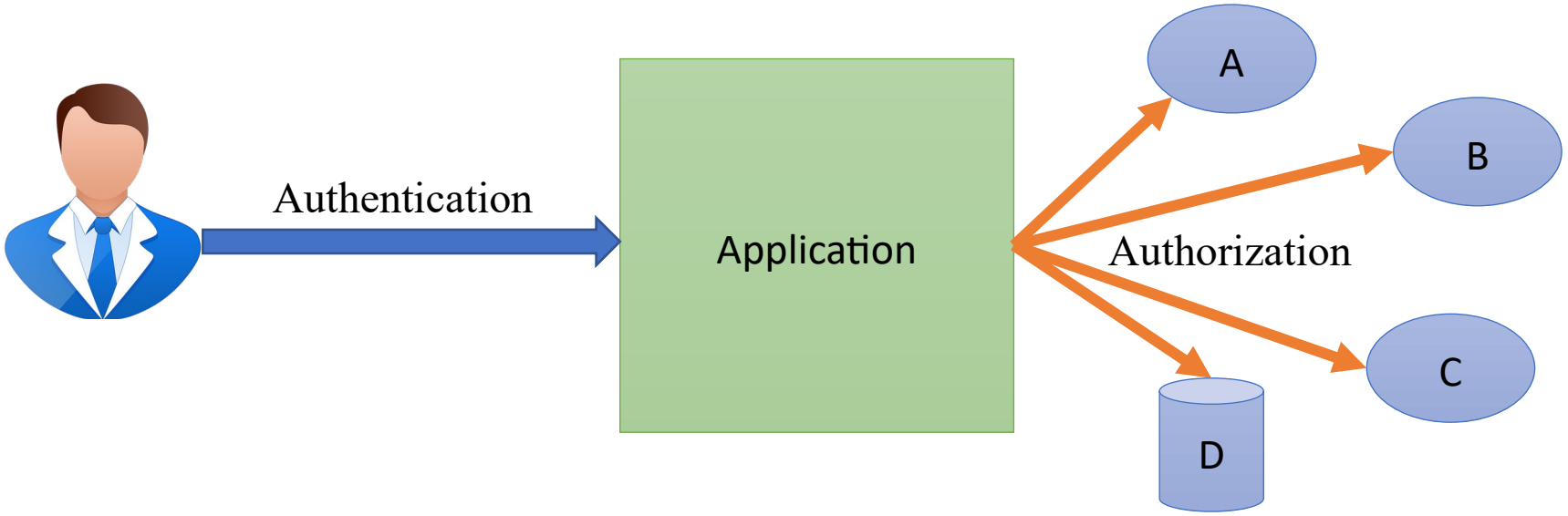
Authentication and Authorization

Authentication – Authentication is about validating your credentials like Username and password to verify your identity.

Authorization – Authorization is the process to determine whether the authenticated user has access to the particular resources. It checks your rights to grant you access to resources such as information, databases, files, etc.



User Authentication System

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions.

Django authentication provides both authentication and authorization together and is generally referred to as the authentication system.

By default, the required configuration is already included in the settings.py generated by django-admin startproject, these consist of two items listed in your INSTALLED_APPS setting:

'django.contrib.auth' contains the core of the authentication framework, and its default models.

'django.contrib.contenttypes' is the Django content type system, which allows permissions to be associated with models you create.

and these items in your MIDDLEWARE setting:

SessionMiddleware manages sessions across requests.

AuthenticationMiddleware associates users with requests using sessions.

django.contrib.auth

C:\Users\RK\AppData\Local\Programs\Python\Python38-32\Lib\site-packages\django\contrib\auth

models.py

class User

User Object - User objects are the core of the authentication system.

They typically represent the people interacting with your site and are used to enable things like restricting access, registering user profiles, associating content with creators etc.

Only one class of user exists in Django’s authentication framework, i.e., 'superusers' or admin 'staff' users are just user objects with special attributes set, not different classes of user objects.

- Creating Super Users
- Changing Password
- Authenticating User
- Creating Users
- Permissions and Authorization
- Groups
- How to log a user in
- How to log a user out

User Authentication System

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions.

Django authentication provides both authentication and authorization together and is generally referred to as the authentication system.

By default, the required configuration is already included in the settings.py generated by django-admin startproject, these consist of two items listed in your INSTALLED_APPS setting:

'django.contrib.auth' contains the core of the authentication framework, and its default models.

'django.contrib.contenttypes' is the Django content type system, which allows permissions to be associated with models you create.

and these items in your MIDDLEWARE setting:

SessionMiddleware manages sessions across requests.

AuthenticationMiddleware associates users with requests using sessions.

django.contrib.auth

C:\Users\RK\AppData\Local\Programs\Python\Python38-32\Lib\site-packages\django\contrib\auth

models.py

class User

User Object - User objects are the core of the authentication system.

They typically represent the people interacting with your site and are used to enable things like restricting access, registering user profiles, associating content with creators etc.

Only one class of user exists in Django’s authentication framework, i.e., 'superusers' or admin 'staff' users are just user objects with special attributes set, not different classes of user objects.

User object Fields

username - Usernames may contain alphanumeric, _, @, +, . and - characters. Its required and length is 150 characters or fewer.

first_name – Its optional (blank=True) and length is 30 characters or fewer

last_name – Its optional (blank=True) and length is 150 characters or fewer.

email – Its optional (blank=True)

password – A hash of, and metadata about, the password. Django doesn’t store the raw password. Its required.

groups - Many-to-many relationship to Group.

User object Fields

user_permissions - Many-to-many relationship to Permission.

is_staff - Designates whether this user can access the admin site. It takes Boolean.

is_active - Designates whether this user account should be considered active. We recommend that you set this flag to False instead of deleting accounts; that way, if your applications have any foreign keys to users, the foreign keys won't break. It takes Boolean.

is_superuser - Designates that this user has all permissions without explicitly assigning them. It takes Boolean.

last_login - A datetime of the user's last login.

date_joined - A datetime designating when the account was created. Is set to the current date/time by default when the account is created.

User object Fields

is_authenticated - This is a way to tell if the user has been authenticated. This does not imply any permissions and doesn’t check if the user is active or has a valid session. Its a read-only attribute which is always True.

is_anonymous - This is a way of differentiating User and AnonymousUser objects. It’s a read-only attribute which is always False

User object Methods

get_username() - Since the User model can be swapped out, you should use this method instead of referencing the username attribute directly. It returns the username for the user.

get_full_name() – It returns the first_name plus the last_name, with a space in between.

get_short_name() – It returns the first_name.

set_password(raw_password) – It sets the user's password to the given raw string, taking care of the password hashing. Doesn't save the User object.

When the raw_password is None, the password will be set to an unusable password, as if set_unusable_password() were used.

check_password(raw_password) – It returns True if the given raw string is the correct password for the user. (This takes care of the password hashing in making the comparison.)

User object Methods

set_unusable_password() – It marks the user as having no password set. This isn't the same as having a blank string for a password. check_password() for this user will never return True. Doesn't save the User object.

You may need this if authentication for your application takes place against an existing external source such as an LDAP directory.

has_usable_password() – It returns False if set_unusable_password() has been called for this user.

get_user_permissions(obj=None) – It returns a set of permission strings that the user has directly.

If obj is passed in, only returns the user permissions for this specific object.

get_group_permissions(obj=None) – It returns a set of permission strings that the user has, through their groups.

If obj is passed in, only returns the group permissions for this specific object.

User object Methods

get_all_permissions(obj=None) – It returns a set of permission strings that the user has, both through group and user permissions.

If obj is passed in, only returns the permissions for this specific object.

has_perm(perm, obj=None) – It returns True if the user has the specified permission, where perm is in the format "<app label>.<permission codename>". If the user is inactive, this method will always return False. For an active superuser, this method will always return True.

If obj is passed in, this method won't check for a permission for the model, but for this specific object.

has_perms(perm_list, obj=None) – It returns True if the user has each of the specified permissions, where each perm is in the format "<app label>.<permission codename>". If the user is inactive, this method will always return False. For an active superuser, this method will always return True.

If obj is passed in, this method won't check for permissions for the model, but for the specific object.

User object Methods

has_module_perms(package_name) – It returns True if the user has any permissions in the given package (the Django app label). If the user is inactive, this method will always return False. For an active superuser, this method will always return True.

email_user(subject, message, from_email=None, **kwargs) – It sends an email to the user. If from_email is None, Django uses the DEFAULT_FROM_EMAIL. Any **kwargs are passed to the underlying send_mail() call.

UserManager Methods

The User model has a custom manager that has the following helper methods (in addition to the methods provided by BaseUserManager):

create_user(username, email=None, password=None, **extra_fields) – It creates, saves and returns an User.

The username and password are set as given. The domain portion of email is automatically converted to lowercase, and the returned User object will have is_active set to True.

If no password is provided, set_unusable_password() will be called.

The extra_fields keyword arguments are passed through to the User's __init__ method to allow setting arbitrary fields on a custom user model.

create_superuser(username, email=None, password=None, **extra_fields) – It is same as create_user(), but sets is_staff and is_superuser to True.

UserManager Methods

with_perm(perm, is_active=True, include_superusers=True, backend=None, obj=None) – It returns users that have the given permission perm either in the "<app label>.<permission codename>" format or as a Permission instance. Returns an empty queryset if no users who have the perm found.

If is_active is True (default), returns only active users, or if False, returns only inactive users. Use None to return all users irrespective of active state.

If include_superusers is True (default), the result will include superusers.

If backend is passed in and it's defined in AUTHENTICATION_BACKENDS, then this method will use it. Otherwise, it will use the backend in AUTHENTICATION_BACKENDS, if there is only one, or raise an exception.

Group Object Fields

name - Any characters are permitted. It is required and length is 150 characters or fewer.
Example: 'Awesome Users'.

permissions - Many-to-many field to Permission.

Permission Object Fields

name – It is required and length is 255 characters or fewer. Example: 'Can vote'.

content_type - A reference to the django_content_type database table, which contains a record for each installed model. It is required.

codename – It is required and length is 100 characters or fewer. Example: 'can_vote'.

authenticate ( )

authenticate(request=None, **credentials) – This is used to verify a set of credentials. It takes credentials as keyword arguments, username and password for the default case, checks them against each authentication backend, and returns a User object if the credentials are valid for a backend.

If the credentials aren’t valid for any backend or if a backend raises PermissionDenied, it returns None.

request is an optional HttpRequest which is passed on the authenticate() method of the authentication backends.

Example:-

user = authenticate(username=‘sonam’, password=‘geekyshows’)

login ( )

login(request, user, backend=None) - To log a user in, from a view, use login(). It takes an HttpRequest object and a User object. login() saves the user's ID in the session, using Django's session framework.

When a user logs in, the user's ID and the backend that was used for authentication are saved in the user's session. This allows the same authentication backend to fetch the user's details on a future request.

logout ( )

logout(request) - To log out a user who has been logged in via django.contrib.auth.login(), use django.contrib.auth.logout() within your view. It takes an HttpRequest object and has no return value.

When you call logout(), the session data for the current request is completely cleaned out. All existing data is removed. This is to prevent another person from using the same Web browser to log in and have access to the previous user’s session data.

update_session_auth_hash ( )

update_session_auth_hash(request, user) - This function takes the current request and the updated user object from which the new session hash will be derived and updates the session hash appropriately. It also rotates the session key so that a stolen session cookie will be invalidated.

Permissions and Authorization

Django comes with a built-in permissions system. It provides a way to assign permissions to specific users and groups of users.

The Django admin site uses permissions as follows:

- Access to view objects is limited to users with the “view” or “change” permission for that type of object.
- Access to view the “add” form and add an object is limited to users with the “add” permission for that type of object.
- Access to view the change list, view the “change” form and change an object is limited to users with the “change” permission for that type of object.
- Access to delete an object is limited to users with the “delete” permission for that type of object.

Permissions and Authorization

myuser.groups.set([group_list])

myuser.groups.add(group, group, ...)

myuser.groups.remove(group, group, ...)

myuser.groups.clear()

myuser.user_permissions.set([permission_list])

myuser.user_permissions.add(permission, permission, ...)

myuser.user_permissions.remove(permission, permission, ...)

myuser.user_permissions.clear()

Permissions and Authorization

When a model is created, Django will automatically create four default permissions for the following actions:

- add: Users with this permission can add an instance of the model.
- delete: Users with this permission can delete an instance of the model.
- change: Users with this permission can update an instance of the model.
- view: Users with this permission can view instances of this model.

Permission names follow a very specific naming convention: appname.action_modelname

Example:- enroll.delete_blog

perms Template Variable

The currently logged-in user’s permissions are stored in the template variable {{ perms }}. This is an instance of django.contrib.auth.context_processors.PermWrapper, which is a template-friendly proxy of permissions.

Example:-

```
{% if perms.enroll.delete_blog %}
<input type="button" value="Delete"> <br><br>
{% endif %}


{% if perms.enroll %}
<input type="button" value="Delete"> <br><br>
{% endif %}
```