload files in the process.

In Django, data is created in objects, called Models, and is actually tables in a database.

# Creation of App and their structure

## Create Table (Model)

To create a model, navigate to the `models.py` file in the `/members/` folder.

Open it, and add a `Member` table by creating a `Member` `class`, and describe the table fields in it:

`my_tennis_club/members/models.py` :

```
from django.db import models

class Member(models.Model):
  firstname = models.CharField(max_length=255)
  lastname = models.CharField(max_length=255)
```

The first field, `firstname`, is a Text field, and will contain the first name of the members.

The second field, `lastname`, is also a Text field, with the member's last name.

Both `firstname` and `lastname` is set up to have a maximum of 255 characters.

# Creation of App and their structure

## SQLite Database

When we created the Django project, we got an empty SQLite database.

It was created in the `my_tennis_club` root folder, and has the filename `db.sqlite3`.

By default, all Models created in the Django project will be created as tables in this database.

## Migrate

Now when we have described a Model in the `models.py` file, we must run a command to actually create the table in the database.

# Creation of App and their structure

Navigate to the `/my_tennis_club/` folder and run this command:

```
py manage.py makemigrations members
```

Which will result in this output:

```
Migrations for 'members':
  members\migrations\0001_initial.py
    - Create model Member

(myworld) C:\Users\Your Name\myworld\my_tennis_club>
```

# Creation of App and their structure

Django creates a file describing the changes and stores the file in the `/migrations/` folder:

`my_tennis_club/members/migrations/0001_initial.py`:

```
# Generated by Django 4.1.2 on 2022-10-27 11:14

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]
```

# Creation of App and their structure

```
operations = [
    migrations.CreateModel(
        name='Member',
        fields=[
            ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verb
            ('firstname', models.CharField(max_length=255)),
            ('lastname', models.CharField(max_length=255)),
        ],
    ),
]
```

Note that Django inserts an `id` field for your tables, which is an `auto increment number` (first record gets the value 1, the second record 2 etc.), this is the default behavior of Django, you can override it by describing your own `id` field.

The table is not created yet, you will have to run one more command, then Django will create and execute an SQL statement, based on the content of the new file in the `/migrations/` folder.

# Creation of App and their structure

Run the migrate command:

```
py manage.py migrate
```

Which will result in this output:

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

(myworld) C:\Users\Your Name\myworld\my_tennis_club>
```

Now you have a `Member` table in you database!

# Creation of App and their structure

## View SQL

As a side-note: you can view the SQL statement that were executed from the migration above. All you have to do is to run this command, with the migration number:

```
py manage.py sqlmigrate members 0001
```

Which will result in this output:

```
BEGIN;
--
-- Create model Member
--
CREATE TABLE "members_member" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "firstname"
varchar(255) NOT NULL, "lastname" varchar(255) NOT NULL); COMMIT;
```

# Introduction to Admin Console

## Django Admin

Django Admin is a really great tool in Django, it is actually a CRUD* user interface of all your models!

*CRUD stands for Create Read Update Delete.

It is free and comes ready-to-use with Django:

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Introduction to Admin Console

Django administration

WELCOME, JOHNDOE. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Members › Memberss

Select members to change

ADD MEMBERS +

Action: --------- Go 0 of 5 selected

| | FIRSTNAME | LASTNAME | JOINED DATE |
|---|---|---|---|
| ☐ | Stalikken | Refsnes | - |
| ☐ | Lene | Refsnes | - |
| ☐ | Linus | Refsnes | - |
| ☐ | Tobias | Refsnes | - |
| ☐ | Emil | Refsnes | Jan. 5, 2022 |

5 memberss

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Introduction to Admin Console

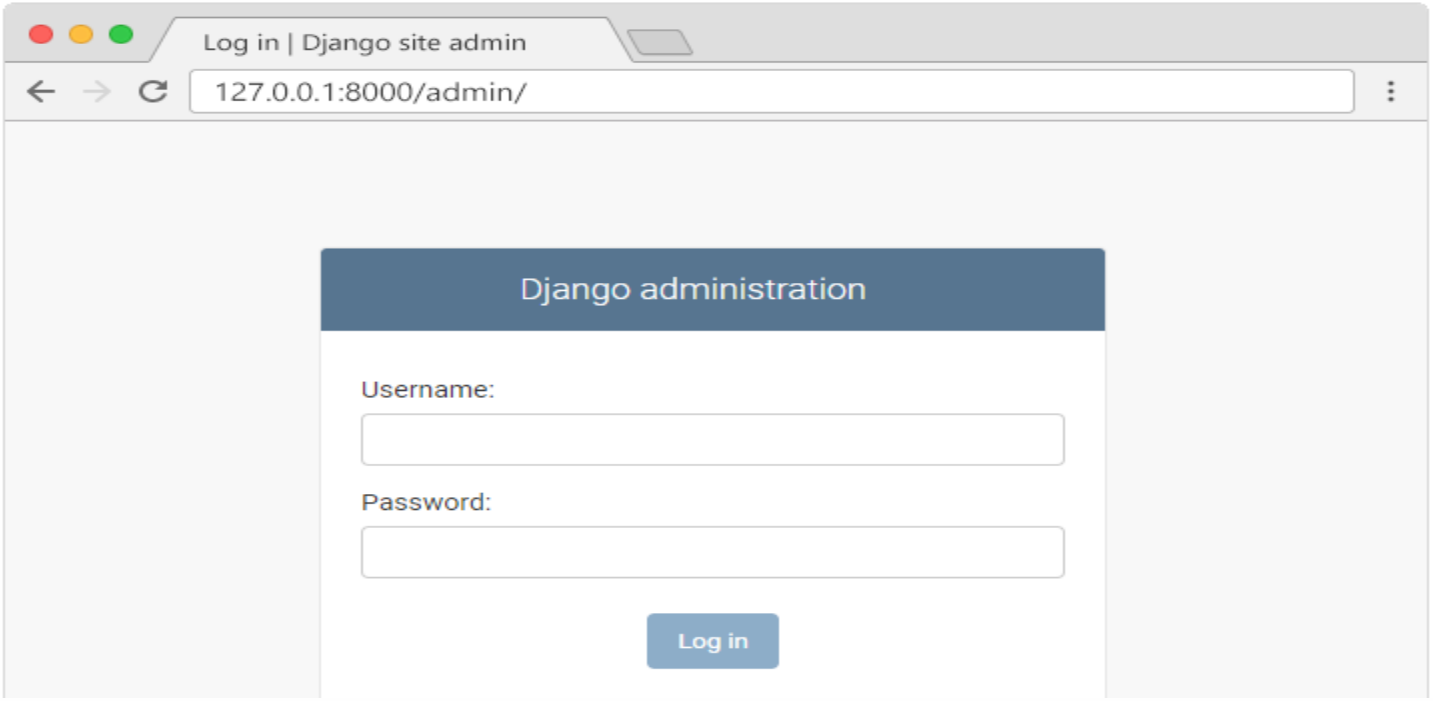
To enter the admin user interface, start the server by navigating to the `/myworld` folder and execute this command:

```
py manage.py runserver
```

# Introduction to Admin Console

In the browser window, type `127.0.0.1:8000/admin/` in the address bar.

The result should look like this:



# Introduction to Admin Console

The reason why this URL goes to the Django admin log in page can be found in the `urls.py` file of your project:

`my_tennis_club/my_tennis_club/urls.py`:

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('', include('members.urls')),
    path('admin/', admin.site.urls),
]
```

# Introduction to Admin Console

The `urlpatterns[]` list takes requests going to `admin/` and sends them to `admin.site.urls`, which is part of a built-in application that comes with Django, and contains a lot of functionality and user interfaces, one of them being the log-in user interface.

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Introduction to Admin Console

## Create User

To be able to log into the admin application, we need to create a user.

This is done by typing this command in the command view:

```
py manage.py createsuperuser
```

Which will give this prompt:

```
Username:
```

# Introduction to Admin Console

Here you must enter: username, e-mail address, (you can just pick a fake e-mail address), and password:

```
Username: johndoe
Email address: johndoe@dummymail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]:
```

My password did not meet the criteria, but this is a test environment, and I choose to create user anyway, by enter y:

```
Bypass password validation and create user anyway? [y/N]: y
```

# Introduction to Admin Console

If you press [Enter], you should have successfully created a user:

```
Superuser created successfully.
```

Now start the server again:

```
py manage.py runserver
```

In the browser window, type `127.0.0.1:8000/admin/` in the address bar.

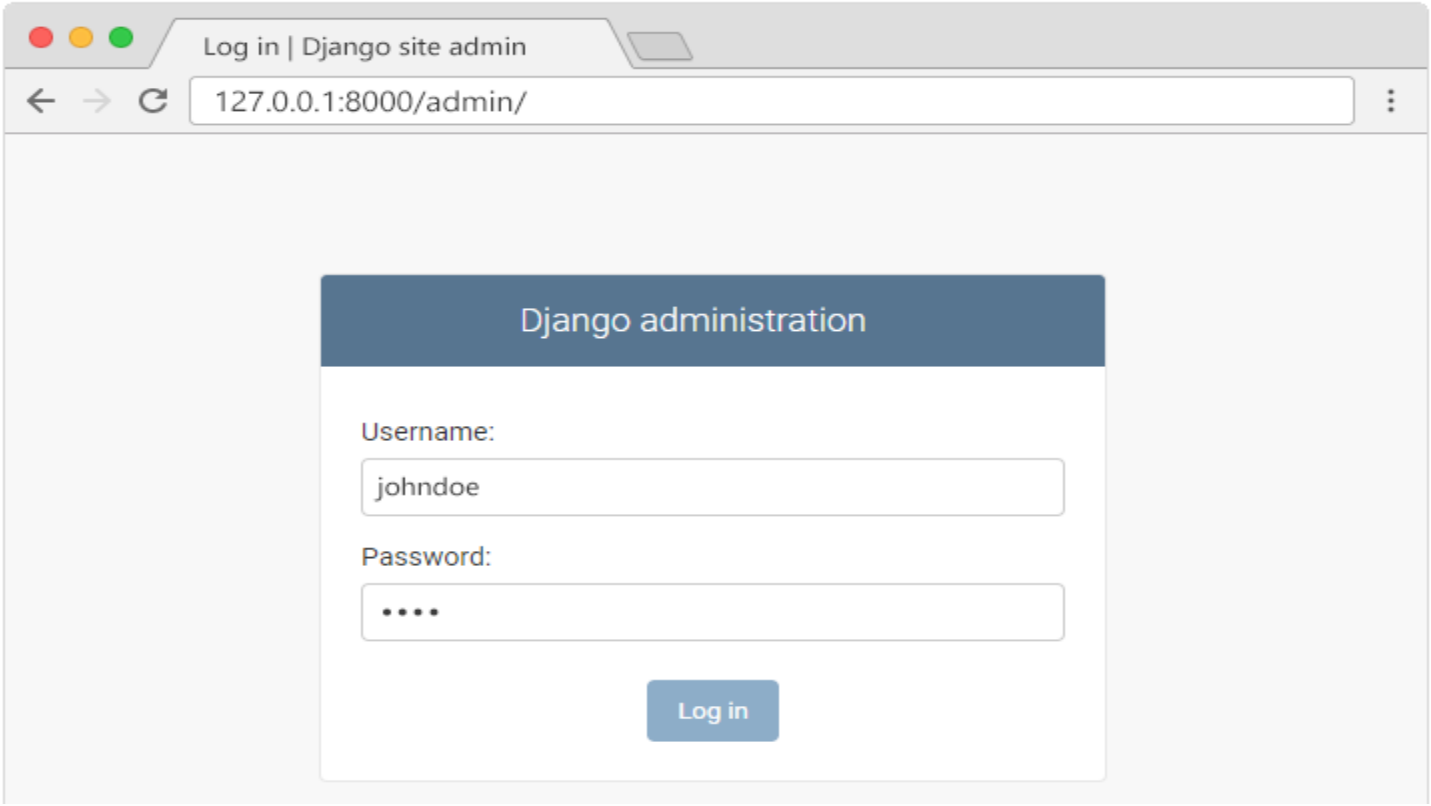
And fill in the form with the correct username and password:

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Introduction to Admin Console



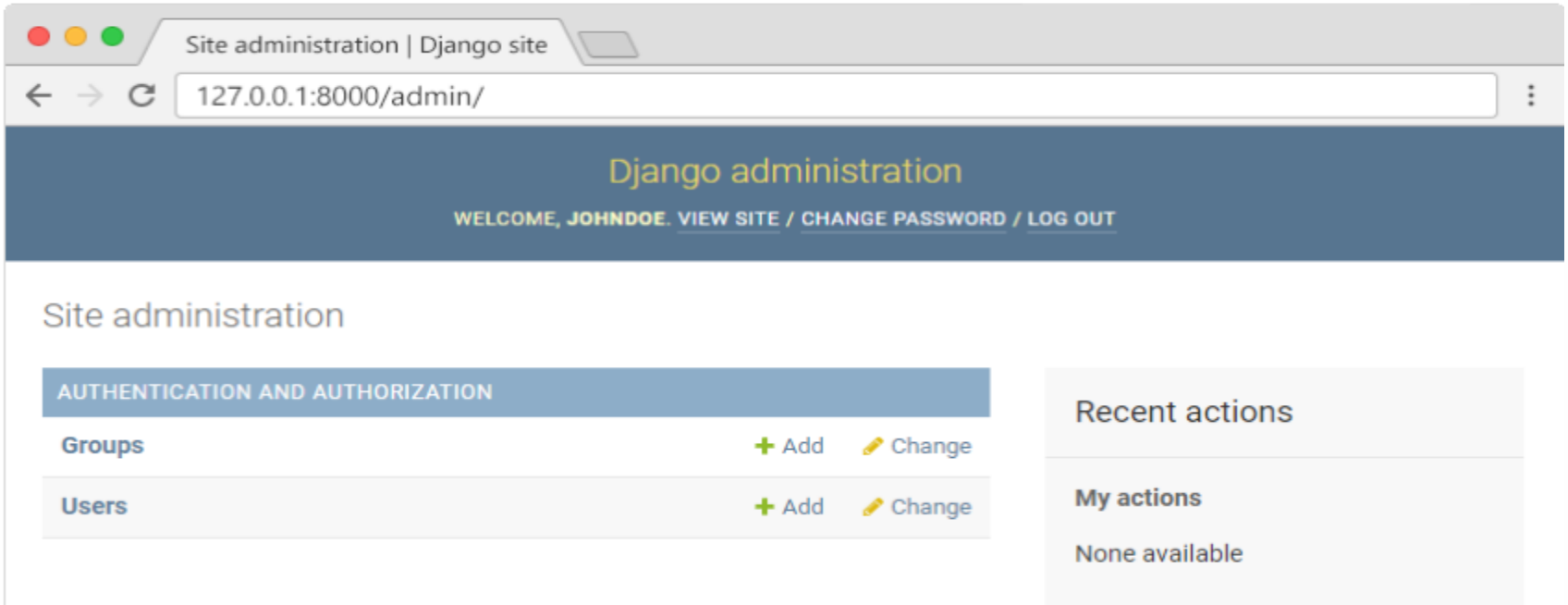
Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Introduction to Admin Console

Which should result in this user interface:



Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Creation of URL Mapping

## Creating Url Patterns in Django

In Django, URL patterns are the fundamental concept that is used to map URLs (Uniform Resource Locators) to views, by which we can specify how different URLs in your web application should be handled. URL patterns are defined in your Django project's urls.py file. Let us understand with this example:

To create a URL Pattern we should

1. open the urls.py file in our Django app
2. Import the path() function from django.urls.
3. Define a list of URL patterns

# Creation of URL Mapping

In this example, we've defined two URL patterns, '/home/' and '/about/', and specified which view functions should handle these URLs.

Python

```python
from django.urls import path
from . import views

urlpatterns = [
    path('home/', views.home_view),
    path('about/', views.about_view),
]
```

# Creation of URL Mapping

### Using Regular Expression Captures in Django URL Patterns

If we want to extract the value from urls then pass it to views than we can do it with the help of regular expression captures .The extracted value then can be used as according to the need in the view function.

To use the Regular Expression capture follow the following steps:

1. Define the regular expression in url pattern and add capturing groups by the help of **'()'** to collect values from the pattern.
2. In the view function, include for each capturing group.
3. When the url pattern is matched ,the matched view function is called.

In below code the 3rd URL pattern is dynamic and captures an integer value called blog_id from the URL. For example, if you access http://yourdomain.com/blog/1/, it will display the detail page for the blog post with an ID of 1. The name of this URL pattern is 'blog_detail'.

# Creation of URL Mapping

Python

```python
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('blog/<int:blog_id>/', views.blog_detail, name='blog_detail'),

]
```

**views.py:** The view.py code for the above urls is shown below. In the below code **'home'** is a basic view function. The **'blog_detail'** is a view function in which the value captured value from url is passed as argument here. The argument here is **'blog_id'** . This can be used inside our view function according to the need.

# Creation of URL Mapping

Python

```python
from django.shortcuts import render
from django.http import HttpResponse


def home(request):
    return HttpResponse("Welcome to our website!")

def blog_detail(request, blog_id):
    blog_post = {'id': blog_id, 'title': 'Sample Blog Post', 'content': 'This is the content of the blog post.'}
    context = {'blog_post': blog_post}
    return render(request, 'blog_detail.html', context)
```

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Creation of URL Mapping

### Naming URL Patterns in Django

We can name our url pattern in Django so that it makes our work easy while refering to them in our code and html files.

To give a name to the url pattern we should follow the following steps:

1. In the path function specify the name ,with a string value.
2. In the templates, use the {% url %} and in the code use the reverse() function to refer to the named URL pattern.

Let us understand with this example:

In this Django URL configuration, we have assigned names to two URL patterns: **'home'**, **'about'.** These names help us refer to these URLs in our code, making it more readable and maintainable.

# Creation of URL Mapping

Python

```python
from django.urls import path
from . import views

urlpatterns = [
    path('home/', views.home, name='home'),
    path('about/', views.about, name='about'),
]
```

# Creation of URL Mapping

**Use in HTML template**

We can use the name of the url pattern as :

```
{% url 'home' %}
```

For Example:

HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h2>Welcome To GFG</h2>
<a href="{% url 'home' %}">HOME</a>
</body>
</html>
```

# Creation of URL Mapping

**Use in HTML template**

We can use the name of the url pattern as :

```
{% url 'home' %}
```

For Example:

HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h2>Welcome To GFG</h2>
<a href="{% url 'home' %}">HOME</a>
</body>
</html>
```

# Static File

## Create Static Folder

When building web applications, you probably want to add some static files like images or css files.

Start by creating a folder named `static` in your project, the same place where you created the `templates` folder:

The name of the folder has to be `static`.

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        templates/
        static/
```

Add a CSS file in the `static` folder, the name is your choice, we will call it `myfirst.css` in this example:

# Static File

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        templates/
        static/
            myfirst.css
```

Open the CSS file and insert the following:

`my_tennis_club/members/static/myfirst.css`:

```
body {
  background-color: lightblue;
  font-family: verdana;
}
```

# Static File

## Modify the Template

Now you have a CSS file, with some CSS styling. The next step will be to include this file in a HTML template:

Open the HTML file and add the following:

```
{% load static %}
```

And:

```
<link rel="stylesheet" href="{% static 'myfirst.css' %}">
```

# Static File

my_tennis_club/members/templates/template.html :

```
{% load static %}
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="{% static 'myfirst.css' %}">
<body>

{% for x in fruits %}
  <h1>{{ x }}</h1>
{% endfor %}

</body>
</html>
```

# Static File

Restart the server for the changes to take effect:

```
py manage.py runserver
```

And check out the result in your own browser: `127.0.0.1:8000/testing/`.

## Didn't Work?

**Just testing?** If you just want to play around, and not going to deploy your work, you can set `DEBUG = True` in the `settings.py` file, and the example above will work.

**Plan to deploy?** If you plan to deploy your work, you should set `DEBUG = False` in the `settings.py` file. The example above will fail, because Django has no built-in solution for serving static files, but there are other ways to serve static files, you will learn how in the next chapter.

# Static File

## Handle Static Files

Static files in your project, like stylesheets, JavaScripts, and images, are not handled automatically by Django when `DEBUG = False`.

When `DEBUG = True`, this worked fine, all we had to do was to put them in the `static` folder of the application.

When `DEBUG = False`, static files have to be collected and put in a specified folder before we can use it.

## Collect Static Files

To collect all necessary static files for your project, start by specifying a `STATIC_ROOT` property in the `settings.py` file.

This specifies a folder where you want to collect your static files.

You can call the folder whatever you like, we will call it `productionfiles`:

# Static File

my_tennis_club/my_tennis_club/settings.py :

```
.
.

STATIC_ROOT = BASE_DIR / 'productionfiles'

STATIC_URL = 'static/'

.
.
```

You could manually create this folder and collect and put all static files of your project into this folder, but Django has a command that do this for you:

```
py manage.py collectstatic
```

Which will produce this result:

# Static File

```
131 static files copied to 'C:\Users\your_name\myworld\my_tennis_club\productionfiles'.
```

131 files? Why so many? Well this is because of the admin user interface, that comes built-in with Django. We want to keep this feature in production, and it comes with a whole bunch of files including stylesheets, fonts, images, and JavaScripts.

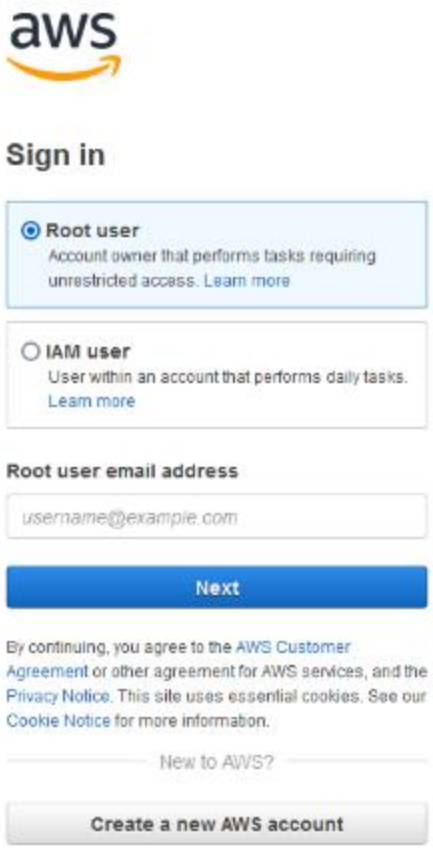
```
my_tennis_club
    members/
    my_tennis_club/
    productionfiles/
        admin/
        myfirst.css
```

# Deploy Django

There are many providers out there that offers servers for Django projects. In this tutorial we will use the Amazon Web Services (AWS) platform, mainly because they offer a free solution that only requires you to create an AWS account.

**Note:** you can choose whatever server provider you like, they will all give you the same result, but they will have some provider-specific settings that you should be aware of when following this tutorial.

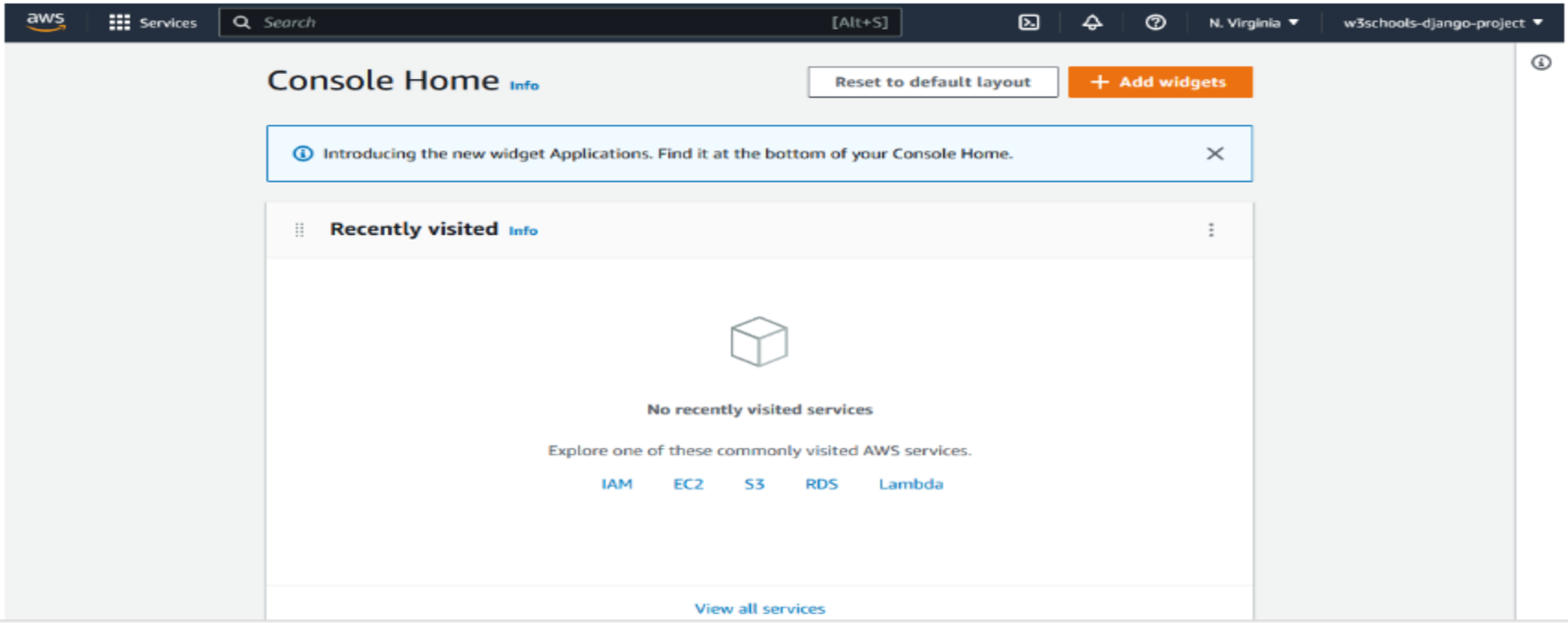
# Deploy Django




# Deploy Django

## AWS Console

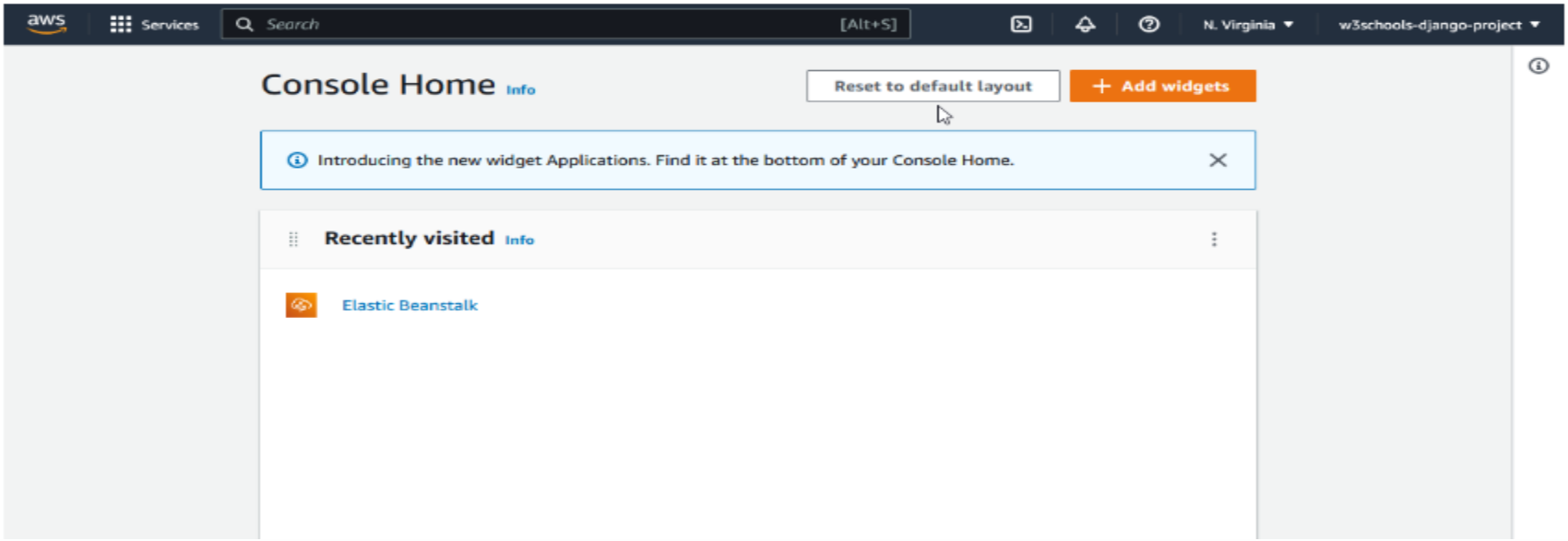
Once you have signed in, you should be directed to the AWS Console Home page:



# Deploy Django

We will be using a service called "Elastic Beanstalk" to deploy the Django project.

In the search field at the top, search for "elastic beanstalk", and click to start the service:



# Deploy Django

## Lock in Dependencies

When you create a Django application, there are some Python packages that your project depends on.

Django itself is a Python package, and we have to make sure that the server where we deploy our project also has the Django package installed, and all the other packages your project requires.

Luckily there is a command for this as well, just run this command in the command view:

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

The result of the above command, is a file called `requirements.txt` being created in the project:

# Deploy Django

```
my_tennis_club
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    requirements.txt
```

The file contains all the packages that this project depends on: with this content:

# Deploy Django

`my_tennis_club/requirements.txt` :

```
asgiref==3.5.2
Django==4.1.4
psycopg2-binary==2.9.5
sqlparse==0.4.3
tzdata==2022.7
whitenoise==6.2.0
```

**Note:** You can create this file on your own, and insert the packages manually, just make sure you get all the packages your project depends on, and you must name the file `requirements.txt`.

Now the hosting provider knows which packages to install when we deploy our project.

# Deploy Django

## Provider-Specific Settings

We have chosen AWS as our hosting provider, and Elastic Beanstalk as a service to deploy the Django project, and it has some specific requirements.

## .ebextension Folder

It requires that you create a folder on the root level of your project called `.ebextensions`:

```
my_tennis_club
    .ebextensions/
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    requirements.txt
```

# Deploy Django

## Create django.config File

In the `.ebextensions` folder, create a file called `django.config`:

```
my_tennis_club
    .ebextensions/
        django.config
```

Open the file and insert these settings:

`my_tennis_club/.ebextensions/django.config`:

```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: my_tennis_club.wsgi:application
```

# Deploy Django

## Zip Your Project

To wrap your project into a .zip file, you cannot zip the entire project folder, but choose the files and folders manually.

The files to include in the .zip file are highlighted (blue) in the example below:

```
my_tennis_club
    .ebextensions/
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    requirements.txt
```

With your file explorer, navigate to the project folder, select these files and folders, right-click and choose to create a zip file.

# Deploy Django

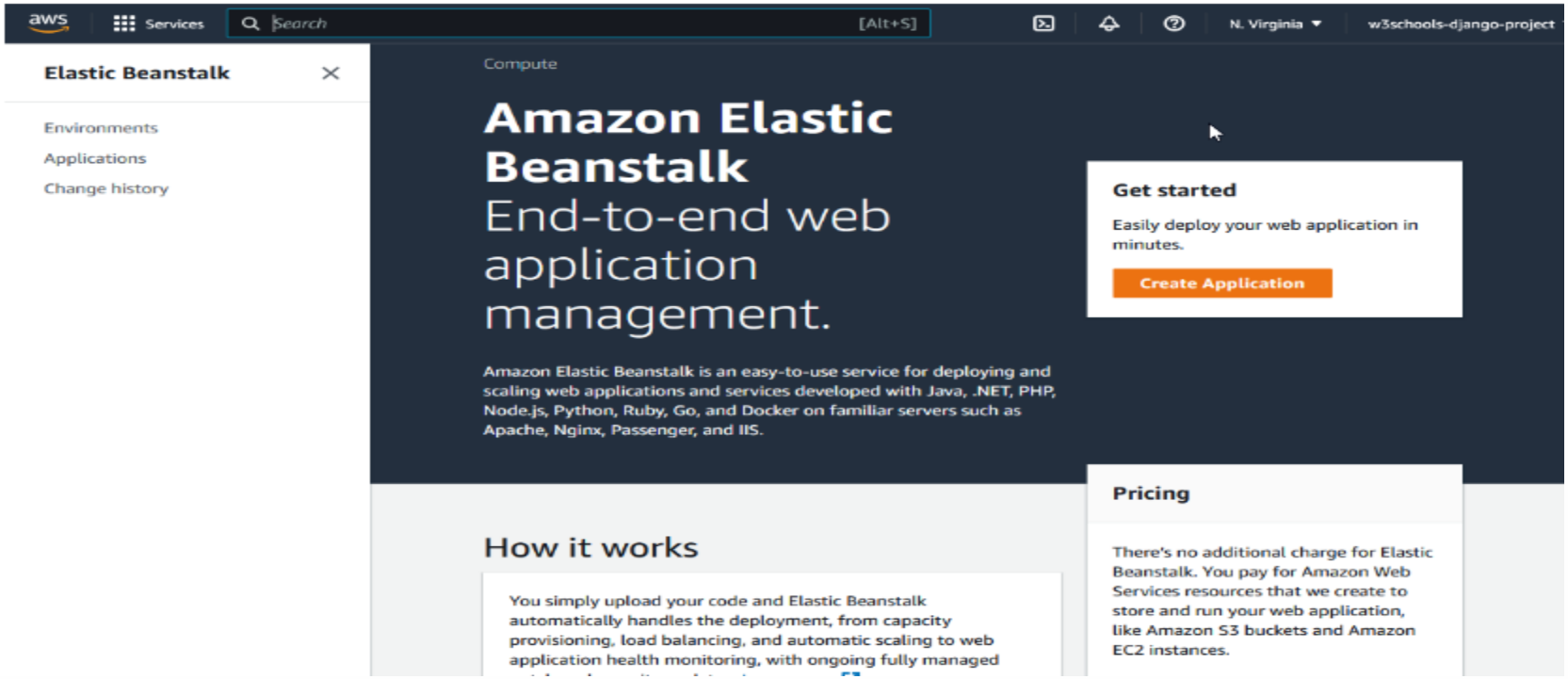
## Zip File

Now you have a .zip file of your project which you can upload to Elastic beanstalk:

```
my_tennis_club
    .ebextensions/
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    my_tennis_clup.zip
    requirements.txt
```

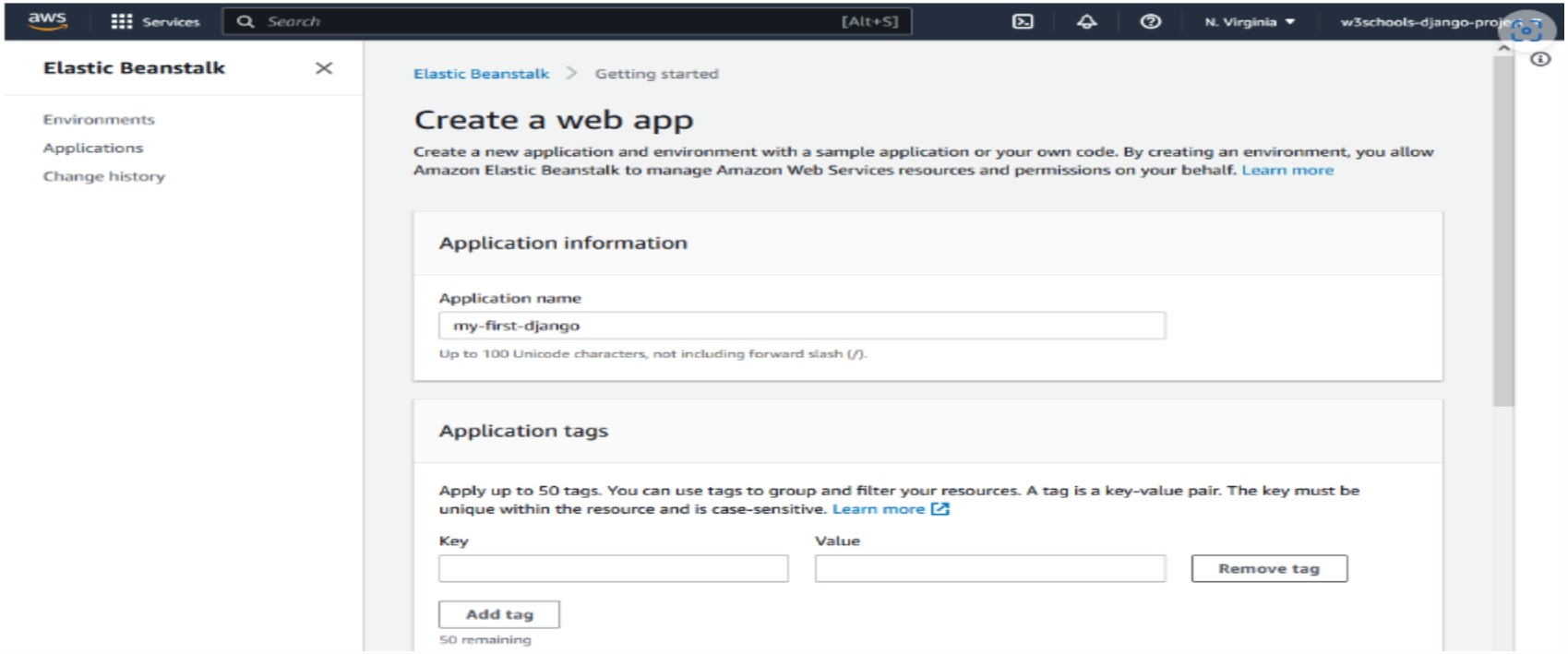
# Deploy Django

In AWS, navigate to the Elastic Beanstalk application, as we did in the Choose Provider chapter, and click the "Create application" button:



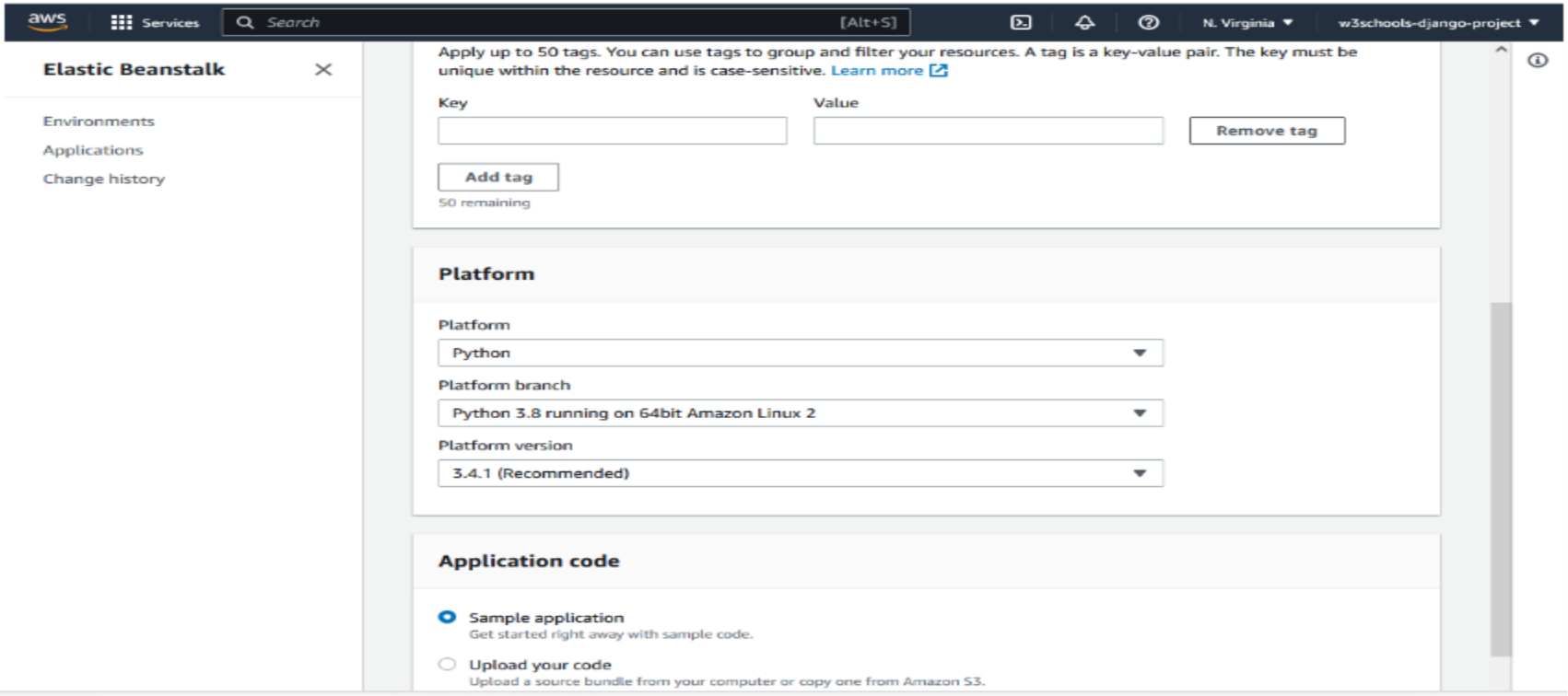
# Deploy Django

Once you have clicked the "Create Application" button, you will be taken to this page, where you can give your Django project a name. I will name it "my-first-django":



# Deploy Django

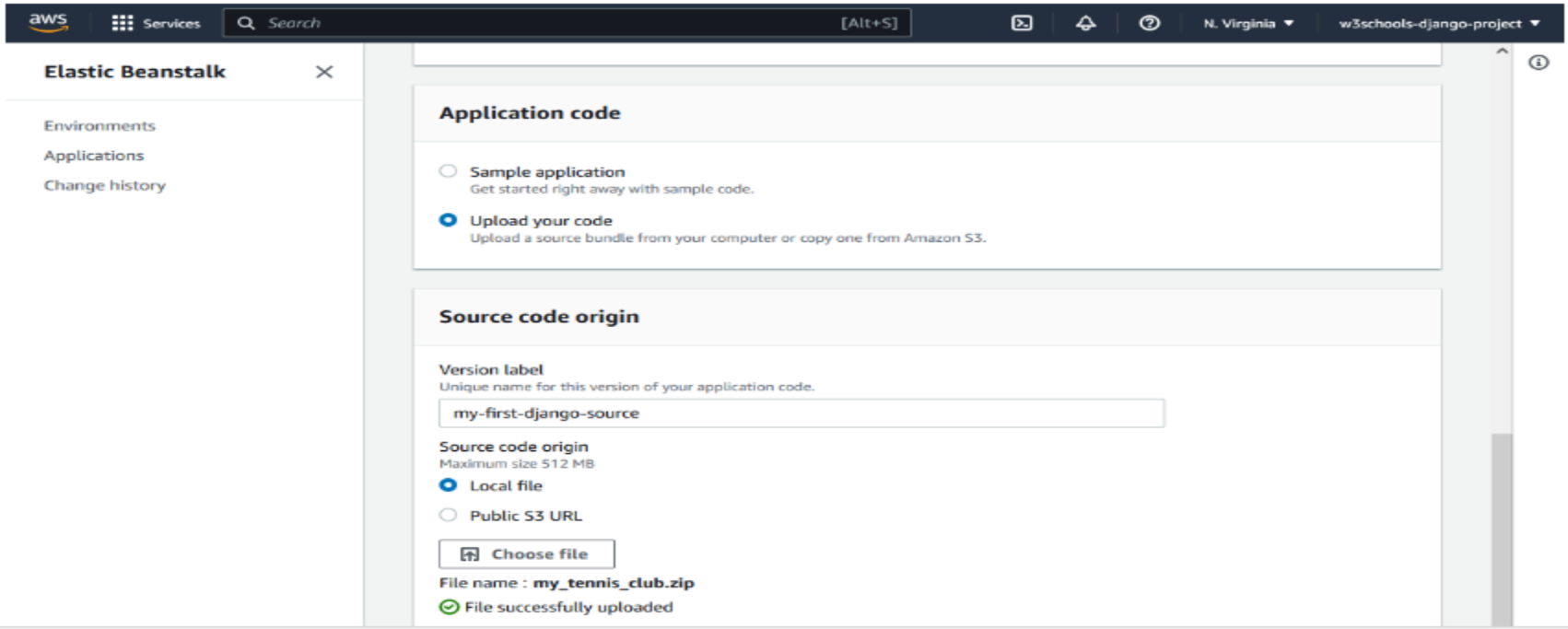
Then scroll down until you see the "Platform" section, and choose "Python", with the recommended version:



# Deploy Django

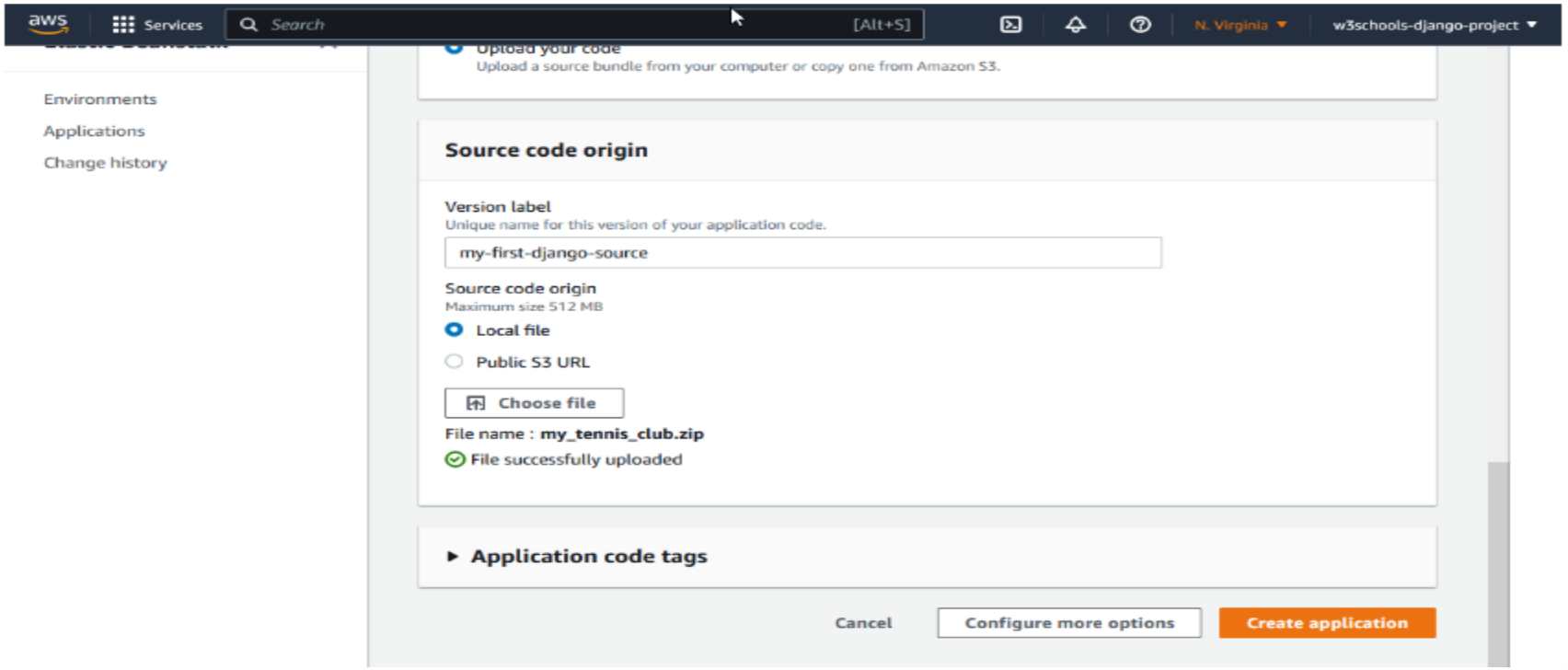
Next, scroll down to the next section, the "Application code" section, and choose "Upload your code".

Click on the "Choose file" button, navigate to the .zip file you created in the previous chapter and upload it:



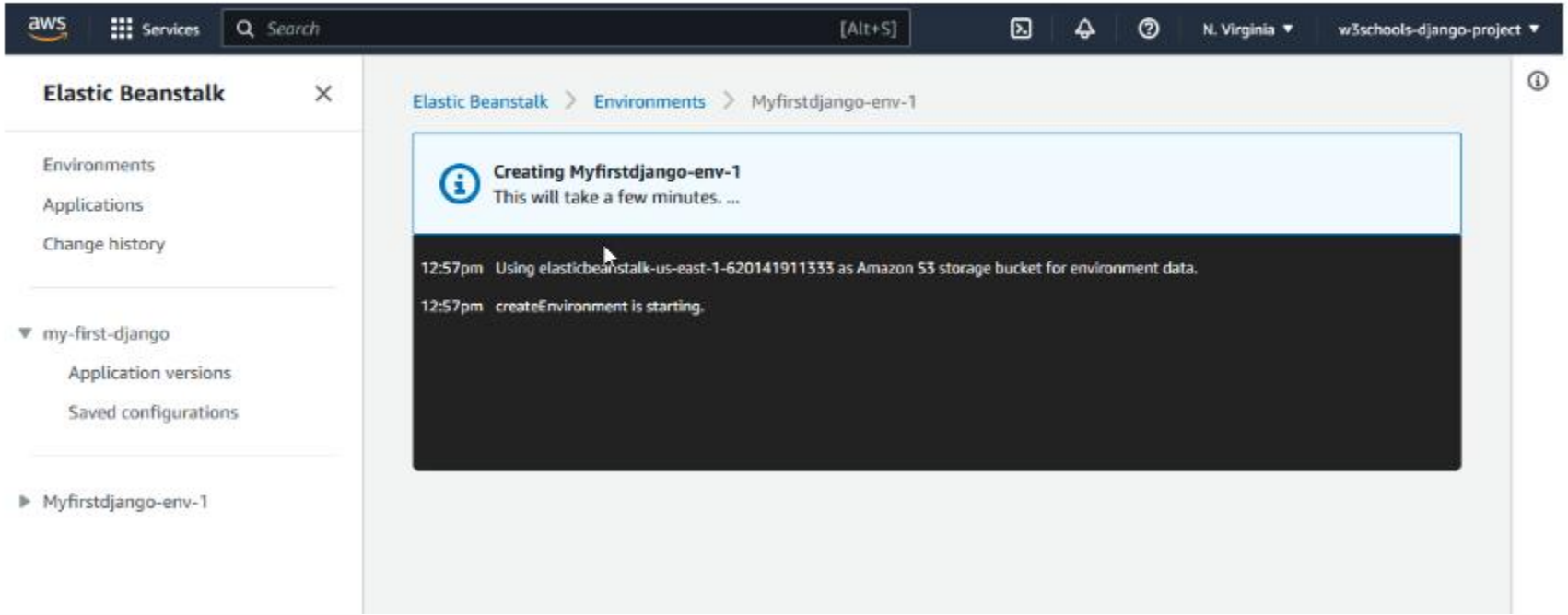
# Deploy Django

Click the "Create application" button to start deploying.



# Deploy Django

The deployment will take a few minutes.

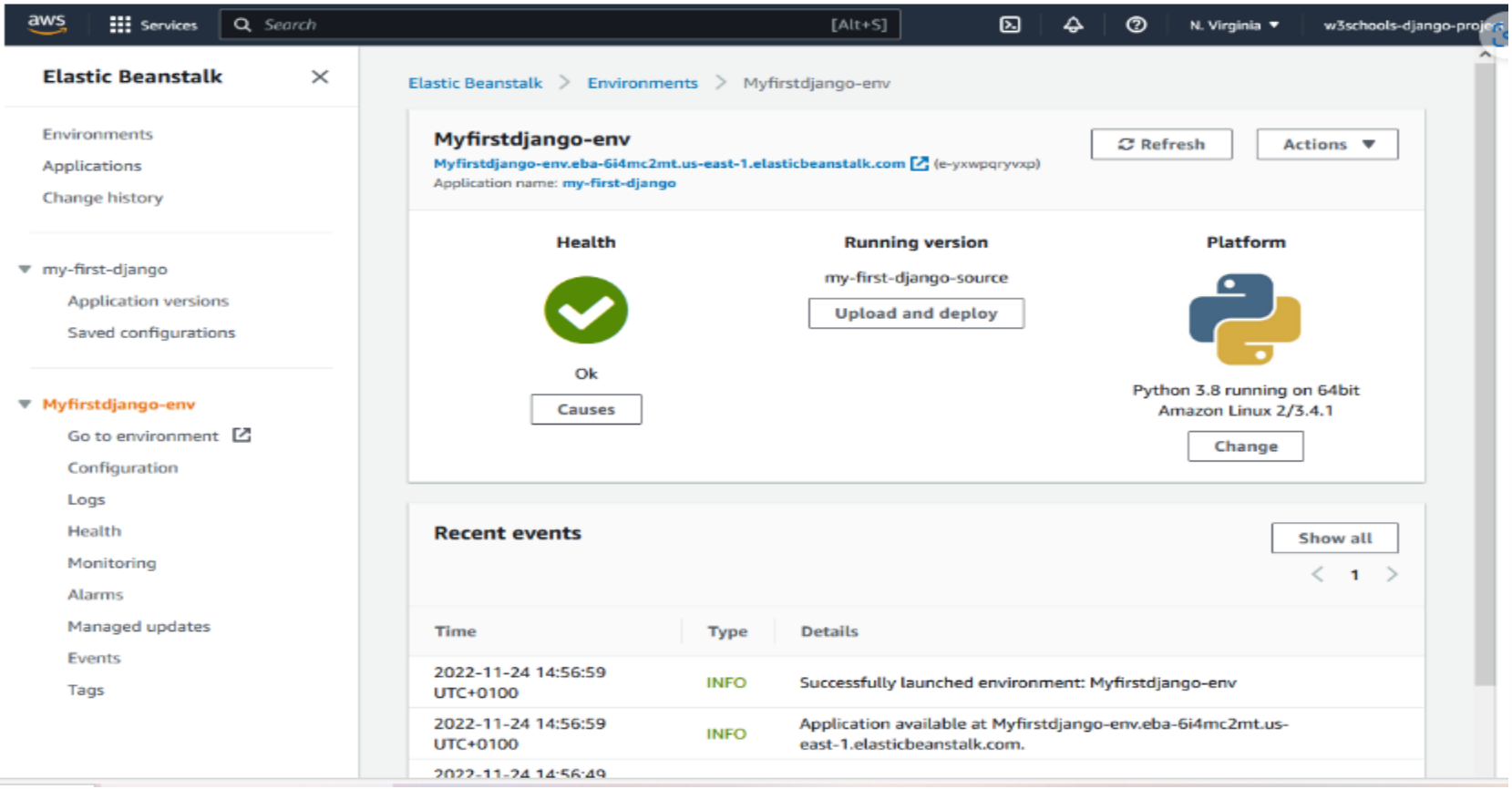


Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# Deploy Django



Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

# THANK YOU

# DIGITAL LEARNING CONTENT


## Parul® University

www.paruluniversi