Django-Rest-Durin

Release v1.0.0

Eshaan Bansal

Jan 20, 2022

SETUP

Per API client token authentication Module for Django-REST-Framework

The idea is to provide one library that does token auth for multiple Web/CLI/Mobile API clients via one interface but allows different token configuration for each client.

Durin authentication is token based, similar to the `TokenAuthentication` built in to DRF. However, it adds some extra sauce:

- Durin allows **multiple tokens** per user. But only one token each user per API client.
- Each user token is associated with an API Client.
  - These API Clients (*`durin.models.Client`*) are configurable via Django's Admin Interface.
  - Includes Permission-enforcing to allow only specific clients to make authenticated requests to certain `APIViews` or vice-a-versa.
  - Configure Rate-Throttling per User <-> Client pair.
- All Durin **tokens have an expiration time**. This expiration time can be different per API client.
- Durin provides an option for a logged in user to **remove all tokens** that the server has - forcing him/her to re-authenticate for all API clients.
- Durin **tokens can be renewed** to get a fresh expiry.
- Durin provides a *`durin.auth.CachedTokenAuthentication`* backend as well which uses memoization for faster look ups.
- **Durin provides Session-Management features. Refer to Session-Management-Views i.e.,**
  - REST view for an authenticated user to get list of sessions (in context of django-rest-durin, this means `AuthToken` instances) and revoke a session. Useful for pages like "View active browser sessions".
  - REST view for an authenticated user to get/create/delete token against a pre-defined client. Useful for pages like "Get API key" where a user can get an API key to be able to interact directly with your project's RESTful API using cURL or a custom client.

CHAPTER

ONE

INDEX

Get started at *Installation*.

1.1 Installation

1.1.1 Django Compatibility Matrix

If your project uses an older verison of Django or Django Rest Framework, you can choose an older version of this project.

This Project	Python Version	Django Version	Django Rest Framework
0.1+	3.5 - 3.10	2.2, 3.0, 3.1, 3.2, 4.0	3.7>=

Make sure to use at least `DRF 3.10` when using `Django 3.0` or newer.

1.1.2 Install Durin

Durin should be installed with `pip`:

```
$ pip install django-rest-durin
```

1.1.3 Setup Durin

- Add `rest_framework` and `durin` to your `INSTALLED_APPS`, remove `rest_framework.authtoken` if you were using it.:

```
INSTALLED_APPS = (
  ...
  'rest_framework',
  'durin',
  ...
)
```

- Make Durin's *`durin.auth.TokenAuthentication`* your default authentication class for django-rest-framework:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': ('durin.auth.TokenAuthentication',),
    ...
}
```

- Add the Durin's *URLs (durin.urls)* patterns to your project.
- Customize Durin's *Settings (durin.settings)* for your project.
- Apply the migrations for the models:

```
$ python manage.py migrate
```

Hint: To use the cache backend for faster lookups, see *durin.auth.CachedTokenAuthentication*.

1.2 Settings (durin.settings)

Settings in durin are handled in a similar way to the rest framework settings. All settings are namespaced in the 'REST_DURIN' setting.

Example settings.py:

```
#...snip...
# These are the default values if none are set
from datetime import timedelta
from rest_framework.settings import api_settings
REST_DURIN = {
        "DEFAULT_TOKEN_TTL": timedelta(days=1),
        "TOKEN_CHARACTER_LENGTH": 64,
        "USER_SERIALIZER": None,
        "AUTH_HEADER_PREFIX": "Token",
        "EXPIRY_DATETIME_FORMAT": api_settings.DATETIME_FORMAT,
        "TOKEN_CACHE_TIMEOUT": 60,
        "REFRESH_TOKEN_ON_LOGIN": False,
        "AUTHTOKEN_SELECT_RELATED_LIST": ["user"],
        "API_ACCESS_CLIENT_NAME": None,
        "API_ACCESS_EXCLUDE_FROM_SESSIONS": False,
        "API_ACCESS_RESPONSE_INCLUDE_TOKEN": False,
}
#...snip...
```

DEFAULT_TOKEN_TTL

Default: timedelta(days=1)

This is how long a token can exist before it expires. Expired tokens are automatically removed from the system.

The setting should be set to an instance of datetime.timedelta.

Durin provides setting a different token Time To Live (token_ttl) value per client object. So this is the default value the *durin.models.Client* model uses incase a custom value wasn't specified.

Warning: setting a 0 or negative timedelta will create tokens that instantly expire, the system will not prevent you setting this.

TOKEN_CHARACTER_LENGTH

Default: 64

This is the length of the token that will be sent to the client. This shouldn't need changing.

USER_SERIALIZER
Default: `None`

This is the reference to the class used to serialize the `User` objects when succesfully returning from *durin.views.LoginView*. The default is *durin.serializers.UserSerializer*.

AUTH_HEADER_PREFIX
Default: `"Token"`

This is the Authorization header value prefix.

EXPIRY_DATETIME_FORMAT
Default: DATETIME_FORMAT (of Django REST framework)

This is the expiry datetime format returned in the login and refresh views.

May be any of `None`, `iso-8601` or a Python strftime format string.

TOKEN_CACHE_TIMEOUT
Default: `60`

This is the cache timeout (in seconds) used by `django-memoize` in case you are using *durin.auth.CachedTokenAuthentication* backend in your app.

REFRESH_TOKEN_ON_LOGIN
Default: `False`

When a request is made to the *durin.views.LoginView*. One of two things happen:

1. Token instance for a particular user-client pair already exists.
2. A new token instance is generated for the provided user-client pair.

In the first case, the already existing token is sent in response. So this setting if set to `True` should extend the expiry time of the token by it's *durin.models.Client* `token_ttl` everytime login happens.

AUTHTOKEN_SELECT_RELATED_LIST
Default: `["user"]`

This is passed as an argument to `select_related` when the *durin.auth.TokenAuthentication* class fetches the *durin.models.AuthToken* instance. For example,

```
AuthToken.objects.select_related(*AUTHTOKEN_SELECT_RELATED_LIST).get(token=token_
↪string)
```

Otherwise, set to a falsy value such as `None` or `False` to not use `select_related`.

Hint: Refer to Django's select_related docs to see how this can boost performance by reducing number of SQL queries made.

API_ACCESS_CLIENT_NAME
Default: `None`

There may be an use-case where you want to issue API keys to your users so they can call your RESTful API using cURL or a custom client.

Set this setting to the ``name` of the specific *durin.models.Client* instance to issue these API keys against.

Note: The *durin.views.APIAccessTokenView* view allows management of this.

API_ACCESS_EXCLUDE_FROM_SESSIONS
Default: `False`

If set to `True`, the `AuthToken` instance for the specifc `API_ACCESS_CLIENT_NAME`'s *Client`* instance will be excluded from the overall "Sessions List" (`GET /api/sessions/`) response.

This is useful because you may want the view to list only the "browser sessions".

API_ACCESS_RESPONSE_INCLUDE_TOKEN
Default: `False`

If set to `False`, the `token` field would be omitted from the *durin.views.APIAccessTokenView* view's (`GET /api/apiaccess/`) response.

In case of `POST` request, the `token` field is always included despite of this setting.

1.3 Authentication (`durin.auth`)

Durin provides one `TokenAuthentication` backend and `CachedTokenAuthentication` which uses memoization for faster look ups.

1.3.1 TokenAuthentication

class durin.auth.**TokenAuthentication**
Bases: `rest_framework.authentication.BaseAuthentication`

This authentication scheme uses Durin's *durin.models.AuthToken* for authentication.

Similar to DRF's authentication system, it overrides it a bit to accomodate that tokens can be expired.

If successful,

- `request.user` will be a django `User` instance
- `request.auth` will be an `AuthToken` instance

Durin tokens should be generated using the provided views. Any `APIView` or `ViewSet` can be accessed using these tokens by adding `TokenAuthentication` to the View's `authentication_classes`. To authenticate, the `Authorization` header should be set on the request, like:

```
Authorization: Token adee69d0e4bbdc6e4m9836F45E23A325
```

Note: The prefix can be configured by setting the `REST_DURIN["AUTH_HEADER_PREFIX"]` (ref).

Example Usage:

```
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework.views import APIView

from durin.auth import TokenAuthentication

class ExampleView(APIView):
    authentication_classes = (TokenAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request, *args, **kwargs):
        content = {
```

(continues on next page)

(continued from previous page)

```
            'foo': 'bar'
        }
        return Response(content)
```

Tokens expire after a preset time. See settings.DEFAULT_TOKEN_TTL.

1.3.2 CachedTokenAuthentication

class durin.auth.**CachedTokenAuthentication**

Bases: *durin.auth.TokenAuthentication*

Similar to `TokenAuthentication` but uses django-cache-memoize as cache backend for faster lookups.

The cache timeout is configurable by setting the `REST_DURIN["TOKEN_CACHE_TIMEOUT"]` under your app's `settings.py`.

How To Enable:

1. Install django-cache-memoize

```
pip install django-cache-memoize
```

2. Then you need to use `CachedTokenAuthentication` instead of `TokenAuthentication`.

1.3.3 Global usage on all views

You can activate Durin's *durin.auth.TokenAuthentication* or *durin.auth.CachedTokenAuthentication* on all your views by adding it to `REST_FRAMEWORK["DEFAULT_AUTHENTICATION_CLASSES"]` under your app's `settings.py`. Make sure to not use both of these together.

> **Warning:** If you use *Token Authentication* in production you must ensure that your API is only available over HTTPS (SSL).

1.4 Views (`durin.views`)

Durin provides four views that handle token management for you. And two additional views to allow sessions management.

1.4.1 Auth Management Views

LoginView

class durin.views.**LoginView**(***kwargs*)

Bases: rest_framework.views.APIView

Durin's Login View.

This view will return a JSON response when valid `username`, `password` and (if not overwritten) `client` fields are POSTed to the view using form data or JSON.

It uses the default serializer provided by Django-Rest-Framework (`rest_framework.authtoken.serializers.AuthTokenSerializer`) to validate the user credentials.

It is possible to customize LoginView behaviour by overriding the following helper methods:

static format_expiry_datetime(*expiry: datetime.datetime*) → str
To format the expiry `datetime` object at your convenience.

get_client_obj(*request*) → *durin.models.Client*
To get and return the associated *`durin.models.Client`* object.

:raises rest_framework.exceptions.ValidationError

get_context()
to change the context passed to the `UserSerializer`.

get_post_response_data(*request*, *token_obj:* durin.models.AuthToken) → dict
Override this to return a fully customized payload.

get_token_obj(*request*, *client:* durin.models.Client) → *durin.models.AuthToken*
Flow used to return the *`durin.models.AuthToken`* object.

get_user_serializer_class()
To change the class used for serializing the user.

renew_token(*request*, *token:* durin.models.AuthToken) → None
How to renew the token instance in case `settings.REFRESH_TOKEN_ON_LOGIN` is set to `True`.

Response Data and User Serialization

When the endpoint authenticates a request, a JSON object will be returned containing the `token` as a string, `expiry` as a timestamp for when the token expires.

This is because `USER_SERIALIZER` setting is `None` by default.

If you wish to return custom data upon successful authentication like `first_name`, `last_name`, and `username` then the included `UserSerializer` class can be used inside `REST_DURIN` settings by adding *`durin.serializers.UserSerializer`*.

Obviously, if your app uses a custom user model that does not have these fields, a custom serializer must be used.

Client Configuration

In most cases, you would want to customize how the login view gets the client object to associate with the token. By default, it is the `client` attribute in POSTed request body. Here's an example snippet of how you can override this behaviour:

```
### views.py:

from durin.models import Client as APIClient
from durin.views import LoginView as DurinLoginView

class LoginView(DurinLoginView):

    @staticmethod
    def get_client_obj(request):
        # get the client's name from a request header
        client_name = request.META.get("X-my-personal-header", None)
        if not client_name:
            raise ParseError("No client specified.", status.HTTP_400_BAD_REQUEST)
        return APIClient.objects.get_or_create(name=client_name)


### urls.py:

from durin import views as durin_views
from yourapp.views import LoginView

urlpatterns = [
    url(r'login/', LoginView.as_view(), name='durin_login'),
    url(r'refresh/', durin_views.RefreshView.as_view(), name='durin_refresh'),
    url(r'logout/', durin_views.LogoutView.as_view(), name='durin_logout'),
    url(r'logoutall/', durin_views.LogoutAllView.as_view(), name='durin_logoutall'),
]
```

RefreshView

class durin.views.**RefreshView**(***kwargs*)

Bases: `rest_framework.views.APIView`

Durin's Refresh View

This view accepts only a post request with an empty body. It responds to Durin Token Authentication. On a successful request,

1. The given token's expiry is extended by it's associated *durin.models.Client.token_ttl* duration and a JSON object will be returned containing a single `expiry` key as the new timestamp for when the token expires.
2. *durin.signals.token_renewed()* is called.

static format_expiry_datetime(*expiry: datetime.datetime*) → str

To format the expiry `datetime` object at your convenience.

renew_token(*request*, *token:* durin.models.AuthToken) → datetime.datetime

How to renew the token instance.

LogoutView

class durin.views.**LogoutView**(***kwargs*)
Bases: rest_framework.views.APIView

Durin's Logout View.

This view accepts only a post request with an empty body. It responds to Durin Token Authentication. On a successful request,

1. The token used to authenticate is deleted from the database and can no longer be used to authenticate.
2. django.contrib.auth.signals.user_logged_out() is called.

Returns 204 (No content)

LogoutAllView

class durin.views.**LogoutAllView**(***kwargs*)
Bases: rest_framework.views.APIView

Durin's LogoutAllView.

This view accepts only a post request with an empty body. It responds to Durin Token Authentication. On a successful request,

1. The token used to authenticate, and **all other tokens** registered to the same User account, are deleted from the system and can no longer be used to authenticate.
2. django.contrib.auth.signals.user_logged_out() is called.

Returns 204 (No content)

Note: It is not recommended to alter the Logout views. They are designed specifically for token management, and to respond to durin authentication. Modified forms of the class may cause unpredictable results.

1.4.2 Session Management Views

TokenSessionsViewSet

class durin.views.**TokenSessionsViewSet**(***kwargs*)
Bases: rest_framework.mixins.ListModelMixin, rest_framework.mixins.DestroyModelMixin, rest_framework.viewsets.GenericViewSet

Durin's TokenSessionsViewSet.

- Returns list of active sessions of authed user.
- Only list() and delete() operations.

New in version 1.0.0.

APIAccessTokenView

class durin.views.**APIAccessTokenView**(***kwargs*)

Bases: rest_framework.views.APIView

Durin's APIAccessTokenView.

- GET -> get token-client pair info
- POST -> create and get token-client pair info
- DELETE -> delete existing API access token

New in version 1.0.0.

1.5 URLs (durin.urls)

Durin provides a URL config ready with its 6 default views routed.

This can easily be included in your url config:

```
urlpatterns = [
  #...snip...
  re_path(r'api/auth/', include('durin.urls'))
  #...snip...
]
```

Note: It is important to use the string syntax and not try to import durin.urls, as the reference to the User model will cause the app to fail at import time.

The views would then accessible as:

- /api/auth/login -> LoginView
- /api/auth/refresh - RefreshView
- /api/auth/logout -> LogoutView
- /api/auth/logoutall -> LogoutAllView
- /api/auth/sessions -> TokenSessionsViewSet
- /api/auth/apiaccess -> APIAccessTokenView

they can also be looked up by name:

```
from rest_framework import reverse

reverse('durin_login')
reverse('durin_logout')
reverse('durin_refresh')
reverse('durin_logoutall')
reverse('durin_tokensessions-list')
reverse('durin_apiaccess')
```

1.6 Signals (durin.signals)

Durin provides 2 custom signals that can be subscribed to the same way as done for Django's Inbuilt Signals.

1.6.1 token_expired

durin.signals.**token_expired = <django.dispatch.dispatcher.Signal object>**
When a token is expired and deleted.

providing_args=["username", "source"]

1.6.2 token_renewed

durin.signals.**token_renewed = <django.dispatch.dispatcher.Signal object>**
When a token is renewed by either *durin.views.LoginView* or *durin.views.RefreshView*.

providing_args=["request", "new_expiry"]

1.7 Models (durin.models)

class durin.models.**AuthToken**(**args*, ***kwargs*)
Token model with a unique constraint on `User` <-> `Client` relationship.

client
Client ForeignKey

created
Created time

property expires_in
Dynamic property that gives the *expiry* attribute in human readable string format.

Uses humanize package.

expiry
Expiry time

property has_expired
Dynamic property that returns `True` if token has expired, otherwise `False`.

renew_token(*request=None*) → datetime.datetime
Utility function to renew the token.

Updates the *expiry* attribute by `Client.token_ttl`.

token
Token string

user
`User` ForeignKey

class durin.models.**Client**(**args*, ***kwargs*)
Identifier to represent any API client/browser that consumes your RESTful API.

See `example_project.models.ClientSettings` if you wish to extend this model per your convenience.

name
A unique identification name for the client.

throttle_rate
Throttle rate for requests authed with this client.

Format: `number_of_requests/period` where period should be one of: *('s', 'm', 'h', 'd')*. (same format as DRF's throttle rates)

Example: `100/h` implies 100 requests each hour.

New in version 0.2.

token_ttl
Token Time To Live (TTL) in timedelta. Format: `DAYS HH:MM:SS`.

1.8 Permissions (durin.permissions)

Durin provides two *abstract* permission classes which make use of the *durin.models.Client* model it offers.

You will need to subclass them and modify the `allowed_clients_name` or `disallowed_clients_name` property per your wish.

Then you may use them the same way as other DRF permissions or activate them on all your views by adding them to `REST_FRAMEWORK["DEFAULT_PERMISSION_CLASSES"]` under your app's `settings.py`

1.8.1 AllowSpecificClients

class durin.permissions.**AllowSpecificClients**
Bases: `rest_framework.permissions.BasePermission`

Allows access to only specific clients.

Should be used along with *Authentication (durin.auth)*.

allowed_clients_name = ()
Include names of allowed clients to `allowed_clients_name`.

has_permission(*request*, *view*)
Return *True* if permission is granted, *False* otherwise.

1.8.2 DisallowSpecificClients

class durin.permissions.**DisallowSpecificClients**
Bases: rest_framework.permissions.BasePermission

Restrict specific clients from making requests.

Should be used along with *Authentication (durin.auth)*.

disallowed_clients_name = ()
Include names of disallowed clients to disallowed_clients_name.

has_permission(*request*, *view*)
Return *True* if permission is granted, *False* otherwise.

1.9 Throttling (durin.throttling)

Durin provides a throttling class which make use of the *durin.models.Client* model it offers.

Usage is the same way as other DRF throttling classes.

Example settings.py:

```
#...snip...
REST_FRAMEWORK = {
    "DEFAULT_THROTTLE_CLASSES": ["durin.throttling.UserClientRateThrottle"],
    "DEFAULT_THROTTLE_RATES": {"user_per_client": "10/min"},
}
#...snip...
```

"user_per_client"
default scope for the *UserClientRateThrottle* class.

The rate defined here serves as the default rate incase the throttle_rate field on *durin.models.Client* is null.

1.9.1 UserClientRateThrottle

class durin.throttling.**UserClientRateThrottle**
Bases: rest_framework.throttling.UserRateThrottle

Throttles requests by identifying the *authed* **user-client pair**.

This is useful if you want to define different user throttle rates per *durin.models.Client* instance.

New in version 0.2.

allow_request(*request*, *view*)
The rate is set here because we need access to request object which is not available inside get_rate().

cache_format = 'throttle_%(scope)s_%(ident)s'
Same as the default

get_cache_key(*request*, *view*) → str
Should return a unique cache-key which can be used for throttling. Must be overridden.

May return *None* if the request should not be throttled.

scope = 'user_per_client'
Scope for this throttle

static validate_client_throttle_rate(*rate*)
Used for validating the `throttle_rate` field on *durin.models.Client*.

For internal use only.

1.10 Other submodules

Note: If you are looking for information on a specific function, class or method, this part of the documentation is for you.

1.10.1 durin.admin module

class durin.admin.**AuthTokenAdmin**(*model*, *admin_site*)
Django's ModelAdmin for AuthToken.

In most cases, you would want to override this to make `AuthTokenAdmin.raw_id_fields = ("user",)`

class durin.admin.**ClientAdmin**(*model*, *admin_site*)
Django's ModelAdmin for `Client` model.

1.10.2 durin.serializers module

class durin.serializers.**APIAccessTokenSerializer**(**args*, ***kwargs*)
Used in *durin.views.APIAccessTokenView*.

New in version 1.0.0.

class durin.serializers.**TokenSessionsSerializer**(**args*, ***kwargs*)
Used in *durin.views.TokenSessionsViewSet*.

New in version 1.0.0.

class durin.serializers.**UserSerializer**(**args*, ***kwargs*)

1.11 FAQ: Why use durin over JWT or other libraries ?

Good question.

Authentication is tricky. There are many libraries available for DRF which provide token authentication. I've personally used drf-simplejwt and django-rest-knox and they are both great at their *implementation*.

So why would you want to use Django-Rest-Durin ?

Here are a few use cases which I needed (and why it lead me to create durin) and might help you make a better decision too,

- If you'd like to use Django's Admin interface to manage the different clients which consume your API.

- If you want the token expiration to be dependent on what API client it is meant for. For example, you might want to create tokens which never expire for a Command Line Client but want a shorter expiry for a JavaScript (web) client.
- If you want to limit number of tokens allowed per user.
- If you'd like to refresh token expiry without changing token key.
- If you or your organization are interested in Client Level Analytics such as keeping track of which user uses what client the most, etc.
- If you want to restrict certian `APIView` or `Viewsets` to allow authenticated requests from only specific clients of your choice.

.... and more. Make a PR on GitHub to tell us what you use durin for!

1.12 Development

If you would like to contribute to django-rest-durin, you can clone the repository from GitHub.

```
git clone https://github.com/Eshaan7/django-rest-durin
```

Extra dependencies required during testing or development can be installed with:

```
pip install django-rest-durin[dev]
```

Before committing your changes with git or pushing them to remote, please run the following:

```
bash pre-commit.sh
```

1.13 Run the tests locally

If you need to debug a test locally and if you have docker installed:

simply run the `./docker-run-tests.sh` script and it will run the test suite in every Python / Django versions.

You could also simply run regular `tox` in the root folder as well, but that would make testing the matrix of Python / Django versions a bit more tricky.

1.14 Changelog

1.14.1 v1.0.0

Note: If in your `urls.py` you have a url pattern with `include("durin.urls")`, then 2 new URL paths `apiaccess/` and `sessions/` will get added to your project if you upgrade to this version.

If you do not wish to have these new views, remove the above include statement and refer to durin/urls.py on how to define the URL patterns selectively for the views you want.

Features:

- Session Management serializers and views. (Issue 19)

Refer to Session-Management-Views i.e.,

- REST view for an authenticated user to get list of sessions (in context of django-rest-durin, this means `AuthToken` instances) and revoke a session. Useful for pages like "View active browser sessions".
- REST view for an authenticated user to get/create/delete token against a pre-defined client. Useful for pages like "Get API key" where a user can get an API key to be able to interact directly with your project's RESTful API using cURL or a custom client.

1.14.2 v0.4.0

Breaking Changes:

- Remove the hard-coding of `authentication_classes`, `permission_classes` variables in *Views (durin.views)*. Meaning they will now use the defaults set under `REST_FRAMEWORK` in `settings.py`.

Other:

- Support for Python 3.10. Enable CI tests for same.
- Support for Django 4.0. Enable CI tests for same.

1.14.3 v0.3.0

Features:

- AUTHTOKEN_SELECT_RELATED_LIST setting to enable performance optimization. (Issue 16)

Other:

- More advanced use-cases in example_project/permissions.py.
- Test cases now cover the *Permissions (durin.permissions)*.

1.14.4 v0.2.0

Breaking Changes:

- Replace `django-memoize` with `django-cache-memoize` package. Refer to updated *`durin.auth.CachedTokenAuthentication`*. (Issue 13)
- Update arguments passed to durin's signals and remove `providing_args` argument (Django deprecation notice). Please see updated *Signals (durin.signals)*.
- The `get_client_obj`, `get_token_obj` and `renew_token` member methods of *`durin.views.LoginView`* are no longer `staticmethod` or `classmethod`.

Features:

- *`durin.throttling.UserClientRateThrottle`* throttle class. (Issue 9)
- `ClientSettings` model in example_project. (Issue 14)
- `renew_token` method on *`durin.views.RefreshView`* to enable easier extensibility.

Bug Fixes:

- Fix bug in `AuthTokenAdminView` ("save and continue editing" button was not working).
- Exception handling was missing in `get_client_obj` member method of *`durin.views.LoginView`*.

Other:

- Enable CI tests for Django 3.2.
- Better document `models.py` and categorize modules in documentation.

1.14.5 v0.1.0

- Initial release

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

d

INDEX

L

M

N

R

S

T

U

V