Settings

Some of Simple JWT's behavior can be customized through settings variables in `settings.py`:

```
# Django project settings.py

from datetime import timedelta
...

SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=5),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "ROTATE_REFRESH_TOKENS": False,
    "BLACKLIST_AFTER_ROTATION": False,
    "UPDATE_LAST_LOGIN": False,

    "ALGORITHM": "HS256",
    "SIGNING_KEY": settings.SECRET_KEY,
    "VERIFYING_KEY": "",
    "AUDIENCE": None,
    "ISSUER": None,
    "JSON_ENCODER": None,
    "JWK_URL": None,
    "LEEWAY": 0,

    "AUTH_HEADER_TYPES": ("Bearer",),
    "AUTH_HEADER_NAME": "HTTP_AUTHORIZATION",
    "USER_ID_FIELD": "id",
    "USER_ID_CLAIM": "user_id",
    "USER_AUTHENTICATION_RULE":
"rest_framework_simplejwt.authentication.default_user_authentication_rule",

    "AUTH_TOKEN_CLASSES": ("rest_framework_simplejwt.tokens.AccessToken",),
    "TOKEN_TYPE_CLAIM": "token_type",
    "TOKEN_USER_CLASS": "rest_framework_simplejwt.models.TokenUser",

    "JTI_CLAIM": "jti",

    "SLIDING_TOKEN_REFRESH_EXP_CLAIM": "refresh_exp",
    "SLIDING_TOKEN_LIFETIME": timedelta(minutes=5),
    "SLIDING_TOKEN_REFRESH_LIFETIME": timedelta(days=1),

    "TOKEN_OBTAIN_SERIALIZER": "rest_framework_simplejwt.serializers.TokenObtainPairSerializer",
    "TOKEN_REFRESH_SERIALIZER": "rest_framework_simplejwt.serializers.TokenRefreshSerializer",
    "TOKEN_VERIFY_SERIALIZER": "rest_framework_simplejwt.serializers.TokenVerifySerializer",
    "TOKEN_BLACKLIST_SERIALIZER": "rest_framework_simplejwt.serializers.TokenBlacklistSerializer",
    "SLIDING_TOKEN_OBTAIN_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenObtainSlidingSerializer",
    "SLIDING_TOKEN_REFRESH_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenRefreshSlidingSerializer",
}
```

Above, the default values for these settings are shown.

`ACCESS_TOKEN_LIFETIME`

A `datetime.timedelta` object which specifies how long access tokens are valid. This `timedelta` value is added to the current UTC time during token generation to obtain the token's default "exp" claim value.

`REFRESH_TOKEN_LIFETIME`

A `datetime.timedelta` object which specifies how long refresh tokens are valid. This `timedelta` value is added to the current UTC time during token generation to obtain the token's default "exp" claim value.

`ROTATE_REFRESH_TOKENS`

When set to `True`, if a refresh token is submitted to the `TokenRefreshView`, a new refresh token will be returned along with the new access token. This new refresh token will be supplied via a "refresh" key in the JSON response. New refresh tokens will have a renewed expiration time which is determined by adding the timedelta in the `REFRESH_TOKEN_LIFETIME` setting to the current time when the request is made. If the blacklist app is in use and the `BLACKLIST_AFTER_ROTATION` setting is set to `True`, refresh tokens submitted to the refresh view will be added to the blacklist.

`BLACKLIST_AFTER_ROTATION`

When set to `True`, causes refresh tokens submitted to the `TokenRefreshView` to be added to the blacklist if the blacklist app is in use and the `ROTATE_REFRESH_TOKENS` setting is set to `True`. You need to add `'rest_framework_simplejwt.token_blacklist',` to your `INSTALLED_APPS` in the settings file to use this setting.

Learn more about Blacklist app.

`UPDATE_LAST_LOGIN`

When set to `True`, last_login field in the auth_user table is updated upon login (TokenObtainPairView).

> Warning: Updating last_login will dramatically increase the number of database transactions. People abusing the views could slow the server and this could be a security vulnerability. If you really want this, throttle the endpoint with DRF at the very least.

`ALGORITHM`

The algorithm from the PyJWT library which will be used to perform signing/verification operations on tokens. To use symmetric HMAC signing and verification, the following algorithms may be used: `'HS256'`, `'HS384'`, `'HS512'`. When an HMAC algorithm is chosen, the `SIGNING_KEY` setting will be used as both the signing key and the verifying key. In that case, the `VERIFYING_KEY` setting will be ignored. To use asymmetric RSA signing and verification, the following algorithms may be used: `'RS256'`, `'RS384'`, `'RS512'`. When an RSA algorithm is chosen, the `SIGNING_KEY` setting must be set to a string that contains an RSA private key. Likewise, the `VERIFYING_KEY` setting must be set to a string that contains an RSA public key.

`SIGNING_KEY`

The signing key that is used to sign the content of generated tokens. For HMAC signing, this should be a random string with at least as many bits of data as is required by the signing protocol. For RSA signing, this should be a string that contains an RSA private key that is 2048 bits or longer. Since Simple JWT defaults to using 256-bit HMAC signing, the `SIGNING_KEY` setting defaults to the value of the `SECRET_KEY` setting for your django project. Although this is the most reasonable default that Simple JWT can provide, it is recommended that developers change this setting to a value that is independent from the django project secret key. This will make changing the signing key used for tokens easier in the event that it is compromised.

`VERIFYING_KEY`

The verifying key which is used to verify the content of generated tokens. If an HMAC algorithm has been specified by the `ALGORITHM` setting, the `VERIFYING_KEY` setting will be ignored and the value of the `SIGNING_KEY` setting will be used. If an RSA algorithm has been specified by the `ALGORITHM` setting, the `VERIFYING_KEY` setting must be set to a string that contains an RSA public key.

`AUDIENCE`

The audience claim to be included in generated tokens and/or validated in decoded tokens. When set to `None`, this field is excluded from tokens and is not validated.

`ISSUER`

The issuer claim to be included in generated tokens and/or validated in decoded tokens. When set to `None`, this field is excluded from tokens and is not validated.

`JWK_URL`

The JWK_URL is used to dynamically resolve the public keys needed to verify the signing of tokens. When using Auth0 for example you might set this to 'https://yourdomain.auth0.com/.well-known/jwks.json'. When set to `None`, this field is excluded from the token backend and is not used during validation.

`LEEWAY`

Leeway is used to give some margin to the expiration time. This can be an integer for seconds or a `datetime.timedelta`. Please reference https://pyjwt.readthedocs.io/en/latest/usage.html#expiration-time-claim-exp for more information.

`AUTH_HEADER_TYPES`

The authorization header type(s) that will be accepted for views that require authentication. For example, a value of `'Bearer'` means that views requiring authentication would look for a header with the following format: `Authorization: Bearer <token>`. This setting may also contain a list or tuple of possible header types (e.g. `('Bearer', 'JWT')`). If a list or tuple is used in this way, and authentication fails, the first item in the collection will be used to build the "WWW-Authenticate" header in the response.

`AUTH_HEADER_NAME`

The authorization header name to be used for authentication. The default is `HTTP_AUTHORIZATION` which will accept the `Authorization` header in the request. For example if you'd like to use `X_Access_Token` in the header of your requests please specify the `AUTH_HEADER_NAME` to be `HTTP_X_ACCESS_TOKEN` in your settings.

`USER_ID_FIELD`

The database field from the user model that will be included in generated tokens to identify users. It is recommended that the value of this setting specifies a field that does not normally change once its initial value is chosen. For example, specifying a "username" or "email" field would be a poor choice since an account's username or email might change depending on how account management in a given service is designed. This could allow a new account to be created with an old username while an existing token is still valid which uses that username as a user identifier.

`USER_ID_CLAIM`

The claim in generated tokens which will be used to store user identifiers. For example, a setting value of `'user_id'` would mean generated tokens include a "user_id" claim that contains the user's identifier.

`USER_AUTHENTICATION_RULE`

Callable to determine if the user is permitted to authenticate. This rule is applied after a valid token is processed. The user object is passed to the callable as an argument. The default rule is to check that the `is_active` flag is still `True`. The callable must return a boolean, `True` if authorized, `False` otherwise resulting in a 401 status code.

`AUTH_TOKEN_CLASSES`

A list of dot paths to classes that specify the types of token that are allowed to prove authentication. More about this in the "Token types" section below.

`TOKEN_TYPE_CLAIM`

The claim name that is used to store a token's type. More about this in the "Token types" section below.

`JTI_CLAIM`

The claim name that is used to store a token's unique identifier. This identifier is used to identify revoked tokens in the blacklist app. It may be necessary in some cases to use another claim besides the default "jti" claim to store such a value.

`TOKEN_USER_CLASS`

A stateless user object which is backed by a validated token. Used only for the JWTStatelessUserAuthentication authentication backend. The value is a dotted path to your subclass of `rest_framework_simplejwt.models.TokenUser`, which also is the default.

`SLIDING_TOKEN_LIFETIME`

A `datetime.timedelta` object which specifies how long sliding tokens are valid to prove authentication. This `timedelta` value is added to the current UTC time during token generation to obtain the token's default "exp" claim value. More about this in the "Sliding tokens" section below.

`SLIDING_TOKEN_REFRESH_LIFETIME`

A `datetime.timedelta` object which specifies how long sliding tokens are valid to be refreshed. This `timedelta` value is added to the current UTC time during token generation to obtain the token's default "exp" claim value. More about this in the "Sliding tokens" section below.

`SLIDING_TOKEN_REFRESH_EXP_CLAIM`

The claim name that is used to store the expiration time of a sliding token's refresh period. More about this in the "Sliding tokens" section below.

`CHECK_REVOKE_TOKEN`

If this field is set to `True`, the system will verify whether the token has been revoked or not by comparing the md5 hash of the user's current password with the value stored in the REVOKE_TOKEN_CLAIM field within the payload of the JWT token.

`REVOKE_TOKEN_CLAIM`

The claim name that is used to store a user hash password. If the value of this CHECK_REVOKE_TOKEN field is `True`, this field will be included in the JWT payload.