

---

# **django-sage-painless**

*Release version 0.0.1*

**SageTeam**

**Jul 23, 2021**



# CONTENTS

<b>1</b>	<b>Functionality</b>	<b>3</b>
<b>2</b>	<b>Documentation</b>	<b>5</b>
2.1	Quick Start . . . . .	5
2.1.1	Getting Started . . . . .	5
2.1.2	Start Project . . . . .	5
2.1.3	Install Generator . . . . .	6
2.2	Usage . . . . .	6
2.3	Diagram . . . . .	9
2.3.1	Template . . . . .	9
2.3.2	Examples . . . . .	12
2.4	Contribute . . . . .	16
2.4.1	Project Detail . . . . .	16
2.4.2	Git Rules . . . . .	16
2.4.3	Development . . . . .	16
2.5	FAQ . . . . .	17
2.5.1	What is code generator? . . . . .	17
2.5.2	What is django-sage-painless? . . . . .	17
2.5.3	Why should we use this package? . . . . .	17
2.5.4	What are the main features of the package? . . . . .	17
2.5.5	Why don't we produce the whole Django project? . . . . .	18
2.5.6	How to learn to create a diagram? . . . . .	18
2.5.7	How does the cache algorithm work? . . . . .	18
<b>3</b>	<b>Issues</b>	<b>19</b>
<b>4</b>	<b>Indices and tables</b>	<b>21</b>





This app supports the following combinations of Django and Python:

Django	Python
3.1	3.7, 3.8, 3.9
3.2	3.7, 3.8, 3.9



## FUNCTIONALITY

painless creates django backend projects without developer coding

it can generate these parts:

- models.py
- admin.py
- serializers.py
- views.py
- urls.py
- tests
- api documentation
- Dockerfile
- docker-compose.yml

it also can config pro stuff in django:

- Redis cache
- RabbitMQ





## DOCUMENTATION

## 2.1 Quick Start

### 2.1.1 Getting Started

Before creating django project you must first create virtualenv.

```
$ python3.9 -m pip install virtualenv
$ python3.9 -m virtualenv venv
```

To activate virtual environment in ubuntu:

```
$ source venv/bin/activate
```

To deactivate virtual environment use:

```
$ deactivate
```

### 2.1.2 Start Project

First create a Django project

```
$ mkdir GeneratorTutorials
$ cd GeneratorTutorials
$ django-admin startproject kernel .
```

Next we have to create a sample app that we want to generate code for it (it is required for development. you will run tests on this app)

```
$ python manage.py startapp products
```

Now we have to add 'products' to INSTALLED\_APPS in settings.py

```
INSTALLED_APPS = [
    ...
    'products',
    ...
]
```

### 2.1.3 Install Generator

First install package

```
$ pip install django-sage-painless
```

Then add 'sage\_painless' to INSTALLED\_APPS in settings.py

These apps should be in your INSTALLED\_APPS:

- 'rest\_framework'
- 'drf\_yasg'
- 'django\_seed'

```
INSTALLED_APPS = [  
    ...  
    'sage_painless',  
    ...  
    'rest_framework',  
    'drf_yasg',  
    'django_seed',  
    ...  
]
```

## 2.2 Usage

For generating a whole project you just need a diagram. diagram is a json file that contains information about database tables.

[you can find examples of diagram file here](#)

start to generate (it is required for development. you will run tests on this app)

[NEW]: First validate the format of your diagram, It will raise errors if diagram format was incorrect.

```
$ python manage.py validate_diagram --diagram <path to diagram>
```

Now you can generate code

```
$ python manage.py generate --diagram <path to diagram>
```

Here system will ask you what you want to generate for your app. Questions:

Question	Description
Would you like to generate models.py(yes/no)?	generates models.py from json diagram for your app
Would you like to generate admin.py(yes/no)?	generates admin.py from admin settings in json diagram for your project
Would you like to generate serializers.py & views.py(yes/no)?	generates serializers.py and views.py in api directory for your project
Would you like to generate test for your project(yes/no)?	generates model test and api test for your project in tests directory
Would you like to add cache queryset support(yes/no)?	it will cache queryset via redis in your views.py
Would you like to dockerize your project(yes/no)?	creates Dockerfile and docker-compose.yml for your project
Would you like to generate docs(yes/no)?	generates README.md and saves in docs/sage_painless/git/

If you chose to dockerize project:

Question	Description
Please enter the version of your project(e.g 2.1):	the version of your project in number
Please enter your project's database image(e.g postgres):	it can be one of docker images recommended <i>postgres</i>
Please enter database name:	your database name that set in django settings
Please enter database user username:	the database will create by this user
Please enter database user password:	password of user
Would you like to config redis server for your project(yes/no)?	if you want cache support <i>yes</i> else <i>no</i>
Would you like to config rabbitMQ for your project(yes/no)?	configs RabbitMQ in your docker environment

If you chose to config RabbitMQ:

Question	Description
Please enter rabbitMQ user username:	RabbitMQ user username
Please enter rabbitMQ user password:	RabbitMQ user password

If you generated api you have to add app urls to urls.py:

```
urlpatterns = [
    ...
    path('api/', include('products.api.urls')),
    ...
]
```

If you set cache support add CACHES to your settings:

```
REDIS_URL = 'redis://localhost:6379/'
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": os.environ['REDIS_URL'] if os.environ.get('REDIS_URL') else settings.
        REDIS_URL if hasattr(settings, 'REDIS_URL') else 'redis://localhost:6379/'
    }
}
```

If you have encrypted field in diagram:

- your database should be PostgreSQL
- you should install *pgcrypto* extension for PostgreSQL with this command

```
$ sudo -u postgres psql <db_name>
$ CREATE EXTENSION pgcrypto;
```

- You have to migrate your new models

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

- You can run tests for your app

```
$ python manage.py test products
```

- Django run server

```
$ python manage.py runserver
```

- For support Rest API doc add this part to your `urls.py`

```
from rest_framework.permissions import AllowAny
from drf_yasg.views import get_schema_view
from drf_yasg import openapi

schema_view = get_schema_view(
    openapi.Info(
        title="Rest API Doc",
        default_version='v1',
        description="Auto Generated API Docs",
        license=openapi.License(name="S.A.G.E License"),
    ),
    public=True,
    permission_classes=(AllowAny, ),
)

urlpatterns = [
    ...
    path('api/doc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-swagger-
↪ui'),
    ...
]
```

- Rest API documentation is available at `localhost:8000/api/doc/`

## 2.3 Diagram

### 2.3.1 Template

Diagram is a json file that contains database tables, settings for admin panel and API configs It is the only thing you need to generate a whole project

[NEW]: You can also use *encryption* capability in diagram. Example:

```
"title": {
  "type": "character",
  "max_length": 255,
  "unique": true,
  "encrypt": true
}
```

the template of the diagram is something like this:

```
{
  "apps": {
    "ecommerce": {
      "models": {
        "Category": {
          "fields": {
            "title": {
              "type": "character",
              "max_length": 255,
              "unique": true
            },
            "created": {
              "type": "datetime",
              "auto_now_add": true
            },
            "modified": {
              "type": "datetime",
              "auto_now": true
            }
          }
        },
        "admin": {
          "list_display": ["title", "created", "modified"],
          "list_filter": ["created", "modified"],
          "search_fields": ["title"]
        },
        "api": {
          "methods": ["GET", "POST", "PUT", "PATCH", "DELETE"]
        }
      },
      "Product": {
        "fields": {
          "title": {
            "type": "character",
            "max_length": 255
          }
        }
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

        "description": {
            "type": "character",
            "max_length": 255
        },
        "price": {
            "type": "integer"
        },
        "category": {
            "type": "fk",
            "to": "Category",
            "related_name": "'products'",
            "on_delete": "CASCADE"
        },
        "created": {
            "type": "datetime",
            "auto_now_add": true
        },
        "modified": {
            "type": "datetime",
            "auto_now": true
        }
    },
    "admin": {
        "list_display": ["title", "price", "category"],
        "list_filter": ["created", "modified"],
        "search_fields": ["title", "description"],
        "raw_id_fields": ["category"]
    }
},
"Discount": {
    "fields": {
        "product": {
            "type": "fk",
            "to": "Product",
            "related_name": "'discounts'",
            "on_delete": "CASCADE"
        },
        "discount": {
            "type": "integer"
        },
        "created": {
            "type": "datetime",
            "auto_now_add": true
        },
        "modified": {
            "type": "datetime",
            "auto_now": true
        }
    },
    "admin": {
        "list_display": ["discount", "product", "created", "modified"],
        "list_filter": ["created", "modified"],

```

(continues on next page)

(continued from previous page)

```

        "raw_id_fields": ["product"]
    }
}
}
}
}

```

}

field types are:

Type	Django
character	CharField
integer	IntegerField
float	FloatField
datetime	DateTimeField
date	.DateField
text	TextField
fk	ForeignKey
one2one	OneToOneField
m2m	ManyToManyField
image	ImageField
file	FileField
bool	BooleanField
slug	SlugField

in admin you can set:

Option	Input
list_display	list of strings
list_filter	list of strings
search_fields	list of strings
filter_horizontal	list of strings
filter_vertical	list of strings
raw_id_fields	list of strings
has_add_permission	boolean
has_change_permission	boolean
has_delete_permission	boolean

in api you can set:

Option	Input
methods	list of strings

## 2.3.2 Examples

example 1:

2 apps (ecommerce & discount)

```
{
  "apps": {
    "ecommerce": {
      "models": {
        "Category": {
          "fields": {
            "title": {
              "type": "character",
              "max_length": 255,
              "unique": true
            },
            "created": {
              "type": "datetime",
              "auto_now_add": true
            },
            "modified": {
              "type": "datetime",
              "auto_now": true
            }
          },
          "admin": {
            "list_display": [
              "title",
              "created",
              "modified"
            ],
            "list_filter": [
              "created",
              "modified"
            ],
            "search_fields": [
              "title"
            ]
          },
          "api": {
            "methods": [
              "GET",
              "POST",
              "PUT",
              "PATCH",
              "DELETE"
            ]
          }
        },
        "Product": {
          "fields": {
            "title": {
              "type": "character",
```

(continues on next page)



(continued from previous page)

```

        "max_length": 255
    },
    "description": {
        "type": "character",
        "max_length": 255
    },
    "price": {
        "type": "integer"
    },
    "category": {
        "type": "fk",
        "to": "Category",
        "related_name": "'products'",
        "on_delete": "CASCADE"
    },
    "created": {
        "type": "datetime",
        "auto_now_add": true
    },
    "modified": {
        "type": "datetime",
        "auto_now": true
    }
},
"admin": {
    "list_display": [
        "title",
        "price",
        "category"
    ],
    "list_filter": [
        "created",
        "modified"
    ],
    "search_fields": [
        "title",
        "description"
    ],
    "raw_id_fields": [
        "category"
    ]
}
}
},
"discount": {
    "models": {
        "Discount": {
            "fields": {
                "product": {
                    "type": "fk",
                    "to": "Product",

```

(continues on next page)

(continued from previous page)

```
        "related_name": "'discounts'",
        "on_delete": "CASCADE"
    },
    "discount": {
        "type": "integer"
    },
    "created": {
        "type": "datetime",
        "auto_now_add": true
    },
    "modified": {
        "type": "datetime",
        "auto_now": true
    }
},
"admin": {
    "list_display": [
        "discount",
        "product",
        "created",
        "modified"
    ],
    "list_filter": [
        "created",
        "modified"
    ],
    "raw_id_fields": [
        "product"
    ]
}
}
```

example 2:

1 app (articles)

```
{
  "apps": {
    "articles": {
      "models": {
        "Article": {
          "fields": {
            "title": {
              "type": "character",
              "max_length": 120
            },
            "body": {
              "type": "character",
              "max_length": 255
            }
          }
        }
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

    },
    "slug": {
        "type": "slug",
        "max_length": 255,
        "unique": true
    },
    "created": {
        "type": "datetime",
        "auto_now_add": true
    },
    "publish": {
        "type": "datetime",
        "null": true,
        "blank": true
    },
    "updated": {
        "type": "datetime",
        "auto_now": true
    },
    "options": {
        "type": "character",
        "max_length": 2,
        "choices": [
            [
                "dr",
                "Draft"
            ],
            [
                "pb",
                "public"
            ],
            [
                "sn",
                "soon"
            ]
        ]
    }
},
"admin": {
    "list_display": [
        "title",
        "created",
        "updated"
    ],
    "list_filter": [
        "created",
        "updated",
        "options"
    ],
    "search_fields": [
        "title",
        "body"
    ]
}

```

(continues on next page)

(continued from previous page)

```

    ],
    "api": {
      "methods": [
        "get",
        "post"
      ]
    }
  }
}

```

## 2.4 Contribute

### 2.4.1 Project Detail

You can find all technologies we used in our project into these files:

- Version: 1.0.0
- Frameworks: Django 3.2.4
- **Libraries:**
  - Django rest framework 3.12.4
  - Jinja2 3.0.1
- Language: Python 3.9.4

## 2.4.2 Git Rules

Sage team Git Rules Policy is available here:

- Sage Git Policy

### 2.4.3 Development

Run project tests before starting to develop - products app is required for running tests

```
$ python manage.py startapp products
```

```
INSTALLED_APPS = [
    ...
    'products',
    ...
]
```

- you have to generate everything for this app

- diagram file is available here: [Diagram](#)

```
$ python manage.py generate --app products --diagram sage_painless/tests/diagrams/
↪product_diagram.json
```

- run tests

```
$ python manage.py test sage_painless
```

## 2.5 FAQ

### 2.5.1 What is code generator?

A code generator is a tool or resource that generates a particular sort of code or computer programming language. This has many specific meanings in the world of IT, many of them related to the sometimes complex processes of converting human programming syntax to the machine language that can be read by a computing system. One of the most common and conventional uses of the term “code generator” involves other resources or tools that help to turn out specific kinds of code. For example, some homemade or open source code generators can generate classes and methods for easier or more convenient computer programming. This type of resource might also be called a component generator.

### 2.5.2 What is django-sage-painless?

The django-sage-painless is a valuable package based on Django Web Framework & Django Rest Framework for high-level and rapid web development. The introduced package generates Django applications. After completing many projects, we concluded that any basic project and essential part is its database structure. You can give the database schema in this package and get some parts of the Django application, such as API, models, admin, signals, model cache, setting configuration, mixins, etc. All of these capabilities come with a unit test. So you no longer have to worry about the simple parts of Django, and now you can write your advanced services in Django. Django-sage-painless dramatically speeds up the initial development of the project in Django. However, we intend to make it possible to use it in projects that are in progress. But the reality now is that we have made Django a quick start. We used the name painless instead of the Django code generator because this package allows you to reach your goals with less effort.

### 2.5.3 Why should we use this package?

One of the most important reasons to use this package is to speed up the development of Django applications. Then, another important reason is that you can use many features with this package if you want. Therefore, you DO NOT have to use all the features of the generator.

### 2.5.4 What are the main features of the package?

- Generate models based on your defined diagram
- Support database relationships: [one-to-one] [one-to-many] [many-to-many]
- Generate cache mixin to your models (OPTIONAL)
- Generate model test
- Generate signals (if you use one-to-one relationship)
- Generate rest framework API endpoints (OPTIONAL)

- Generate rest framework documentation (OPTIONAL)
- Generate API URLs (if request for API)
- Generate API test
- Generate admin via filter and search capability (OPTIONAL)
- Generate setting configuration of (Redis, RabbitMQ, Celery, etc. OPTIONAL)
- Generate docker compose file, Dockerfile and related documentation (OPTIONAL)

### **2.5.5 Why don't we produce the whole Django project?**

Based on this question, we took a new attitude was taken in the package. One of the important issues in package design is that it is scalable and compatible with projects that are under development. That's why we decided to automate only the apps according to the project design model instead of producing a complete Django project. Therefore, anyone can use this package in the middle of their startup development and release their new features faster than before.

### **2.5.6 How to learn to create a diagram?**

In the example section, we have taught all the sections related to Digram.

### **2.5.7 How does the cache algorithm work?**

Caching algorithm works in such a way that once your data is loaded, it is cached in Redis, and there is no need to query the database again. We have also designed the algorithm like that if your data in the database changes, cached data will be deleted automatically from Redis.

## ISSUES

If you have questions or have trouble using the app please file a bug report at:

<https://github.com/sageteam-org/django-sage-painless/issues>





## INDICES AND TABLES

- search