Authentication Views

Django provides several views that you can use for handling login, logout, and password management.

These make use of the stock auth forms but you can pass in your own forms as well.

Django provides no default template for the authentication views.

You should create your own templates for the views you want to use.

urls.py

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

These all url will be available:-

accounts/login/ [name='login']

accounts/logout/ [name='logout']

accounts/password_change/ [name='password_change']

accounts/password_change/done/ [name='password_change_done']

accounts/password_reset/ [name='password_reset']

accounts/password_reset/done/ [name='password_reset_done']

accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']

accounts/reset/done/ [name='password_reset_complete']

```python
urlpatterns = [
    path('login/', views.LoginView.as_view(), name='login'),
    path('logout/', views.LogoutView.as_view(), name='logout'),

    path('password_change/', views.PasswordChangeView.as_view(), name='password_change'),
    path('password_change/done/', views.PasswordChangeDoneView.as_view(), name=
    'password_change_done'),

    path('password_reset/', views.PasswordResetView.as_view(), name='password_reset'),
    path('password_reset/done/', views.PasswordResetDoneView.as_view(), name=
    'password_reset_done'),
    path('reset/<uidb64>/<token>/', views.PasswordResetConfirmView.as_view(), name=
    'password_reset_confirm'),
    path('reset/done/', views.PasswordResetCompleteView.as_view(), name='password_reset_complete'
    ),
]
```

Authentication Views

```python
urlpatterns = [
    path('login/', views.LoginView.as_view(), name='login'),
    path('logout/', views.LogoutView.as_view(), name='logout'),

    path('password_change/', views.PasswordChangeView.as_view(), name='password_change'),
    path('password_change/done/', views.PasswordChangeDoneView.as_view(), name=
    'password_change_done'),

    path('password_reset/', views.PasswordResetView.as_view(), name='password_reset'),
    path('password_reset/done/', views.PasswordResetDoneView.as_view(), name=
    'password_reset_done'),
    path('reset/<uidb64>/<token>/', views.PasswordResetConfirmView.as_view(), name=
    'password_reset_confirm'),
    path('reset/done/', views.PasswordResetCompleteView.as_view(), name='password_reset_complete'
    ),
]
```

login_required Decorator

login_required(redirect_field_name='next', login_url=None)

If the user is logged in, execute the view normally. The view code is free to assume the user is logged in.

If the user isn't logged in, redirect to settings.LOGIN_URL, passing the current absolute path in the query string. Example: /accounts/login/?next=/accounts/profile/

django.contrib.auth.decorators.login_required

Where,

redirect_field_name - If you would prefer to use a different name for this parameter, login_required() takes an optional redirect_field_name parameter. If you provide a value to redirect_field_name, you will most likely need to customize your login template as well, since the template context variable which stores the redirect path will use the value of redirect_field_name as its key rather than "next" (the default).

login_url - If you don't specify the login_url parameter, you'll need to ensure that the settings.LOGIN_URL and your login view are properly associated.

The settings.LOGIN_URL also accepts view function names and named URL patterns. This allows you to freely remap your login view within your URLconf without having to update the setting.

login_required Decorator

Example:-

from django.contrib.auth.decorators import login_required

@login_required

def profile(request):

 return render(request, 'registration/profile.html')

staff_member_required decorator

staff_member_required(redirect_field_name='next', login_url='admin:login')

This decorator is used on the admin views that require authorization. A view decorated with this function will having the following behavior:

- If the user is logged in, is a staff member (User.is_staff=True), and is active (User.is_active=True), execute the view normally.

- Otherwise, the request will be redirected to the URL specified by the login_url parameter, with the originally requested path in a query string variable specified by redirect_field_name.
  For example: /admin/login/?next=/profile/

Example:-

from django.contrib.admin.views.decorators import staff_member_required

@staff_member_required

def profile(request):

 return render(request, 'registration/profile.html')

permission_required Decorator

permission_required(perm, login_url=None, raise_exception=False)

It's a relatively common task to check whether a user has a particular permission. For that reason, Django provides a shortcut for that case: the permission_required() decorator.

Just like the has_perm() method, permission names take the form "<app label>.<permission codename>"

Example:-

from django.contrib.auth.decorators import permission_required

@permission_required('blog.can_add')

def profile(request):

 return render(request, 'registration/profile.html')

Decorating Class-Based View

Decorating in urls.py or URLconf

The simplest way of decorating class-based views is to decorate the result of the as_view() method. The easiest place to do this is in the URLconf where you deploy your view:

from django.urls import path

from django.views.generic import TemplateView

from registration.views import ProfileTemplateView

from django.contrib.auth.decorators import login_required

urlpatterns = [

   path('dashboard/', login_required(TemplateView.as_view(template_name='blog/dash.html')), name='dash'),

   path('profile/', login_required(ProfileTemplateView.as_view(template_name='registration/profile.html')), name='profile'),

   path('blogpost/', permission_required('blog.can_add')(BlogPostView.as_view())),

]

method_decorator

The method_decorator decorator transforms a function decorator into a method decorator so that it can be used on an instance method.

A method on a class isn't quite the same as a standalone function, so you can't just apply a function decorator to the method you need to transform it into a method decorator first.

@method_decorator(*args, **kwargs)

Decorating Class-Based View

Decorating in the Class

To decorate every instance of a class-based view, you need to decorate the class definition itself. To do this you apply the decorator to the dispatch() method of the class.

from django.utils.decorators import method_decorator

from django.views.generic import TemplateView

class ProfileTemplateView(TemplateView):

    template_name = 'registration/profile.html'

    @method_decorator(login_required)

    def dispatch(self, *args, **kwargs):

        return super().dispatch(*args, **kwargs)

Decorating Class-Based View

Decorating in the Class

You can decorate the class instead and pass the name of the method to be decorated as the keyword argument name:

@method_decorator(login_required, name='dispatch')

class ProfileTemplateView(TemplateView):

 template_name = 'registration/profile.html'

Decorating Class-Based View

Decorating in the Class

If you have a set of common decorators used in several places, you can define a list or tuple of decorators and use this instead of invoking method_decorator() multiple times.

@method_decorator(never_cache, name='dispatch')

@method_decorator(login_required, name='dispatch')

class ProfileTemplateView(TemplateView):

 template_name = 'registration/profile.html'

decorators = [never_cache, login_required]

@method_decorator(decorators, name='dispatch')

class ProfileTemplateView(TemplateView):

 template_name = 'registration/profile.html'

The decorators will process a request in the order they are passed to the decorator. In the example, never_cache() will process the request before login_required().

Authentication Views

This is a list with all the views *django.contrib.auth* provides.

- LoginView

- LogoutView

- PasswordChangeView

- PasswordChangeDoneView

- PasswordResetView

- PasswordResetDoneView

- PasswordResetConfirmView

- PasswordResetCompleteView

LoginView

django.contrib.auth.views.LoginView

template_name = 'registration/login.html'

path('login/', views.LoginView.as_view(), name='login')

It's your responsibility to provide the html for the login template , called registration/login.html by default.

- If called via GET, it displays a login form that POSTs to the same URL.

- If called via POST with user submitted credentials, it tries to log the user in.

- If login is successful, the view redirects to the URL specified in next.

- If next isn't provided, it redirects to settings.LOGIN_REDIRECT_URL (which defaults to /accounts/profile/).

- If login isn't successful, it redisplays the login form.

LoginView

Default Template:-

registration/login.html

This template gets passed four template context variables:

- form: A Form object representing the AuthenticationForm.

- next: The URL to redirect to after successful login. This may contain a query string, too.

- site: The current Site, according to the SITE_ID setting. If you don't have the site framework installed, this will be set to an instance of RequestSite, which derives the site name and domain from the current HttpRequest.

- site_name: An alias for site.name. If you don't have the site framework installed, this will be set to the value of request.META['SERVER_NAME'].

Custom Template:-

path('login/', views.LoginView.as_view(template_name='myapp/login.html'), name='login')

LoginView

<u>Attributes:-</u>

template_name: The name of a template to display for the view used to log the user in. Defaults to registration/login.html.

redirect_field_name: The name of a GET field containing the URL to redirect to after login. Defaults to *next*.

authentication_form: A callable (typically a form class) to use for authentication. Defaults to AuthenticationForm.

extra_context: A dictionary of context data that will be added to the default context data passed to the template.

redirect_authenticated_user: A boolean that controls whether or not authenticated users accessing the login page will be redirected as if they had just successfully logged in. Defaults to False.

success_url_allowed_hosts: A set of hosts, in addition to request.get_host(), that are safe for redirecting after login. Defaults to an empty set.

LogoutView

django.contrib.auth.views.LogoutView

template_name ='registration/logged_out.html'

path('logout/', views.LogoutView.as_view(), name='logout')

logged_out.html template is already available and used by admin logout template file.

You can create your own custom logout template by defining template_name attribute.
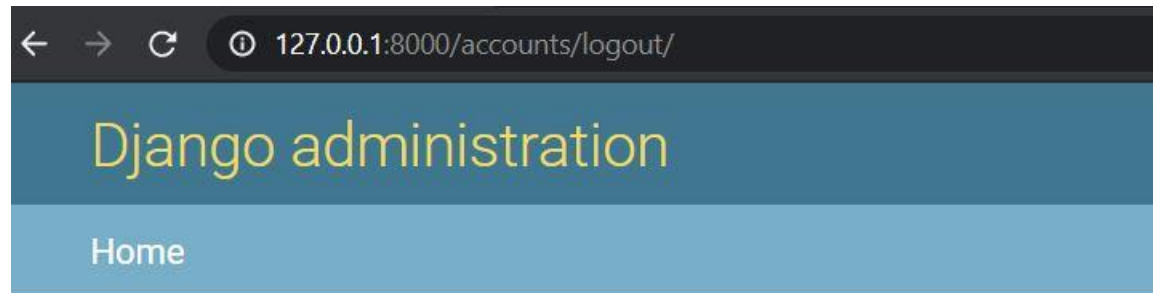
LogoutView

Default Template:-

registration/logged_out.html

This template gets passed three template context variables:

- title: The string "Logged out", localized.

- site: The current Site, according to the SITE_ID setting. If you don't have the site framework installed, this will be set to an instance of RequestSite, which derives the site name and domain from the current HttpRequest.

- site_name: An alias for site.name. If you don't have the site framework installed, this will be set to the value of request.META['SERVER_NAME'].

Custom Template:-

path('logout/', views.LogoutView.as_view(template_name='myapp/logout.html'), name='logout')

LogoutView

next_page: The URL to redirect to after logout. Defaults to settings.LOGOUT_REDIRECT_URL.

template_name: The full name of a template to display after logging the user out. Defaults to registration/logged_out.html.

redirect_field_name: The name of a GET field containing the URL to redirect to after log out. Defaults to next. Overrides the next_page URL if the given GET parameter is passed.

extra_context: A dictionary of context data that will be added to the default context data passed to the template.

success_url_allowed_hosts: A set of hosts, in addition to request.get_host(), that are safe for redirecting after logout. Defaults to an empty set.

PasswordChangeView
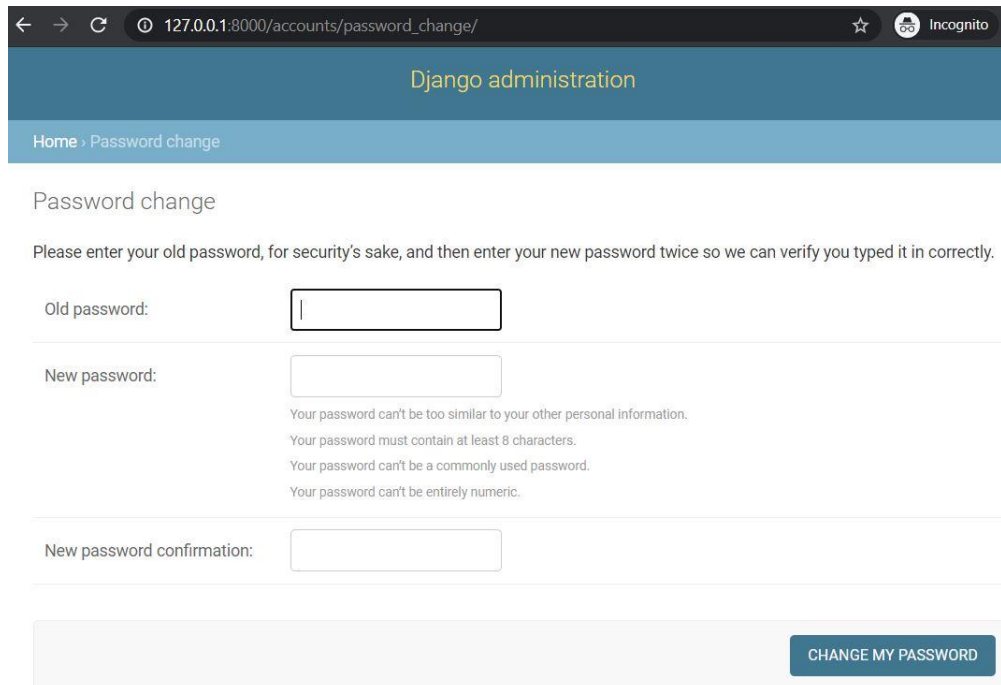
django.contrib.auth.views.PasswordChangeView

template_name ='registration/password_change_form.html'

path('password_change/', views.PasswordChangeView.as_view(), name='password_change'),

registration/password_change_form.html template is already available and used by admin change password template file.

You can create your own custom change password template by defining template_name attribute.

PasswordChangeView

registration/password_change_form.html

This template gets passed following template context variables:

- form: The password change form

path('password_change/',
views.PasswordChangeView.as_view(template_name='myapp/changepass.html'),
name='password_change'),

PasswordChangeView

Attributes:-

template_name: The full name of a template to use for displaying the password change form. Defaults to registration/password_change_form.html if not supplied.

success_url: The URL to redirect to after a successful password change. Defaults to 'password_change_done'.

form_class: A custom "change password" form which must accept a user keyword argument. The form is responsible for actually changing the user's password. Defaults to PasswordChangeForm.

extra_context: A dictionary of context data that will be added to the default context data passed to the template.
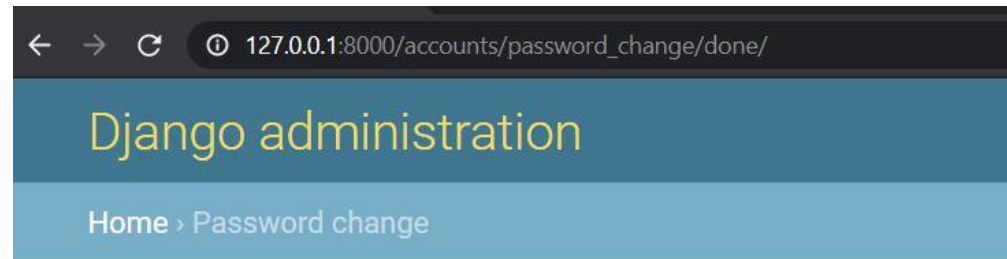
PasswordChangeDoneView

django.contrib.auth.views.PasswordChangeDoneView

template_name ='registration/password_change_done.html'

path('password_change/done/', views.PasswordChangeDoneView.as_view(), name='password_change_done'),

registration/password_change_done.html template is already available and used by admin password change done template file.

You can create your own custom password change done template by defining template_name attribute.

PasswordChangeDoneView

Default Template:-

registration/password_change_done.html

Custom Template:-

path('password_change/done/',
views.PasswordChangeDoneView.as_view(template_name='myapp/changepassdone.html'),
name='password_change_done'),

PasswordChangeDoneView

template_name: The full name of a template to use. Defaults to registration/password_change_done.html if not supplied.

extra_context: A dictionary of context data that will be added to the default context data passed to the template.
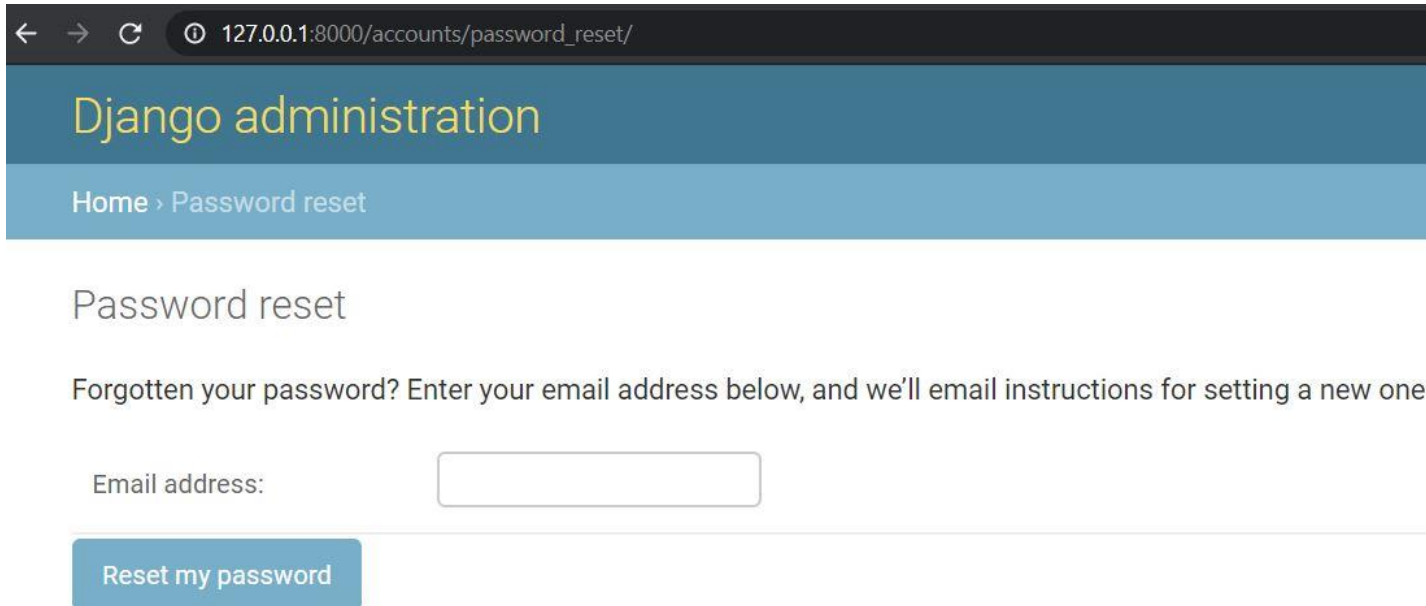
PasswordResetView

django.contrib.auth.views.PasswordResetView

template_name ='registration/password_reset_form.html'

path('password_reset/', views.PasswordResetView.as_view(), name='password_reset'),

registration/password_reset_form.html template is already available and used by admin password reset template file.

You can create your own custom password reset template by defining template_name attribute.

PasswordResetView

Default Template:-

registration/password_reset_form.html

This template gets passed following template context variables:

- form: The form for resetting the user's password.

Custom Template:-

path('password_reset/', views.PasswordResetView.as_view(template_name='myapp/resetpass.html'), name='password_reset'),

PasswordResetView

This Email template gets passed following email template context variables:

- email: An alias for user.email

- user: The current User, according to the email form field. Only active users are able to reset their passwords (User.is_active is True).

- site_name: An alias for site.name. If you don't have the site framework installed, this will be set to the value of request.META['SERVER_NAME']. For more on sites, see The "sites" framework.

- domain: An alias for site.domain. If you don't have the site framework installed, this will be set to the value of request.get_host().

- protocol: http or https

- uid: The user's primary key encoded in base 64.

- token: Token to check that the reset link is valid.

PasswordResetView

<u>Attributes:-</u>

template_name: The full name of a template to use for displaying the password reset form. Defaults to registration/password_reset_form.html if not supplied.

form_class: Form that will be used to get the email of the user to reset the password for. Defaults to PasswordResetForm.

email_template_name: The full name of a template to use for generating the email with the reset password link. Defaults to registration/password_reset_email.html if not supplied.

subject_template_name: The full name of a template to use for the subject of the email with the reset password link. Defaults to registration/password_reset_subject.txt if not supplied.

token_generator: Instance of the class to check the one time link. This will default to default_token_generator, it's an instance of django.contrib.auth.tokens.PasswordResetTokenGenerator.

success_url: The URL to redirect to after a successful password reset request. Defaults to 'password_reset_done'.

PasswordResetView

from_email: A valid email address. By default Django uses the DEFAULT_FROM_EMAIL.

extra_context: A dictionary of context data that will be added to the default context data passed to the template.

html_email_template_name: The full name of a template to use for generating a text/html multipart email with the password reset link. By default, HTML email is not sent.

extra_email_context: A dictionary of context data that will be available in the email template. It can be used to override default template context values listed below e.g. domain.

PasswordResetDoneView

django.contrib.auth.views.PasswordResetDoneView

The page shown after a user has been emailed a link to reset their password. This view is called by default if the PasswordResetView doesn't have an explicit success_url URL set.

If the email address provided does not exist in the system, the user is inactive, or has an unusable password, the user will still be redirected to this view but no email will be sent.

template_name ='registration/password_reset_done.html'

path('password_reset/done/', views.PasswordResetDoneView.as_view(), name='password_reset_done'),

registration/password_reset_done.html template is already available and used by admin password reset done template file.

You can create your own custom password reset done template by defining template_name attribute.
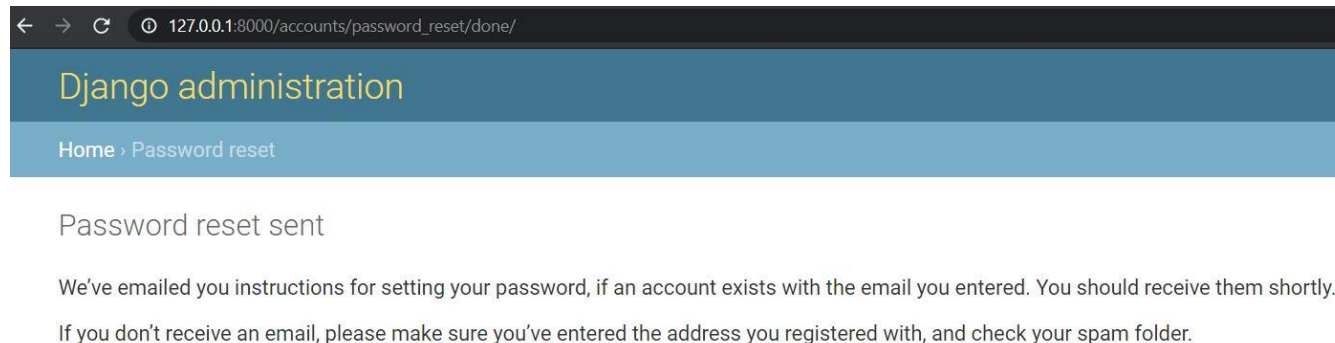
PasswordResetDoneView

Default Template:-

registration/password_reset_done.html

Custom Template:-

path('password_reset/done/',
views.PasswordResetDoneView.as_view(template_name='myapp/resetpassdone.html'),
name='password_reset_done'),

PasswordResetDoneView

<u>Attributes:-</u>

template_name: The full name of a template to use. Defaults to registration/password_reset_done.html if not supplied.

extra_context: A dictionary of context data that will be added to the default context data passed to the template.

PasswordResetConfirmView

django.contrib.auth.views.PasswordResetConfirmView

It presents a form for entering a new password.

Keyword arguments from the URL:

- uidb64: The user's id encoded in base 64.

- token: Token to check that the password is valid.

template_name ='registration/password_reset_confirm.html'

path('reset/<uidb64>/<token>/', views.PasswordResetConfirmView.as_view(), name='password_reset_confirm'),

registration/password_reset_confirm.html template is already available and used by admin password reset confirm template file.

You can create your own custom password reset confirm template by defining template_name attribute.

PasswordResetConfirmView

Default Template:-

registration/password_reset_confirm.html

This template gets passed following template context variables:

- form: The form for setting the new user's password.

- validlink: Boolean, True if the link (combination of uidb64 and token) is valid or unused yet.

Custom Template:-

path('reset/<uidb64>/<token>/',
views.PasswordResetConfirmView.as_view(template_name='myapp/resetpassconfirm.html'),
name='password_reset_confirm'),

PasswordResetConfirmView

Attributes:-

template_name: The full name of a template to display the confirm password view. Default value is registration/password_reset_confirm.html.

token_generator: Instance of the class to check the password. This will default to default_token_generator, it's an instance of django.contrib.auth.tokens.PasswordResetTokenGenerator.

post_reset_login: A boolean indicating if the user should be automatically authenticated after a successful password reset. Defaults to False.

post_reset_login_backend: A dotted path to the authentication backend to use when authenticating a user if post_reset_login is True. Required only if you have multiple AUTHENTICATION_BACKENDS configured. Defaults to None.

form_class: Form that will be used to set the password. Defaults to SetPasswordForm.

success_url: URL to redirect after the password reset done. Defaults to 'password_reset_complete'.

extra_context: A dictionary of context data that will be added to the default context data passed to the template.

PasswordResetConfirmView

Attributes:-

reset_url_token: Token parameter displayed as a component of password reset URLs. Defaults to 'set-password'.

PasswordResetCompleteView

django.contrib.auth.views.PasswordResetCompleteView

It presents a view which informs the user that the password has been successfully changed.

template_name ='registration/password_reset_complete.html'

path('reset/done/', views.PasswordResetCompleteView.as_view(), name='password_reset_complete'),

registration/password_reset_complete.html template is already available and used as admin password reset complete template file.

You can create your own custom password reset complete template by defining template_name attribute.

PasswordResetCompleteView

Default Template:-

registration/password_reset_complete.html

Custom Template:-

path('reset/done/',
views.PasswordResetCompleteView.as_view(template_name='myapp/resetpasscomp.html'),
name='password_reset_complete'),

PasswordResetCompleteView

template_name: The full name of a template to display the view. Defaults to registration/password_reset_complete.html.

extra_context: A dictionary of context data that will be added to the default context data passed to the template.

Built-in Forms

- AdminPasswordChangeForm - A form used in the admin interface to change a user's password. It takes the user as the first positional argument.

- AuthenticationForm - A form for logging a user in. It takes request as its first positional argument, which is stored on the form instance for use by sub-classes.

- PasswordChangeForm - A form for allowing a user to change their password.

- PasswordResetForm - A form for generating and emailing a one-time use link to reset a user's password.

- SetPasswordForm - A form that lets a user change their password without entering the old password.

- UserChangeForm - A form used in the admin interface to change a user's information and permissions.

- UserCreationForm - A ModelForm for creating a new user.