

# Django Project Base

version 0.1.6.7

July 13, 2021



# Welcome to Django Project Base's documentation!

## What is django-project-base?

We start with a project. Everything revolves around it: users, roles, permissions, tags, etc. Everything belongs to a project first, then to database. This project makes it easy to work on that premise. At the same time it integrates a few basic operations that you need in every project so that you don't have to do them over and over again.

This is a [django](#) library, based on [django-rest-framework](#) with [DynamicForms](#) and [Django REST Registration](#) integration.

## Why django-project-base?

Functionalities provided:

- A base Project definition and editor for it. Extend as you like.
- User profile editor. Manage emails, confirmations, social connections
- Support for REST-based authentication / session creation
- Session / user caching for speed
- Project users editor. Invite users to project. Assign them into roles.
- Roles management & rights assignment.
- Tags editor & manager + support API for marking tagged items with their colours or icons

Index:

## Installation

### Django project base

Install the package:

```
pip install django-project-base
```

Extend the BaseProject & BaseProfile model:

Django project base uses Swapper <https://pypi.org/project/swapper/>, an unofficial API for Django swappable models. You need to override the Project and Profile models before you can use the library: there aren't any migrations available in the library itself. The library only declares properties it itself supports, but you have the option to extend them as you wish to fit your needs too.

```
# myapp/models.py
from django_project_base import BaseProject

class MyProject(BaseProject):
    # add any fields & methods you like here

class MyProfile(BaseProfile):
    # add any fields & methods you like here
```

Then also make sure your swappable models are loaded instead of django-project-base models:

```
# myproject/settings.py

DJANGO_PROJECT_BASE_PROJECT_MODEL = 'myapp.MyProject'
DJANGO_PROJECT_BASE_PROFILE_MODEL = 'myapp.MyProfile'

Add to INSTALLED_APPS
'rest_registration',
'django_project_base',
'drf_spectacular',
```

```
Add:
    REST_FRAMEWORK = {
    # YOUR SETTINGS
    'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
    }
```

Append django project base urls:

```
# myproject/urls.py
urlpatterns = [
    ...
    path('', include('django_project_base.urls')),
    ...
]
```

There are some additional URLs available for the Django project base, like swagger or documentation. Appending those URLs is described in more details in suitable chapters.

## Dynamic Forms

Django project base is dependent on Dynamic Forms project <https://github.com/velis74/DynamicForms>

Read Dynamic Forms documentation for installation steps and more information about project.

You should add at least following code to your project, to enable Dynamic Forms.

```
# myproject/settings.py

REST_FRAMEWORK = {
    ...
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
        'dynamicforms.renderers.TemplateHTMLRenderer',
    )
    ...
}
```

## Environment setup

For code formatting use .jshintrc file present in repository. Set tab size, ident, continuation ident in your editor to 2 places.

For JS development go to <https://nodejs.org/en/> and install latest stable version of nodejs and npm. In project base directory run npm install. To run a development server run `npm run dev` (go to <http://0.0.0.0:8080/>). To generate a build run `npm run build`.

JS code is present in src directory. For web UI components library vuejs(<https://vuejs.org/>) is used. Components are built as Vue global components(<https://vuejs.org/v2/guide/components.html>) with x-templates. Templates are present in templates directory.

When developing webpack development server expects that service which provides data runs on host <http://127.0.0.1:8000>. This can be changed in webpack.config.js file. For running example django project prepare python environment and run (run in repository root):

- pip install -r requirements.txt (run in content root)
- python manage.py runserver

Try logging in with user “miha”, pass “mihamiha”.

## Settings options - quick overview

### DJANGO\_PROJECT\_BASE\_PROJECT\_MODEL

```
DJANGO_PROJECT_BASE_PROJECT_MODEL = 'myapp.MyProject'
```

Set swappable model for Django project base Project model. Read more in Django project base chapter.

## DJANGO\_PROJECT\_BASE\_PROFILE\_MODEL

```
DJANGO_PROJECT_BASE_PROFILE_MODEL = 'myapp.MyProfile'
```

Set swappable model for Django project base Profile model. Read more in Django project base chapter.

## DJANGO\_PROJECT\_BASE\_BASE\_REQUEST\_URL\_VARIABLES

```
DJANGO_PROJECT_BASE_BASE_REQUEST_URL_VARIABLES: {  
    'project': {'value_name': 'current_project_slug', 'url_part': 'project-'},  
    'language': {'value_name': 'current_language', 'url_part': 'language-'}  
}
```

A dictionary of attribute names on the request object. Read more in DJANGO\_PROJECT\_BASE\_BASE\_REQUEST\_URL\_VARIABLES chapter.

## DJANGO\_PROJECT\_BASE\_SLUG\_FIELD\_NAME

```
DJANGO_PROJECT_BASE_SLUG_FIELD_NAME: 'slug'
```

Read more in Project slug chapter.

## MAINTENENACE\_NOTIFICATIONS\_CACHE\_KEY

```
MAINTENENACE_NOTIFICATIONS_CACHE_KEY= ""
```

Read more in Maintenance notifications chapter.

## DJANGO\_USER\_CACHE

```
DJANGO_USER_CACHE='django-user-%d'
```

Key name for user caching background. Default value is 'django-user-%d'. Read more in User caching backend chapter.

## CACHE\_IMPERSONATE\_USER

```
CACHE_IMPERSONATE_USER = 'impersonate-user-%d'
```

Cache key name for impersonate user. Default value is 'impersonate-user-%d'. Read more in Impersonate user chapter.

## PROFILE\_REVERSE\_FULL\_NAME\_ORDER

```
PROFILE_REVERSE_FULL_NAME_ORDER = (bool)
```

Read more in Profile reverse name order chapter.

## DELETE\_PROFILE\_TIMEDELTA

```
DELETE_PROFILE_TIMEDELTA = 0
```

Value in days, when the automatic process should delete profile marked as for delete. Read more in Deleting profile chapter.

## DOCUMENTATION\_DIRECTORY

```
DOCUMENTATION_DIRECTORY='/docs/build/'
```

Path for documentation directory. Read more in [Documentation](#) chapter.

## Tags

Django project base supports tags usage. See example implementation bellow.

```
class DemoProjectTag(BaseTag):  
    content = models.CharField(max_length=20, null=True, blank=True)  
    class Meta:  
        verbose_name = "Tag"  
        verbose_name_plural = "Tags"  
  
class TaggedItemThrough(GenericTaggedItemBase):  
    tag = models.ForeignKey(  
        DemoProjectTag,
```

```

        on_delete=models.CASCADE,
        related_name="%s_%s_items" % (app_label, class_name),
    )

class Apartment(models.Model):
    number = fields.IntegerField()
    tags = TaggableManager(blank=True, through=TaggedItemThrough,
                           related_name="apartment_tags")

# Example code
from example.demo_django_base.models import DemoProjectTag
dt = DemoProjectTag.objects.create(name='color tag 20', color='#ff0000')

from example.demo_django_base.models import Apartment
a = Apartment.objects.create(number=1)
a.tags.add(dt)
a.tags.all()

<QuerySet [<DemoProjectTag: color tag 20>]>

# Get background svg for tags
DemoProjectTag.get_background_svg_for_tags(Apartment.objects.all().first().tags.all())

```

## Fields

### HEXColorField

Field with validator for color in hex format, currently used for setting background color for Tags.

## Middleware

### \*Project Middleware\*

ProjectMiddleware: If you want to set current project which is selected to request object you can use ProjectMiddleware which should be placed to start of MIDDLEWARE list in settings.py. Middleware sets DJANGO\_PROJECT\_BASE\_BASE\_REQUEST\_URL\_VARIABLES setting dict values to request object. Default value for DJANGO\_PROJECT\_BASE\_BASE\_REQUEST\_URL\_VARIABLES setting is {'project': 'current\_project\_slug', 'language': 'current\_language'}.

This means request will have current\_project\_slug attribute which will have value set to current project slug and request will have current\_language attribute which will have value set to current language set. If project or language cannot be determined its value is set to None.

To set current project to ajax requests 'Current-Project' header should be used: 'Current-Project': 'current project slug'. Current slug can also be determined from request path. See DJANGO\_PROJECT\_BASE\_PROJECT\_DEFINED\_URL\_PART setting description in setting section.

```

# myproject/settings.py

MIDDLEWARE = [
    'django_project_base.base.UrlVarsMiddleware',
    ...
]

```

### \*Performance profiler\*

Performance profiler module is providing functionality to log and display the summary of the most time-consuming requests.

To enable middleware add following line to project files:

```

# myproject/settings.py

MIDDLEWARE = [
    ...
]

```

```

'django_project_base.profiling.profile_middleware',
...
]

# myproject/urls.py
from django_project_base.profiling import app_debug_view

urlpatterns = [
    path('app-debug/', app_debug_view, name='app-debug'),
    ...
]

```

Overview of current state is available on url [http://hostname/app\\_debug/](http://hostname/app_debug/)

## Modules

The page contains all information about Django Project Base modules:

### Project

Project API is core part of Django project base.

#### Project slug

##### DJANGO\_PROJECT\_BASE\_SLUG\_FIELD\_NAME

When creating models with slug field they should be named with this setting value. This enables that we can use object slug instead of object pk when making api requests. Default value is “slug”.

#### DJANGO\_PROJECT\_BASE\_BASE\_REQUEST\_URL\_VARIABLES

```

DJANGO_PROJECT_BASE_BASE_REQUEST_URL_VARIABLES: {
    'project': {'value_name': 'current_project_slug', 'url_part': 'project-'},
    'language': {'value_name': 'current_language', 'url_part': 'language-'}
}

```

This setting defines a dictionary of attribute names on the request object. E.g. project info is set on the request object under property `current_project_slug`. Language information is set on request objects under property `current_language`. If language or project is given in request path like `language-EN`, then `url_part` settings is found and `EN` string is taken as language value.

### Profile

Account / profile API.

#### Profile reverse name order

Settings option **PROFILE\_REVERSE\_FULL\_NAME\_ORDER** defines `first_name`, `last_name` order for readonly field `full_name`. Default order is *False* - “First Last”. Changing setting to *true* will reverse order to “Last First”.

Global setting can be also overridden with profile option `reverse_full_name_order` (bool).

#### Deleting profile

Super admins can either delete profile or mark it for deletion in future.

User cannot delete its profile, he can only mark it for deletion in future. After confirmation for deletion, his profile is marked for deletion, user is logged out and is not able to log in or use features that require logged in user.

Settings value **DELETE\_PROFILE\_TIMEDELTA** defines how far in future will user profile be actually deleted with automatic process. Value is set in days.

## Notifications

### What is notifications module?

Notifications module will provide functionality to create and deliver notifications to users via channels like: email, websocket, push notification,.. Currently only maintenance notifications are implemented.

### Maintenance notifications

#### Description

When we have a planned server downtime to upgrade or some such, we need to somehow notify the users. But before maintenance occurs, the app itself must also notify the users that server will soon be down for maintenance. This notifications is presented to users 8 hours before planned downtime, 1 hour before planned downtime, 5 minutes before server is going offline.

In order to achieve that we can create a maintenance notification via REST api described in [Swagger UI](#). If we have django project base titlebar UI component integrated into our web UI this component will display notifications for planned maintenance in above described intervals.

#### Installation

Add app to your installed apps.

```
# myproject/settings.py

INSTALLED_APPS = [
    ...
    'django_project_base.notifications',
]
```

Add django-project-base notifications urls:

```
# url.py

urlpatterns = [
    ...
    path('', include(notifications_router.urls)),
    ...
]
```

Run migrations:

```
python manage.py migrate
```

## Authentication

### Rest\_registration

Currently, for basic authentication operations, rest\_registration module is used. It is overridden with custom rest actions used to override api documentation, but it just redirects requests back to rest\_registration.

Use rest\_registration documentation <https://django-rest-registration.readthedocs.io/en/latest/index.html> for details.

#### \*Overridden rest\_registration\*

If you want to use overridden rest\_registration views, replace rest\_registration urls with:

```
# myproject/urls.py
from django_project_base.account import accounts_router

urlpatterns = [
    path('account/', include(accounts_router.urls)),
    ...
]
```



## Impersonate user

Sometimes is useful if we can login into app as another user for debugging or help purposes. User change is supported via REST api calls or you can use userProfile component (django\_project\_base/templates/user-profile/bootstrap/template.html) which already integrates api functionality. Functionality is based on django-hijack package.

For determining which user can impersonate which user you can set your own logic. Example below:

```
# settings.py
HIJACK_AUTHORIZATION_CHECK = 'app.utils.authorization_check'

# app.utils.py
def authorization_check(hijacker, hijacked):
    """
    Checks if a user is authorized to hijack another user
    """
    if my_condition:
        return True
    else:
        return False
```

## User caching backend

To increase AUTH performance you can set a backend that caches users.

To enable User caching backend to add the following line to *AUTHENTICATION\_BACKENDS* section in settings.py:

```
# myproject/settings.py

AUTHENTICATION_BACKENDS = (
    ...
    'django_project_base.base.auth_backends.UsersCachingBackend',
    ...
)
```

User caching is not enabled for bulk updates by default, since Django doesn't call signal on .update() .bulk\_update() or .delete(). Updating data with a query or running bulk update, without clearing cache for every object could potentially cause race conditions. Avoid it if possible, or take care of manually clearing the cache for the user.

Example for clearing cache after bulk update:

```
...
from django.core.cache import cache
from django_project_base.settings import DJANGO_USER_CACHE
...
# Bulk update multiple users. Give them superuser permission.
# If those users are logged in, they don't have permission until cache is
# cleared or they log out and log in again.
UserProfile.objects.filter(username__in=['miha', 'janez']).update(
    is_superuser=True, is_staff=True)

# After clearing users cache for those users will be able
# to work with additional permissions
staff = UserProfile.objects.filter(username__in=['miha', 'janez'])
for user in staff:
    cache.delete(DJANGO_USER_CACHE % user.id)
```

It is possible to add a clear cache option also for bulk updates if needed with a custom QuerySet manager. You can find example code below.

```
# models.py
...
from django.core.cache import cache
from django_project_base.settings import DJANGO_USER_CACHE
...
class ProfilesQuerySet(models.QuerySet):
```

```

def update(self, **kwargs):
    for profile in self:
        cache.delete(DJANGO_USER_CACHE % profile.id)
    res = super(ProfilesQuerySet, self).update(**kwargs)
    return res

def delete(self):
    for profile in self:
        cache.delete(DJANGO_USER_CACHE % profile.id)
    res = super(ProfilesQuerySet, self).delete()
    return res

class UserProfile(BaseProfile):
    """Use this only for enabling cache clear for bulk update"""
    objects = ProfilesQuerySet.as_manager()
    ...

```

## Social auth integrations

Django Project Base offers easy-to-setup social authentication mechanism. Currently the following providers are supported:

- **Facebook**
  - provider identifier: facebook
- **Google**
  - provider identifier: google-oauth2
- **Twitter**
  - provider identifier: twitter
- **Microsoft**
  - provider identifier: microsoft-graph
- **Github**
  - provider identifier: github
- **Gitlab**
  - provider identifier: gitlab

OAuth providers require redirect URL which is called after the authentication process in OAuth flow.

Your redirect url is: [SCHEME]://[HOST]/account/social/complete/[PROVIDER IDENTIFIER]/

Information which settings are required for a social provider can be found at <https://python-social-auth.readthedocs.io/en/latest/backends/index.html>

For social authentication functionalities [Python Social Auth](#) library was used. Please checkout this documentation to make any custom changes.

### Installation

Add app to your installed apps.

```

# myproject/settings.py

from django_project_base.accounts import ACCOUNT_APP_ID

INSTALLED_APPS = [
    ...
    'social_django',
    ACCOUNT_APP_ID,

```

```
...
]
```

Make sure you have django project base urls included:

```
# url.py

urlpatterns = [
    ....
    path('account/', include(accounts_router.urls)),
    path('account/social/', include('social_django.urls', namespace="social")),
    ....
]
```

Run migrations:

```
python manage.py migrate
```

### Social login integration example - Google

To enable a social provider create an account at provider webpage and create an oauth app. For example for Google OAuth login visit <https://console.developers.google.com/apis/credentials>. Click + CREATE CREDENTIALS and select OAuth Client ID. Then create OAuth app with OAuth Consent screen.

Example value for Authorized JavaScript origins can be <http://localhost:8080>.

Example value for Authorized redirect URIs can be <http://localhost:8080/account/social/complete/google-oauth2/>.

To enable Google OAuth login add folowing to settings:

```
# myproject/settings.py
# enable google social login
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = '*Client ID*'
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = '*Client secret*'
```

## Translations

Currently translations in JS code, are done with Vue custom translations method.

It should be trivial to enable Django javascript-catalog, but it doesn't work correctly at the moment. It might change to correct Django javascript-catalog in future.

## Examples

### Titlebar component integration example

```
# define view function, put it in one of urls definition in urls.py
from django.shortcuts import render

def index_view(request):
    return render(request=request, template_name='template.html')
```

```
<!-- prepare html template template.html -->

{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Titlebar component example</title>
  {# include django javascript catalog for internationalization #}
  <script src="{% url 'javascript-catalog' %}"></script>
  {# add bootstrap library with dependencies and font-awesome #}
  <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/all.min.css"
```

```

    rel="stylesheet" crossorigin="anonymous">
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.js"
    crossorigin="anonymous">
</script>
<link
    href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.1.1/css/bootstrap.css"
    rel="stylesheet" crossorigin="anonymous">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
    crossorigin="anonymous">
</script>
{# include django project base js lib and appropriate css #}
<link href="{% static 'bootstrap_template.css' %}" rel="stylesheet"
    crossorigin="anonymous">
<script src="{% static 'django-project-base.min.js' %}"></script>
</head>
<body>
    {# set div which will contain titlebar component #}
    <div id="titlebar-app" class="titlebar-app">
        {# use/render titlebar component #}
        <titlebar></titlebar>
    </div>
    {# include vue inline template for titlebar component from folder
        corresponding to included css file #}
    {% include "bootstrap/titlebar.html" %}
    <script>
        // initialize titlebar component
        createApp('titlebar-app', titlebar);
    </script>
</body>
</html>

```

## Example project

You can find examples of most of the functionality of Django project base project in `/example/` folder.

## Run example project

Run Python runserver from root directory of this project and visit url that is provided in command output.

```

$python manage.py runserver

...
Django version 3.1.8, using settings 'example.setup.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
...

```

## Serve Sphinx documentation on localhost

Include documentation url to project urls.

```

# url.py

urlpatterns = [
    ....
    re_path(r'^docs-files/(?P<path>.*)$', documentation_view, {'document_root': DOCUMENTATION_DIRECTORY},
        name='docs-files'),
    ....
] + _static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)

```

## Sample data

### Users

- miha:
  - username: miha

- password: mihamiha
- janez:
  - username: janez
  - password: janezjanez

## Swagger

### Installation

To enable swagger gui, add following to urls.py

```
# my_project/urls.py
urlpatterns = [
    ...
    path('schema/', SpectacularAPIView.as_view(), name='schema'),
    path('schema/swagger-ui/', SpectacularSwaggerView.as_view(url_name='schema', ),
         name='swagger-ui'),
    ...
]
```

Swagger UI is now accessible on /schema/swagger-ui/ url by running example project.

### Open Api

Add following to settings.py

```
# myapp/settings.py
REST_FRAMEWORK = {
    ...
    'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
    ...
}
```