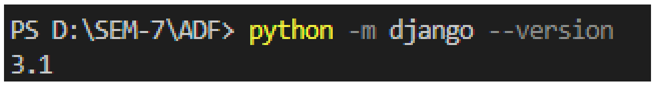
Practical-2

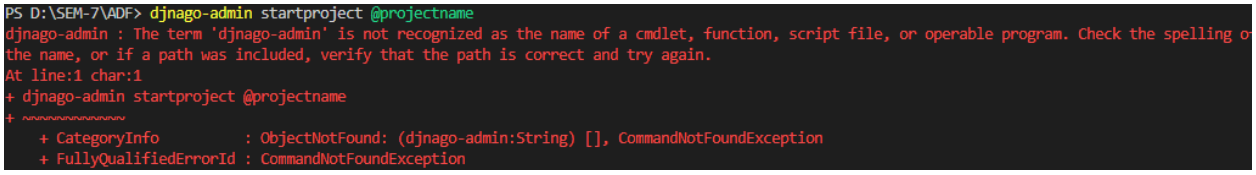
P-2: Hands-on for understanding Django Project Structure and Application Structure and prepare a document which describes importance and significance of each Django Project and Application structure files. Sample Hello world webpage to be created.

For checking django version:

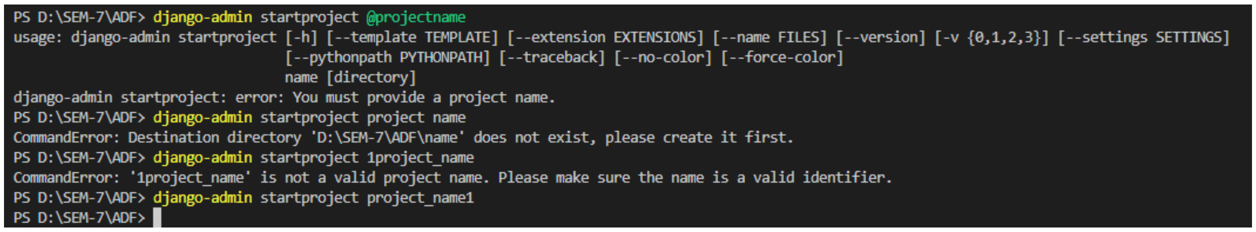


```
PS D:\SEM-7\ADF> python -m django --version
3.1
```

To make the project in django:



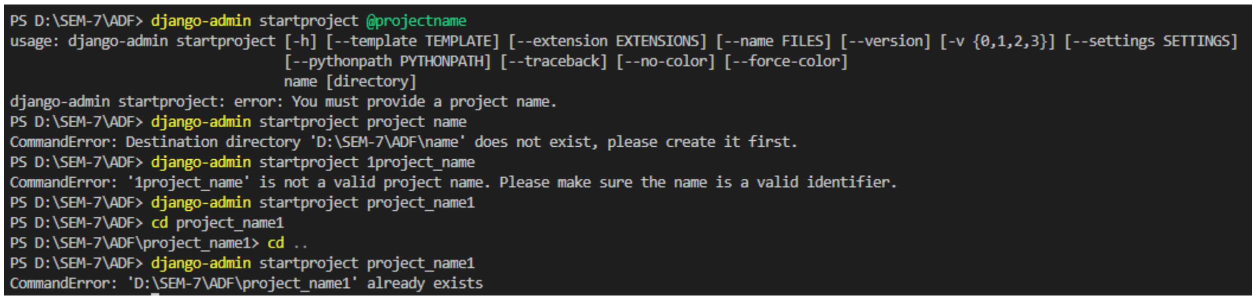
```
PS D:\SEM-7\ADF> djnago-admin startproject @projectname
djnago-admin : The term 'djnago-admin' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling o
the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ djnago-admin startproject @projectname
+ ~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (djnago-admin:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```



```
PS D:\SEM-7\ADF> django-admin startproject @projectname
usage: django-admin startproject [-h] [--template TEMPLATE] [--extension EXTENSIONS] [--name FILES] [--version] [-v {0,1,2,3}] [--settings SETTINGS]
                                 [--pythonpath PYTHONPATH] [--traceback] [--no-color] [--force-color]
                                 name [directory]
django-admin startproject: error: You must provide a project name.
PS D:\SEM-7\ADF> django-admin startproject project name
CommandError: Destination directory 'D:\SEM-7\ADF\name' does not exist, please create it first.
PS D:\SEM-7\ADF> django-admin startproject 1project_name
CommandError: '1project_name' is not a valid project name. Please make sure the name is a valid identifier.
PS D:\SEM-7\ADF> django-admin startproject project_name1
PS D:\SEM-7\ADF>
```

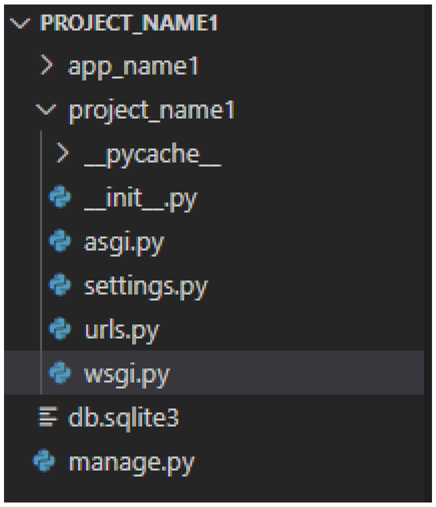
Delete: manually delete project folder, if using SQLite then database also removed this way

If using PostgreSQL then we have to delete databse manually.



```
PS D:\SEM-7\ADF> django-admin startproject @projectname
usage: django-admin startproject [-h] [--template TEMPLATE] [--extension EXTENSIONS] [--name FILES] [--version] [-v {0,1,2,3}] [--settings SETTINGS]
                                 [--pythonpath PYTHONPATH] [--traceback] [--no-color] [--force-color]
                                 name [directory]
django-admin startproject: error: You must provide a project name.
PS D:\SEM-7\ADF> django-admin startproject project name
CommandError: Destination directory 'D:\SEM-7\ADF\name' does not exist, please create it first.
PS D:\SEM-7\ADF> django-admin startproject 1project_name
CommandError: '1project_name' is not a valid project name. Please make sure the name is a valid identifier.
PS D:\SEM-7\ADF> django-admin startproject project_name1
PS D:\SEM-7\ADF> cd project_name1
PS D:\SEM-7\ADF\project_name1> cd ..
PS D:\SEM-7\ADF> django-admin startproject project_name1
CommandError: 'D:\SEM-7\ADF\project_name1' already exists
```

Django Project structure:



```
PROJECT_NAME1
  app_name1
  project_name1
    __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  db.sqlite3
  manage.py
```

<u>manage.py:</u>

Django-admin is command line utility, in project folder the same function is done by manage.py file.

<u>db.sqlite3 :</u>

the db.sqlite3 file in a Django project is a built-in SQLite database file that serves as the default database backend for the project. It stores the data for the project's models and allows you to interact with the database using Django's ORM system. Suitable for small projects, not for large-scale o production projects.

__init__.py :

Empty file which tells python that treate this folder as python package.

<u>asgi.py:</u>

introduced to support Asynchronous Server Gateway Interface (ASGI) applications. ASGI is a specification that allows Django to handle asynchronous web protocols and is crucial for building real-time applications, handling long-running requests, and working with WebSocket connections when deployed with ASGI servers like Daphne or Uvicorn.

<u>settings.py:</u>

Database Configuration: The settings.py file includes settings related to the database, such as the database engine, name, user, password, host, and port. These settings determine how Django interacts with the database and store data.

Installed Apps: This setting defines the list of installed Django applications for the project. Each application provides specific functionality to the project, and Django uses this list to know which apps to include.

Middleware: Middleware components are used to process requests and responses globally. The MIDDLEWARE setting contains a list of middleware classes that handle specific tasks such as authentication, security, and more.

Static and Media Files: The STATIC_URL, STATIC_ROOT, MEDIA_URL, and MEDIA_ROOT settings are used to configure the URL and location of static and media files (e.g., CSS, JavaScript, images, user uploads).

Template Configuration: This includes settings like TEMPLATES which define how Django handles templates, their location, engines, and more.

Internationalization: The LANGUAGE_CODE and TIME_ZONE settings determine the default language and time zone for the project.

Security and Debugging: The DEBUG setting controls whether Django is in debug mode or not, and SECRET_KEY stores the project's secret key for securing cookies and other sensitive data.

Authentication: Settings like AUTHENTICATION_BACKENDS, LOGIN_URL, LOGIN_REDIRECT_URL, and LOGOUT_REDIRECT_URL control how user authentication is handled.

Custom Settings: Developers can add custom settings specific to their project in the settings.py file to extend and tailor the behavior of the application.

Third-Party Libraries: If you use third-party libraries, their specific settings might also be included in settings.py.

urls.py:

used to define URL patterns and map them to views in your Django application. It plays a significant role in handling incoming requests and directing them to the appropriate view functions to generate responses.

In the urls.py file, you define URL patterns using Django's URL patterns syntax. URL patterns are regular expressions or strings that match the incoming URLs from the user's browser. When a user accesses a specific URL on your website, Django uses the urls.py file to find the corresponding view to handle that request.

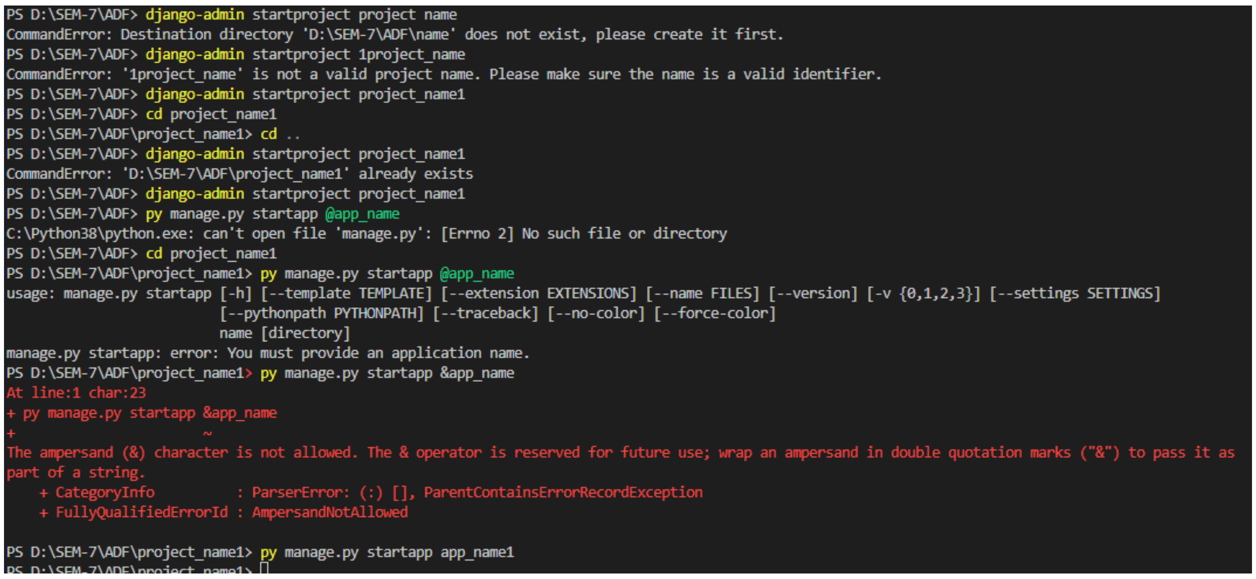
wsgi.py:

wsgi.py file is a Python script used to serve the application using the WSGI (Web Server Gateway Interface) protocol. WSGI is a standard interface between web servers and Python web applications, allowing the application to communicate with the server.

When you deploy your Django project to a WSGI server, such as Apache with mod_wsgi or Gunicorn, the server will use the application object from the wsgi.py file to run your Django application and handle incoming requests.

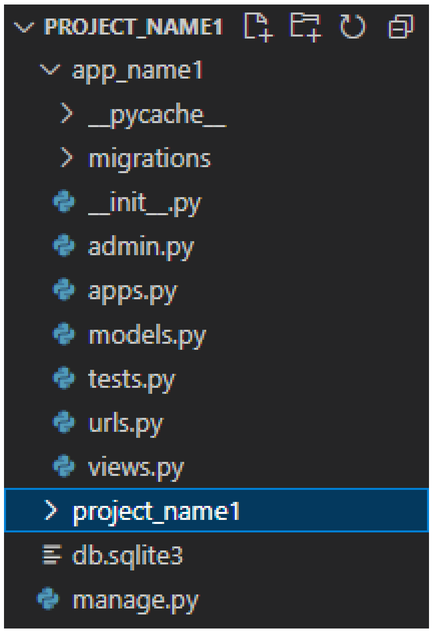
The wsgi.py file is crucial for deploying Django applications to production servers that support the WSGI protocol. It acts as a bridge between the web server and your Django application, allowing for smooth and efficient communication.

<u>**Django app structure:**</u>



```
PS D:\SEM-7\ADF> django-admin startproject project name
CommandError: Destination directory 'D:\SEM-7\ADF\name' does not exist, please create it first.
PS D:\SEM-7\ADF> django-admin startproject 1project_name
CommandError: '1project_name' is not a valid project name. Please make sure the name is a valid identifier.
PS D:\SEM-7\ADF> django-admin startproject project_name1
PS D:\SEM-7\ADF> cd project_name1
PS D:\SEM-7\ADF\project_name1> cd ..
PS D:\SEM-7\ADF> django-admin startproject project_name1
CommandError: 'D:\SEM-7\ADF\project_name1' already exists
PS D:\SEM-7\ADF> django-admin startproject project_name1
PS D:\SEM-7\ADF> py manage.py startapp @app_name
C:\Python38\python.exe: can't open file 'manage.py': [Errno 2] No such file or directory
PS D:\SEM-7\ADF> cd project_name1
PS D:\SEM-7\ADF\project_name1> py manage.py startapp @app_name
usage: manage.py startapp [-h] [--template TEMPLATE] [--extension EXTENSIONS] [--name FILES] [--version] [-v {0,1,2,3}] [--settings SETTINGS]
                          [--pythonpath PYTHONPATH] [--traceback] [--no-color] [--force-color]
                          name [directory]
manage.py startapp: error: You must provide an application name.
PS D:\SEM-7\ADF\project_name1> py manage.py startapp &app_name
At line:1 char:23
+ py manage.py startapp &app_name
+                       ~
The ampersand (&) character is not allowed. The & operator is reserved for future use; wrap an ampersand in double quotation marks ("&") to pass it as
part of a string.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : AmpersandNotAllowed

PS D:\SEM-7\ADF\project_name1> py manage.py startapp app_name1
PS D:\SEM-7\ADF\project_name1>
```



__init__.py :

Empty file which tells python that treate this folder as python package.

admin.py :

In a Django project, the admin.py file is used to customize and configure the Django admin interface. The Django admin is a built-in web-based interface that allows developers and authorized users to manage and interact with the data stored in the database. It provides a user-friendly way to perform CRUD operations on the models defined in the project.

Its primary purpose is to register the models of that particular app with the admin interface. By doing so, the registered models become accessible through the admin panel, and you can easily create, read, update, and delete records from the database using the admin interface.

```
from django.contrib import admin
from .models import Comment
admin.site.register(Comment)
```

apps.py:

The primary use of the apps.py file is to define the configuration class for the app. This configuration class inherits from the AppConfig class provided by Django and allows you to specify various settings and behavior for the app.

verbose_name attribute to provide a custom display name for the app in the Django admin interface.

Also used to defining a ready() method to perform app-specific initialization when the app is loaded.

AppConfig Class:

name: This attribute specifies the name of the app as a string. By default, Django will set this attribute to the Python path of the app, such as 'myapp' for an app named myapp. You can override it to provide a custom name for the app.

label: This attribute is similar to the name attribute, but it is often used as a more human-readable version of the app's name. For example, if name is set to 'myapp', label could be set to 'My App'.

verbose_name: This attribute allows you to specify a more descriptive name for the app, which can be used in various places within the project, such as in the admin interface. If verbose_name is not set, Django will use the label attribute or the capitalized version of the name attribute as the verbose name.

default_auto_field: This attribute allows you to specify the default primary key field for models in the app. It is set to a dictionary with the key 'default' and the value of the field type. For example, 'default_auto_field': 'django.db.models.BigAutoField' sets the default primary key to a BigAutoField. This attribute was introduced in Django 3.2.

ready(): This method can be overridden in the AppConfig class to perform app-specific initialization tasks when the app is loaded. It is called when Django initializes the app and can be used, for example, to connect signal handlers or perform other setup tasks.

models.py:

The primary purpose of the models.py file is to define data models using Django's Object-Relational Mapping (ORM) system. Each data model is represented as a Python class that subclasses django.db.models.Model. Fields in the class represent database columns, and the model's attributes represent rows in the database table.

Ex:

```
title = models.CharField(max_length=200)
content = models.TextField()
pub_date = models.DateTimeField(auto_now_add=True)
```

Earlier AutoField – 32 bits integer (2,147,483,647)

Django 3.2 – Bi-64 Bits   (9,223,372,036,854,775,807)

This means that whenever you create a new model without specifying an explicit primary key field, Django will use BigAutoField instead of the default AutoField.

tests.py:

The tests.py file is an essential part of Django's file structure, and it is used for writing and organizing test cases for your Django application. Tests play a crucial role in ensuring that your application functions correctly and that any changes or updates to the codebase do not introduce bugs or regressions.

To run the test

```
python manage.py test
```

urls.py:

used to define URL patterns and map them to views in your Django application. It plays a significant role in handling incoming requests and directing them to the appropriate view functions to generate responses.

In the urls.py file, you define URL patterns using Django's URL patterns syntax. URL patterns are regular expressions or strings that match the incoming URLs from the user's browser. When a user accesses a specific URL on your website, Django uses the urls.py file to find the corresponding view to handle that request.

views.py:

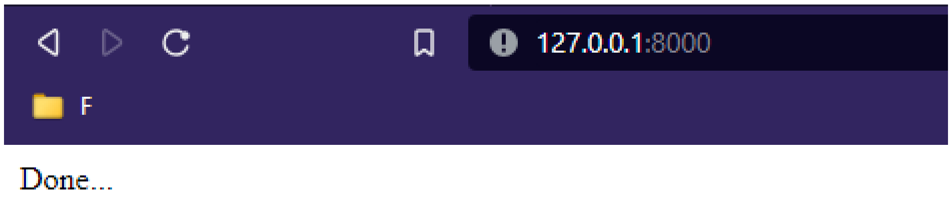
In the views.py file, you define the view functions that handle different URLs and HTTP methods. Each view function typically takes a request object as its first parameter and returns an HTTP response object.

```
article = get_article_from_database(article_id)
return render(request, 'article_detail.html', {'article': article})
```

To run the project:

Python manage.py runserver

Output:



Done...