Page No.
Date

Django App Description

Whenever we initialize a Django web application, it by default creates a number of pre-defined .py files along with folders. It also creates a simple web application. Along with these files & folders, the sample web app consists of an admin route which can only be accessed by the super admin/ creator of the website.

django-project, db.sqlite3 (SQLite database), manage.py (file used to run a django app) get created at first.
django-project folder consists of 4 .py files namely, __init__.py, settings.py, urls.py, wsgi.py
db.sqlite3 file consists the database for our project.
To run the project we call the manage.py file from terminal which redirects to localhost.

As we proceed with the project following files & folders are created. Following folders were created apart from the files & folders created already:

i) blog folder: Handles basics tasks and functions of website
ii) media folder: consists of profile pics that users uploaded
iii) users folder: Consists of all the files related to user functionality

Page No.
Date

Detailed Description → Blog Folder

Blog folder consists of many pre defined files. Only a few were updated & changed according to the website requirements.

Following are the files and folders created by Django automatically but no changes have been made:

i) __init__.py
ii) admin.py
iii) apps.py
iv) migrations folder
v) tests.py

Following are the files & folders to which changes have been made & have been created ~~which~~ during the web app creation:

i) models.py
ii) static folder
iii) templates folder
iv) urls.py
v) views.py

Lets see what these files & folder do in detail:

File 1: models.py

In this file we define what we need to save to the database
models was already imported by django & we didn't make any changes to it. Here

we create our Post model / Post class and we inherit from the inbuilt models.Model class (inbuilt)

Line 8 → It defines the post title. Its type is a character field with max length = 100 i.e it can accepts max 100 characters

Line 9 → It defines the post content. Its type is a text field. It allows user to enter information in multiple lines with no limit on number of characters.

Line 10 → It defines when the ~~the~~ post was created i.e date posted. Its a DateTimeField i.e it accepts the date i.e declares the date. We set its attribute as, default = timezone.now ~~Its default this attribute can be override~~ This attribute is imported (line 2)

Line 11 → We create the author of post. We also have a User table in our database and we need to import this to use it. We import User model in line 3. It has one to many relationship with Post model. author acts as a foreign key and we define it as shown, It takes 2 arguments, User (which is a table in database), on_delete = models.CASCADE. This on_delete argument tells Django to delete posts of user if user is deleted.

Line 13, 14 → We create a dunder __str__ method

This method allows post to be printed by its title.

Line 16,17 → Once the post is created we need to specify where to redirect the user to. We define get_absolute_url method for the same. We import reverse function (line 4). The reverse function returns full path as a string. The path we want to get is post-detail route with a specific post number which is the second argument as shown.

Folder 1 :- static folder
This consists of a CSS file used to style our web page.

Folder 2: templates folder
This folders consists all the HTML files used in blog folder

File 2: urls.py
This file mainly contains routes related to various posts.
Line 12 → A variable urlpatterns list is created.
Line 13 → Route to home page
Line 14 → Route to all the posts created by a specific user
Line 15 → Route to a particular post referred to by the post id

Page No.
Date

Line 16 → Route to create a new post
Line 17 → Route to update a post referred by the post id
Line 18 → Route to delete a post referred by post id
Line 19 → Route to the about page of website

In line 1 we import path module to define a new route
From line 3 to line 8 (both inclusive) we import several view classes from views.py file

File 3: views.py
In this file we use various in built django views add certain functionality to them and use them in our application.
Line 14 to 18 (Both inclusive): We create a home function which takes a request argument. We need to display posts of home page of our website. To do this we'll create a context dictionary which would fetch all posts from database. Then we return the request object, context dictionary and the corresponding html page (home.html). To return home.html we use render module imported in line 1.

Line 21 to 26 (Both inclusive): We create a class which defines how to list the posts on the home page. We inherit from inbuilt list view imported on line 5. We use Post model (line 23), template / corresponding HTML file is home.html (line 24)

(line 23), In our home function we call all our posts as 'posts' in our context dictionary but by default list view calls the variable object list instead of post. To change the variable name we set the attribute as, context_object_name = 'posts' (line 24). We order lists in order of date created with newest posts at top (line 25). We also paginate posts with 5 posts per page (line 26).

Line 29 to line 37 (Both inclusive): We can also view the posts created by a particular user. For that we create a class UserPostList View which inherits from inbuilt list view. Line 30, 31, 32, 33 are same as above class and explained above.

List View has an inbuilt method get_queryset() which we have over ridden in line 35. We need to get username from URL. If the user exists we should get all the posts by that user in descending order of date created i.e newest posts first. If user isn't found we return 404 error. We used get_object_or_404 function for this imported in line 1.

Line 40 to Line 41 (Both inclusive)

This class inherits from Detail View class imported on line 6. In this we change the database model to post model

Page No.
Date

Line 44 to 50 (Both inclusive): This class inherits from CreateView class imported on line 7. It takes the Post database model and is used to create a new post so we set the fields as title (Post Title) and content (Post Content).
We cannot directly use decorators with classes. Instead we use mixins. To create a new post a user should be logged in and hence we pass LoginRequiredMixin.
From line 48 to 50 we overwrite the pre defined form_valid method which takes self, form as arguments. On line 49, we set the author of the form. After this we validate the form

Line 53 to Line 65 (Both inclusive): This class inherits from update view class. It takes 2 additional arguments LoginRequiredMixin (discussed above) & UserPassesTestMixin. Only the post author should be able to update the post. To ensure this we have the UserPassesTestMixin. Once we have the mixin we create a test_func method. On line 62 we first get the post being updated. Then in line 63 we check if author of post is the one who is logged in we allow him to update post otherwise raise an error.

Line 68 to Line 76 (Both inclusive): This class allows user to delete a post and inherits from inbuilt delete view class. Line 70, if post is deleted successfully user is redirected

to home page. Rest of the code is same as above

Line 79, 80: The about function takes request argument & redirects to about page which has about.html as HTML page

Detailed Description → Django_Project Folder

Django_project folder consists of many predefined .py files. Only a few files were updated and to some files new code was added.

Following are the files which are created in this folder:

i) __init__.py
ii) settings.py
iii) urls.py
iv) wsgi.py

Following are the files to which changes have been made:

i) settings.py
ii) urls.py

File1: settings.py

Most of the code in this file was pre-defined. Only certain lines of code have been added.

Following code has been added:

* MEDIA_ROOT is going to be the full path to a directory where we want django to store uploaded files. The files are stored in system and not in database for performance reasons. MEDIA_URL is public URL of that directory

Line 34: This line is used for app configuration. Line 34 is inside a list called INSTALLED_APPS. Anytime we create a new application we need to add it to the list. This is required so that Django correctly searches templates

Line 35: This line is also in the INSTALLED_APPS list. This line is used for users app config.

Line 36: This line is also in INSTALLED_APPS list. crispy-forms allows us to put some simple tags in our template that will style our forms in a bootstrap fashion. Also we can use other CSS frameworks with crispy-forms

We also added lines 125 to line 138 in settings.py file.

* Line 125, 126: These are used to handle media files in our website and are used when we change/add new profile picture.

Line 128: This line defines what CSS framework is to be used by crispy-forms. We are using bootstrap 4 in this case

Line 130, 131: Line 130 tells that when we log in to our account we should be redirected to the home page. In line 131, say a user tries to access the user profile page w/o being logged in. This returns an about error. So to prevent this we write line 131 which redirects to login page

if we try to access anything of website that requires us to be logged in.

Line 133 to 138 (Both inclusive): These lines are used to connect to an email server. We send a link for resetting password incase a user forgets the password.

File 2: urls.py

This file mainly consists of various routes used in our website. The url routes are mainly related to a user.

It consists of the following routes:

i) admin/ route: Used to direct to adminpage which can only be accessed by a creator/super admin of web app

ii) register/: Redirects a user to the register route where he can register himself/create an account

iii) profile/: Redirects to user profile route

iv) login/: Redirects to the route where user can login into his account

v) logout/: Used to logout a user

Line 29 to 48: The routes mentioned here are used to reset a password in case a user forgets his password

→ password-reset/ route redirects to a page where user can enter email to receive the reset password token

→ password-reset/done/ route gives a conformation

that the password has been reset and a mail with the instructions has been sent to the registered mail id

→ password-reset-confirm/<uidb64>/<token>/ route redirects to a password reset page where we can enter our password & confirm it in order to change the password & update it

→ password-reset-complete/ route gives the information that password has been reset and allows user to sign into his account with a newly created password

Detailed Description — users folder

The users folder consists of many predefined files. Only a few files were changed according to requirements of the website.

Following are the files & folders created by django automatically and no changes have been made to them :

i) __init__.py
ii) admin.py
iii) apps.py
iv) migrations folder
v) tests.py

Following are the files & folders to which either changes have been made or are created by the by us during web app creation :

i) forms.py

ii) models.py
iii) signals.py
iv) templates folder
v) views.py

File 1: forms.py

This file consists of the forms that are to be created/filled by the user. This consists of 3 forms:

i) User Registeration Form
ii) User profile update form
iii) User profile picture update form

In all these forms we have created a meta class. It can be thought of as a function decorator. This can be thought of as over writing an existing class.

In each Meta class we define, the database model to be used and the fields for the form we need to use.

File 2: models.py

In the models.py file we create a Profile database which is used to store the profile images of users and inherits from the Model database.

In line 7, we store the users details in a variable called user. This is a one to one field as all the users are unique. on_delete=models.CASCADE means that if a user is deleted from database all his details should

The one to one model used in line 7 means that one user has one profile & one profile will be associated with one user

Page No.
Date

be removed including the profile
In line 8 we have the image field. Default profile picture when a user creates an account would be default.jpg and the profile pictures would be uploaded to profile_pics folder.

Line 10 is a dunder method -- etc --. It defines how ~~it~~ will the profile be displayed when we print it
Line 13 to line 21 defines a function to save the uploaded image. The images save automatically when we upload it. However, we'll overwrite the save method to make some changes. We first run save method of parent class which is line 14. The line 14 save the image if size is less than 300 × 300 pixels. If the either the width or height or both are larger than 300 pixels then we resize it to 300 × 300 pixels (line 19, 20) and save it.

File 3 : signals.py

In this file we first import a signal called post_save (line 1). The post_save signal comes into picture when object gets saved. We want to get a post_save signal when user is created. We also import the User models (line 2) which acts as a sender since it sends the signal. We create a receiver which is a function that receives the signal and performs a task. We import receiver in line 3. We also import Profile from models.py since we are creating a

Page No.
Date

Profile in our function. (line 4)

We'll now create a user profile (line 7 to 10) which takes sender, instance, created & **kwargs as arguments. We run this function everytime a user gets created. The function says if user is created, he should also have a profile i.e a profile should also be created. We need to tie functionality together. We add a decorator receiver (line 7). The decorator takes 2 arguments, signal that we want (post_save) and the sender. In other words, we have a sender and a signal of post_save. When a user is saved, we have to send the signal which is going to be received by receiver. The receiver is create profile function which takes all arguments passed by post_save signal. instance argument defines instance of a user. The function says, if the user was created, then create a profile object with user = instance of the user created.

We'll also create a save_profile function which saves the user profile when the user is created which would be almost similar to above function (line 13 to 15). instance.profile.save() command is used for the same.

Folder 1: templates folder

This folder consists of all the HTML files related to users folder

File 4 : views.py

Line 7 to Line 17 (Both inclusive) : In these lines we create a function ~~route~~ which allows to register a new user. We import the UserRegistrationForm ~~from~~ from forms.py file. If the form is valid, we save the form, store the username in variable, pass a success message and redirect to login route. If the form isn't saved or has error we re-return the form

Line 20 to Line 42 (Both inclusive) : In these lines users can access this profile and also update username, email and profile picture. In this we create 2 forms, using 1 form you can update username, email & using another we can update profile picture. We combine them so that they look as one form. To access a profile, user must be logged in and hence the decorator at line 20. UserUpdateForm (line 23), ProfileUpdateForm (line 24) have been imported from forms.py file. We also create a context dictionary (line 37 to 40) & return it. When we update the form we want that the forms should be filled with the existing username, email & profile picture and hence we pass the required arguments in U-form (line 23), P-form (line 24) If the forms submitted are valid we return a message that account has been created and

redirect to profile page otherwise return a error.

In line 42 we return, @ profile.html is HTML page for the same