django-email-foundation Documentation

Release master

Francesc Arpí Roca

May 20, 2020

Contents

CHAPTER 1

Description

This is a Django package that helps you build email templates for your email engine sender (we recommend you use django-yubin). It uses the zurb foundation for emails templates and *node* packages such as *inky* or *panini*.

It provides you with some commands and functionality to integrate *zurb foundation for emails* in your django project.

- *install_requires*: A command to install the required node packages, such as *inky*, *panini*, *gulp*, etc in your project.
- *create_basic_structure*: It creates an essential tree structure in your project, that contains the basic layout and folders such as *pages*, *helpers* and *partials*, used by panini.
- *email_builder*: Starts a *gulp* process that watches your source templates, builds them and finally copies them to your target email folder. It compiles the sources using panini and inky for the best compatibility with the major email's client.

It also gives you a django view to preview the generated templates. For the preview, you can use a custom fixed context for each template, and this is very useful because it allows designers to edit the layouts.

CHAPTER 2

Index

2.1 Installation

You can find the package on PyPI and it can then be installed using *pip*:

```
pip install django-email-foundation
```

You can also download the .zip distribution file and unpack it or download the sources. Inside this zip, you can find a script file named `setup.py`. Enter this command:

```
python setup.py install
```

. . . and the package will be installed automatically.

2.1.1 Configuration

This package has been tested on:

- Python: 3.5.9, 3.6.7
- Django: 1.9.x, 2.1.7
- npm: 5.8.0
- yarn: 1.13.0
- node: 8.11.4, 10.20.1

In your Django project's settings, add the package to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'django_email_foundation',
)
```

It is also necessary to add the *def* urls in your project. Edit your main *urls.py* and add:

```
urlpatterns = [
    ...
    path('def/', include('django_email_foundation.urls')),
]
```

Below you can see a list of all available settings which can be added to your Django settings configuration. Notice that these constants start with *DEF* (Django Email Foundation).

2.1.2 Required settings

These settings are required and necessary to use any of the *def* commands.

DEF_TEMPLATES_SOURCE_PATH

It refers to the relative path from your root project where your email sources templates are located. For example, if you have the following folder's tree:

```
my_project
├── readme.md
└── src
    ├── emails_app
    │   ├── models.py
    │   ├── templates
    │   │   └── emails_app
    │   ├── templates_sources
    │   │   ├── assets
    │   │   ├── helpers
    │   │   ├── layouts
    │   │   ├── pages
    │   │   └── partials
    │   └── views.py
    └── manage.py
```

Them the constant should be:

```
DEF_TEMPLATES_SOURCE_PATH = 'src/emails_app/templates_sources'
```

Note: Important! The paths must be relative from the root project

DEF_TEMPLATES_TARGET_PATH

It refers to the path where the compiled email templates are stored. For example, from the previous example:

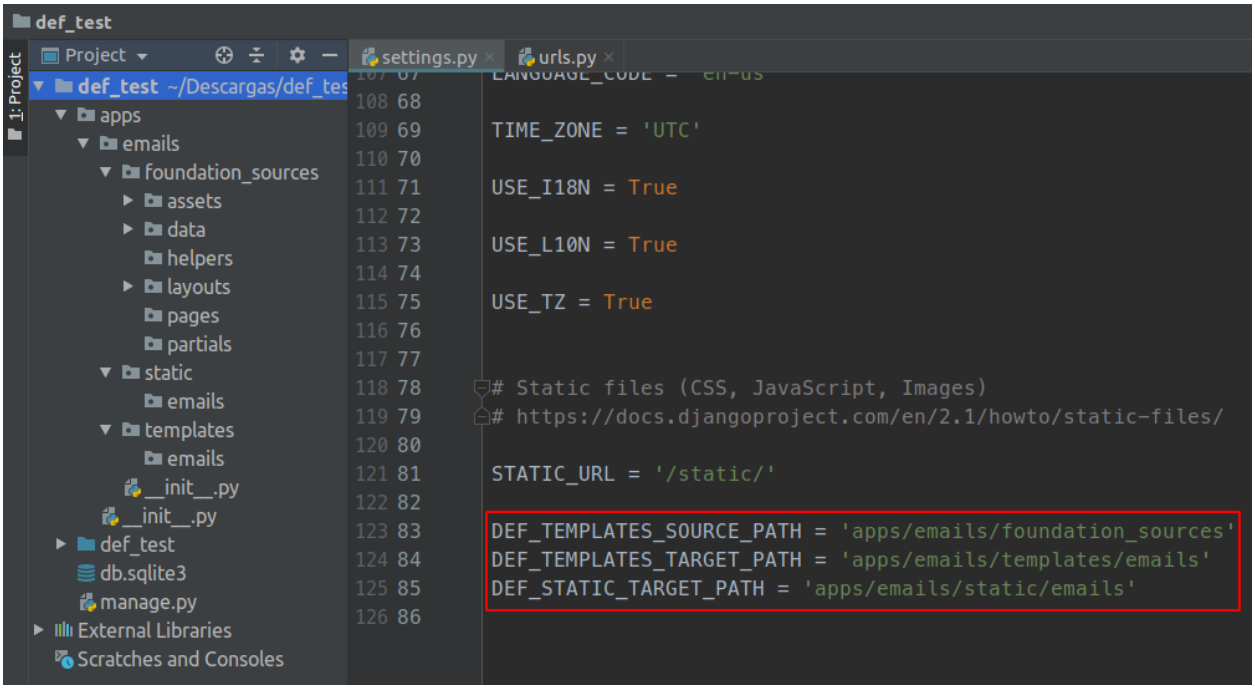
```
DEF_TEMPLATES_TARGET_PATH = 'src/emails_app/templates/emails_app'
```

DEF_STATIC_TARGET_PATH

Necessary for set where store the static files (images) in to the target path. Example:

```
DEF_STATIC_TARGET_PATH = 'src/emails_app/static/emails_app'
```

Take a look on this example with the three required settings:



2.1.3 Optional settings

The optional settings can be used when you want to override the default values.

DEF_NPM_OR_YARN

It allows you to set which node package's system will be used for installing the dependencies. Default optoin is **yarn** but you can replace with **npm**.

DEF_NODE_MODULES_PATH

The path where the node packages will be installed. The *node_modules* folder, by default will be created at the project root folder. Do not include *node_modules* in this setting. For example:

```
DEF_NODE_MODULES_PATH = '/home/my-user/workspace/my-project'
```

DEF_IGNORE_FILES

A list (or tuple) of files that will not be built with *panini* when the *email_builder* command is running. However they will be moved at the target folder path.

By default there are two files, *subject.html* and *body.txt*.

For example you could have the following scenario:

```
templates_sources
├── assets
├── helpers
├── layouts
├── pages
│   └── user_account_validation
│       ├── body.html
│       ├── body.txt
│       └── subject.html
└── partials
```

You may only want to compile the *body.html* file but not the other two. Although you want to move it to the destination folder.

DEF_RUNSERVER_HOST

By default *http://localhost:8000*. Change it if your project runs on another host or port.

2.2 Commands

Three important commands are included with this application.

2.2.1 email_builder

This command launches a *gulp* process and watches your source files so that they are re-compiled, using panini + inky.

How to execute it:

```
./manage.py email_builder
```

If you run this command before your project has been configured, you will see a message similar to this one:

```
→ ./manage.py email_builder
Oops! Something went wrong...
  - Some of the required modules are not installed in "node_modules". Please run "./
↪manage.py install_requires"
  - It is necessary to define DEF_TEMPLATES_SOURCE_PATH in your settings
  - The templates directory must have a valid structure. It must contain the pages,␣
↪layouts, partials and helpers folders. You can run ".manage.py create_basic_
↪structure" to build this structure, and to add a basic layout.
  - It is necessary to define DEF_TEMPLATES_TARGET_PATH in your settings
```

This is due to the command performing some checks before it runs. For example, it verifies that you already have the required node packages, that the required constants have been defined in your settings, etc.

If everything is OK, you'll see something like:

```
→ ./manage.py email_builder
Oh, yes! Punchi, punchi! Lets go!
[16:29:04] Using gulpfile ~                                    /gulpfile.js
[16:29:04] Starting 'watch'...
[16:29:04] Starting 'build'...
[16:29:04] Finished 'build' after 124 ms
[16:29:04] Starting 'preview'...
[16:29:04] Finished 'preview' after 1.19 ms
[16:29:04] Starting 'watch'...
[16:29:04] Opening http://localhost:8000/def/ using the default OS app
```

2.2.2 install_requires

This command uses *npm* or *yarn* (depending on your configuration) to install the required node packages, such as *gulp*, *panini* or *inky*. It also creates the *gulpfile.js* file in your root path to allow you to use the *email_builder* command.

How to execute it:

```
./manage.py install_requires
```

Note: This command will create the *node_modules* folder, and it will also add some files to your root path: *gulpfile.js*, *yarn.lock*, *package.lock* and *package.json*. Remember to add these entries to your *.gitignore* to avoid committing these files.

2.2.3 create_basic_structure

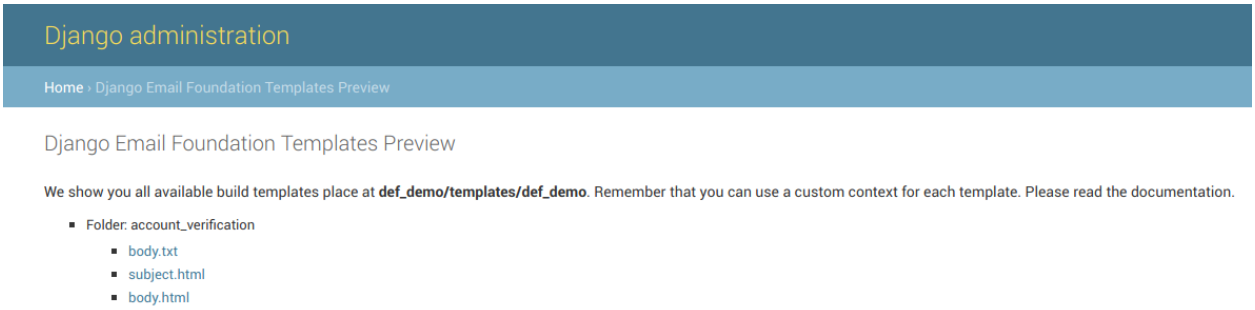
This command creates a basic tree structure in your templates source path. This structure will look like this:

```
templates_sources
├── assets
│   └── scss
│       ├── app.scss
│       ├── _settings.scss
│       └── template
│           └── _template.scss
├── helpers
├── layouts
│   └── default.html
├── pages
├── data
│   └── context.py
└── partials
```

Once you have built this structure, you can start creating your custom templates inside the *pages* folder. Take a look at the official documentation. We recommend that you use the inky template language as it will make your life much easier ;).

2.3 Previewing build templates

If everything is ok and the *email_builder* command is up and running, the following view will open:



Note: For the previous screenshot, we have the following source and target templates.

Sources:

```
foundation_templates
├── assets
│   └── scss
│       ├── app.scss
│       ├── _settings.scss
│       └── template
│           └── _template.scss
├── helpers
├── layouts
│   └── default.html
├── pages
│   └── account_verification
│       ├── body.html
│       ├── body.txt
│       └── subject.html
└── partials
```

Target:

```
templates
└── emails
    └── account_verification
        ├── body.html
        ├── body.txt
        └── subject.html
```

The preview view contains a list of all build templates. You can click on each one and to see the template rendered using your custom context.

2.4 Custom context

Another important functionality of this package, is to use a custom context to preview your templates.

The context file it's stored inside the *data* folder, where are the source templates. It's a python file, named *context.py* which contain a dictionary, also named *context*.

Note: The python dictionary it's more powerful that a json file, for example. It allows you to define date objects, reuse another attributes, etc.

The dictionary must contain two leveled keys. The first level it's for the folder name, and the child, for file name.

For example:

```
{
    "example_folder": {
        "body.html": {
            "name": "Demo Name"
        }
    }
}
```

Now, if in your template, placed in *example_folder/body.html*, contains:

```
Hello \{{ name }}!
```

Note: Notice that the brackets need to be escaped because we are using *inky* and it conflicts with *jinja2*. We must translate our *jinja2* tags to the built-in django template.

You will see the following result in the template preview view:

```
Hello Demo Name!
```

This is very useful for your designers to work on the template directly.

2.5 Contributing

django-email-foundation is an open source project and improvements and bug reports are very appreciated.

You can contribute in many ways:

- Filling a bug on github
- Creating a patch and sending the pull request
- Help on testing and documenting

When sending a pull request, please be sure that all tests and builds passes. On the next section you'll find information about how to write the test.

Please follow the PEP8 coventions and in case you write additional features don't forget to write the tests for them.

2.5.1 Running tests

You tu install those python packages in your virtualenv:

```
pip install pytest pytest-flake8
```

And then run:

```
pytest
```