Authentication

REST framework provides a number of authentication schemes out of the box, and also allows you to implement custom schemes.

- BasicAuthentication
- SessionAuthentication
- TokenAuthentication
- RemoteUserAuthentication
- Custom authentication

SessionAuthentication

This authentication scheme uses Django's default session backend for authentication. Session authentication is appropriate for AJAX clients that are running in the same session context as your website.

If successfully authenticated, SessionAuthentication provides the following credentials.

request.user will be a Django User instance.

request.auth will be None.

Unauthenticated responses that are denied permission will result in an HTTP 403 Forbidden response.

SessionAuthentication

If you're using an AJAX style API with SessionAuthentication, you'll need to make sure you include a valid CSRF token for any "unsafe" HTTP method calls, such as PUT, PATCH, POST or DELETE requests.

Warning: Always use Django's standard login view when creating login pages. This will ensure your login views are properly protected.

CSRF validation in REST framework works slightly differently to standard Django due to the need to support both session and non-session based authentication to the same views. This means that only authenticated requests require CSRF tokens, and anonymous requests may be sent without CSRF tokens. This behaviour is not suitable for login views, which should always have CSRF validation applied.

Permission

Permissions are used to grant or deny access for different classes of users to different parts of the API.

Permission checks are always run at the very start of the view, before any other code is allowed to proceed.

Permission checks will typically use the authentication information in the *request.user* and *request.auth* properties to determine if the incoming request should be permitted.

Permission Classes

Permissions in REST framework are always defined as a list of permission classes.

- AllowAny
- IsAuthenticated
- IsAdminUser
- IsAuthenticatedOrReadOnly
- DjangoModelPermissions
- DjangoModelPermissionsOrAnonReadOnly
- DjangoObjectPermissions
- Custom Permissions

AllowAny

The AllowAny permission class will allow unrestricted access, regardless of if the request was authenticated or unauthenticated.

This permission is not strictly required, since you can achieve the same result by using an empty list or tuple for the permissions setting, but you may find it useful to specify this class because it makes the intention explicit.

IsAuthenticated

The IsAuthenticated permission class will deny permission to any unauthenticated user, and allow permission otherwise.

This permission is suitable if you want your API to only be accessible to registered users.

IsAdminUser

The IsAdminUser permission class will deny permission to any user, unless user.is_staff is True in which case permission will be allowed.

This permission is suitable if you want your API to only be accessible to a subset of trusted administrators.

IsAuthenticatedOrReadOnly

The IsAuthenticatedOrReadOnly will allow authenticated users to perform any request. Requests for unauthorised users will only be permitted if the request method is one of the "safe" methods; GET, HEAD or OPTIONS.

This permission is suitable if you want to your API to allow read permissions to anonymous users, and only allow write permissions to authenticated users.

DjangoModelPermissions

This permission class ties into Django's standard django.contrib.auth model permissions. This permission must only be applied to views that have a *queryset* property set. Authorization will only be granted if the user is authenticated and has the relevant model permissions assigned.

- POST requests require the user to have the add permission on the model.
- PUT and PATCH requests require the user to have the change permission on the model.
- DELETE requests require the user to have the delete permission on the model.

The default behaviour can also be overridden to support custom model permissions. For example, you might want to include a view model permission for GET requests.

To use custom model permissions, override DjangoModelPermissions and set the *perms_map* property.

DjangoModelPermissionsOrAnonReadOnly

Similar to DjangoModelPermissions, but also allows unauthenticated users to have read-only access to the API.

DjangoObjectPermissions

This permission class ties into Django's standard object permissions framework that allows per-object permissions on models. In order to use this permission class, you'll also need to add a permission backend that supports object-level permissions, such as django-guardian.

As with DjangoModelPermissions, this permission must only be applied to views that have a queryset property or get_queryset() method. Authorization will only be granted if the user is authenticated and has the relevant per-object permissions and relevant model permissions assigned.

- POST requests require the user to have the add permission on the model instance.
- PUT and PATCH requests require the user to have the change permission on the model instance.
- DELETE requests require the user to have the delete permission on the model instance.

Custom Permissions

To implement a custom permission, override BasePermission and implement either, or both, of the following methods:

- has_permission(self, request, view)
- has_object_permission(self, request, view, obj)

The methods should return True if the request should be granted access, and False otherwise.

Custom Permissions

custompermissions.py

```
class MyPermission(BasePermission):
    def has_permission(self, request, view)
```

Permission

Third party packages:-

- DRF - Access Policy
- Composed Permissions
- REST Condition
- DRY Rest Permissions
- Django Rest Framework Roles
- Django REST Framework API Key
- Django Rest Framework Role Filters
- Django Rest Framework PSQ