



Developer documentation

Release 5.8.0

VST Consulting

Sep 28, 2023

Contents

1	Quick Start	3
1.1	New application creation	3
1.2	Adding new models to application	6
2	Configuration manual	21
2.1	Introduction	21
2.2	Main settings	21
2.3	Databases settings	22
2.4	Cache settings	23
2.5	Locks settings	24
2.6	Session cache settings	24
2.7	Rpc settings	24
2.8	Worker settings	25
2.9	SMTP settings	25
2.10	Web settings	25
2.11	Centrifugo client settings	27
2.12	Storage settings	27
2.13	Throttle settings	28
2.14	Production web settings	29
2.15	Configuration options	29
3	Backend API manual	31
3.1	Models	31
3.2	Web API	38
3.3	Celery	66
3.4	Endpoint	67
3.5	Testing Framework	70
3.6	Utils	78
4	Frontend Quickstart	89
4.1	Field customization	91
4.2	Change path to FkField	92
4.3	CSS Styling	92
4.4	Show primary key column on list	93
4.5	View customization	93
4.6	Changing title of the view	94

4.7	Basic Webpack configuration	94
4.8	Page store	95
4.9	Overriding root component	95
4.10	Translating values of fields	95
4.11	Changing actions or sublinks	96
4.12	LocalSettings	96
4.13	Store	96
5	Frontend documentation	99
5.1	API Flowchart	99
5.2	Signals	100
5.3	List of signals in VST Utils	102
5.4	Field Format	105
5.5	Layout customization with CSS	107
	Python Module Index	109
	Index	111

VST Utils is a small framework for quick generation of a single-page applications. The main feature of the VST Utils framework is autogenerated GUI, which is formed based on the OpenAPI schema. OpenAPI schema is a JSON, that contains description of the models used in the REST API and info about all paths of the application.

In the documentation you can find info about QuickStart of new project based on VST Utils, description of base models, views and fields available in the framework, and also you will know how you can redefine some standard models, views and fields in your project.

Starting of new project, based on VST Utils Framework, is rather simple. We recommend to create a virtual environment for each project to avoid conflicts in the system.

Let's learn by example. All you need to do is run several commands. This manual consist of two parts:

1. Description of the process of creating a new application and the main commands for launching and deploying.
2. Description of the process of creating new entities in the application.

1.1 New application creation

Throughout this tutorial, we'll go through a creation of a basic poll application.

1. Install VST Utils

```
pip install vstutils
```

In this case, we install a package with the minimum required set of dependencies for creating new projects. However, inside the project, the extra argument *prod* is used, which additionally installs the packages necessary for working in the production environment. There is also a set of test dependencies, which contains everything you need for testing and analyzing code coverage.

It is also worth noting extra dependencies as:

- **rpc** - install dependencies for asynchronous tasks working
- **ldap** - a set of dependencies for ldap authorization support
- **doc** - everything which needed to build documentation and to implement the delivery of documentation inside a running server
- **pil** - library for correct work of image validators
- **boto3** - an additional set of packages for working with S3 storage outside of AWS
- **sqs** - a set of dependencies for connecting asynchronous tasks to SQS queues (can be used instead of the **rpc**).

You can combine multiple dependencies at the same time to assemble your set of functionality into a project. For example, to work an application with asynchronous tasks and media storage in MinIO, you will need the following command:

```
pip install vstutils[prod,rpc,boto3]
```

To install the most complete set of dependencies, you can use the common parameter **all**.

```
pip install vstutils[all]
```

2. Create new project, based on VST Utils

If this is your first time using vstutils, you'll have to take care of some initial setup. Namely, you'll need to auto-generate some code that establishes a vstutils application – a collection of settings for an instance of vstutils, including database configuration, Django-specific and vstutils-specific options and application-specific settings. To create new project execute following command:

```
python -m vstutils newproject --name {{app_name}}
```

This command will offer you to specify such options of new app, as:

- **project name** - name of your new application;
- **project guiname** - name of your new application, that will be used in GUI (web-interface);
- **project directory** - path to directory, where project will be created.

Or you can execute following command, that includes all needed data for new project creation.

```
python -m vstutils newproject --name {{app_name}} --dir {{app_dir}} --  
guiname {{app_guiname}} --noinput
```

This command creates new project without confirming any data.

These commands create several files in `project` directory.

```
/{app_dir}/{app_name}  
├── .coveragerc  
├── frontend_src  
│   ├── app  
│   │   └── index  
│   ├── .editorconfig  
│   ├── .eslintrc.js  
│   └── .prettierrc  
├── MANIFEST.in  
├── package.json  
├── .pep8  
├── README.rst  
├── requirements-test.txt  
├── requirements.txt  
├── setup.cfg  
├── setup.py  
├── {{app_name}}  
│   ├── __init__.py  
│   ├── __main__.py  
│   ├── models  
│   │   └── __init__.py  
│   ├── settings.ini  
│   ├── settings.py  
│   ├── web.ini  
│   └── wsgi.py  
├── test.py  
├── tox.ini  
└── webpack.config.jsdefault
```


where:

- **frontend_src** - directory that contains all sources for frontend;
- **MANIFEST.in** - this file is used for building installation package;
- **{{app_name}}** - directory with files of your application;
- **package.json** - this file contains list of frontend dependencies and commands to build;
- **README.rst** - default README file for your application (this file includes base commands for starting/stopping your application);
- **requirements-test.txt** - file with list of requirements for test environment;
- **requirements.txt** - file with list of requirements for your application;
- **setup.cfg** - this file is used for building installation package;
- **setup.py** - this file is used for building installation package;
- **test.py** - this file is used for tests creation;
- **tox.ini** - this file is used for tests execution;
- **webpack.config.js.default** - this file contain minimal script for webpack (replace '.default' if write smthg in 'app.js').

You should execute below commands from the `/{{app_dir}}/{{app_name}}/` directory. It is good practice to use `tox` (should be installed before use) to create a debugging environment for your application. For these purposes, it is recommended to use `tox -e contrib` in the project directory, which will automatically create a new environment with the required dependencies.

3. Apply migrations

Let's verify a newly created `vstutils` project does work. Change into the outer `/{{app_dir}}/{{app_name}}` directory, if you haven't already, and run the following command:

```
python -m {{app_name}} migrate
```

This command create SQLite (by default) database with default SQL-schema. `VSTUTILS` supports all databases [Django does](#)¹.

4. Create superuser

```
python -m {{app_name}} createsuperuser
```

5. Start your application

```
python -m {{app_name}} web
```

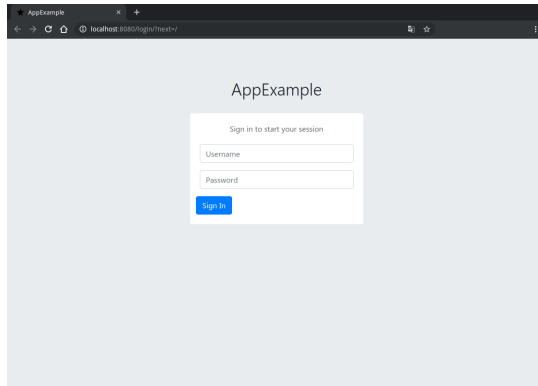
Web-interface of your application has been started on the port 8080. You've started the `vstutils` production server based on `uWSGI`².

Warning: Now's a good time to note: if you want to run the web-server with a debugger, then you should run [the standard Django's dev-server](#)³.

¹ <https://docs.djangoproject.com/en/4.1/ref/databases/#databases>

² <https://uwsgi-docs.readthedocs.io/>

³ <https://docs.djangoproject.com/en/4.1/intro/tutorial01/#the-development-server>



If you need to stop the server, use following command:

```
python -m {{app_name}} web stop=/tmp/{{app_name}}_web.pid
```

You've created the simplest application, based on VST Utils framework. This application only contains User Model. If you want to create your own models look at the section below.

1.2 Adding new models to application

If you want to add some new entities to your application, you need to do following on the back-end:

1. Create Model;
2. Create Serializer (optional);
3. Create View (optional);
4. Add created Model or View to the API;
5. Make migrations;
6. Apply migrations;
7. Restart your application.

Let's look how you can do it on the AppExample - application, that has 2 custom models:

- Task (abstraction for some tasks/activities, that user should do);
- Stage (abstraction for some stages, that user should do to complete the task. This model is nested into the Task Model).

1.2.1 Models creation

Firstly, you need to create file `{{model_name}}.py` in the `{{app_dir}}/{{app_name}}/{{app_name}}/models` directory.

Let make out an example from **BModel**:

```
class vstutils.models.BModel(*args, **kwargs)
```

Default model class that generates model viewset, separate serializers for list() and retrieve(), filters, api endpoints and nested views.

Examples:

```

from django.db import models
from rest_framework.fields import ChoiceField
from vstutils.models import BModel

class Stage(BModel):
    name = models.CharField(max_length=256)
    order = models.IntegerField(default=0)

    class Meta:
        default_related_name = "stage"
        ordering = ('order', 'id',)
        # fields which would be showed on list.
        _list_fields = [
            'id',
            'name',
        ]
        # fields which would be showed on detail view and creation.
        _detail_fields = [
            'id',
            'name',
            'order'
        ]
        # make order as choices from 0 to 9
        _override_detail_fields = {
            'order': ChoiceField((str(i) for i in range(10)))
        }

class Task(BModel):
    name = models.CharField(max_length=256)
    stages = models.ManyToManyField(Stage)
    _translate_model = 'Task'

    class Meta:
        # fields which would be showed.
        _list_fields = [
            'id',
            'name',
        ]
        # create nested views from models
        _nested = {
            'stage': {
                'allow_append': False,
                'model': Stage
            }
        }

```

In this case, you create models which could converted to simple view, where:

- POST/GET to `/api/version/task/` - creates new or get list of tasks
- PUT/PATCH/GET/DELETE to `/api/version/task/:id/` - updates, retrieves or removes instance of task
- POST/GET to `/api/version/task/:id/stage/` - creates new or get list of stages in task
- PUT/PATCH/GET/DELETE to `/api/version/task/:id/stage/:stage_id` - updates, retrieves or removes instance of stage in task.

To attach a view to an API insert the following code in `settings.py`:

```
API[VST_API_VERSION][r'task'] = {
    'model': 'your_application.models.Task'
}
```

For primary access to generated view inherit from *Task.generated_view* property.

To make translation on frontend easier use `_translate_model` attribute with `model_name`.

List of meta-attributes for generating a view:

- `_view_class` - list of additional view classes to inherit from, class or string to import with base class `ViewSet`. Constants are also supported:
 - `read_only` - to create a view only for viewing;
 - `list_only` - to create a view with list only;
 - `history` - to create a view only for viewing and deleting records.

CRUD-view is applied by default.

- `_serializer_class` - class of API serializer; use this attribute to specify parent class for auto-generated serializers. Default is `vstutils.api.serializers.VSTSerializer`. Can take a string to import, serializer class or `django.utils.functional.SimpleLazyObject`.
- `_serializer_class_name` - model name for OpenAPI definitions. This would be a model name in generated admin interface. Default is name of model class.
- `_list_fields` or `_detail_fields` - list of fields which will be listed in entity list or detail view accordingly. Same as DRF serializers meta-attribute “fields”.
- `_override_list_fields` or `_override_detail_fields` - mapping with names and field types that will be redeclared in serializer attributes (think of it as declaring fields in DRF `ModelSerializer`).
- `_properties_groups` - dict with key as group name and value as list of fields(str). Allows to group fields in sections on frontend.
- `_view_field_name` - name of field frontend shows as main view name.
- `_non_bulk_methods` - list of methods which must not used via bulk requests.
- `_extra_serializer_classes` - mapping with additional serializers in viewset. For example, custom serializer, which will compute smth in action (mapping name). Value can be string for import. Important note: setting `model` attribute to `None` allows to use standard serializer generation mechanism and get fields from a list or detail serializer (set `__inject_from__` serializer’s meta attribute to `list` or `detail` accordingly). In some cases, it is required to pass the model to the serializer. For these purposes, the constant `LAZY_MODEL` can be used as a meta attribute. Each time the serializer is used, the exact model where this serializer was declared will be set.
- `_filterset_fields` - list/dict of filterset names for API-filtering. Default is list of fields in list view. During processing a list of fields checks for the presence of special field names and inherit additional parent classes. If the list contains `id`, class will inherit from `vstutils.api.filters.DefaultIDFilter`. If the list contains `name`, class will inherit from `vstutils.api.filters.DefaultNameFilter`. If both conditions are present, inheritance will be from all of the above classes. Possible values include `list` of fields to filter or `dict` where key is a field name and value is a Filter class. Dict extends attribute functionality and provides ability to override filter field class (`None` value disables overriding).
- `_search_fields` - tuple or list of fields using for search requests. By default (or `None`) get all filterable fields in detail view.
- `_copy_attrs` - list of model-instance attributes indicates that object is copyable with this attrs.

- `_nested` - key-value mapping with nested views (key - nested name, kwargs for `vstutils.api.decorators.nested_view` decorator but supports `model` attribute as nested). `model` can be string for import.
- `_extra_view_attributes` - key-value mapping with additional view attributes, but has less priority over generated attributes.

In common, you can also add custom attributes to override or extend the default list of processing classes. Supported view attributes are `filter_backends`, `permission_classes`, `authentication_classes`, `throttle_classes`, `renderer_classes` and `parser_classes`. List of meta-attributes for settings of view is looks like:

- `_pre_{attribute}` - List of classes included before defaults.
- `_{attribute}` - List of classes included after defaults.
- `_override_{attribute}` - boolean flag indicates that attribute override default viewset (otherwise appends). Default is `False`.

Note: You may need to create an [action](#)⁴ on generated view. Use `vstutils.models.decorators.register_view_action` decorator with the `detail` argument to determine applicability to a list or detail entry. In this case, the decorated method will take an instance of the view object as `self` attribute.

Note: In some cases, inheriting models may require to inherit Meta class from the base model. If the Meta is explicitly declared in the base class, then you can get it through the attribute `OriginalMeta` and use it for inheritance.

Note: Docstring of model will be reused for view descriptions. It is possible to write both a general description for all actions and description for each action using the following syntax:

```
General description for all actions.

action_name:
    Description for this action.

another_action:
    Description for another action.
```

More information about Models you can find in [Django Models documentation](#)⁵.

If you don't need to create custom [serializers](#) or [view sets](#), you can go to this [stage](#).

⁴ <https://www.django-rest-framework.org/api-guide/viewsets/#marking-extra-actions-for-routing>

⁵ <https://docs.djangoproject.com/en/4.1/topics/db/models/>

1.2.2 Serializers creation

Note - If you don't need custom serializer you can skip this section

Firstly, you need to create file `serializers.py` in the `/{{app_dir}}/{{app_name}}/{{app_name}}/` directory.

Then you need to add some code like this to `serializers.py`:

```
from datetime import datetime
from vstutils.api import serializers as vst_serializers
from . import models as models

class StageSerializer(models.Stage.generated_view.serializer_class):

    class Meta:
        model = models.Stage
        fields = ('id',
                  'name',
                  'order',)

    def update(self, instance, validated_data):
        # Put custom logic to serializer update
        instance.last_update = datetime.utcnow()
        super().update(instance, validated_data)
```

More information about Serializers you can find in [Django REST Framework documentation for Serializers](#)⁶.

1.2.3 Views creation

Note - If you don't need custom view set you can skip this section

Firstly, you need to create file `views.py` in the `/{{app_dir}}/{{app_name}}/{{app_name}}/` directory.

Then you need to add some code like this to `views.py`:

```
from vstutils.api import decorators as deco
from vstutils.api.base import ModelViewSet
from . import serializers as sers
from .models import Stage, Task

class StageViewSet(Stage.generated_view):
    serializer_class_one = sers.StageSerializer

'''
Decorator, that allows to put one view into another
* 'tasks' - suburl for nested view
* 'methods=["get"]' - allowed methods for this view
* 'manager_name='hosts' - Name of related QuerySet to the child model instances.
→ (we set it in HostGroup model as "hosts = models.ManyToManyField(Host)")
* 'view=Task.generated_view' - Nested view, that will be child view for.
→ decorated view
'''
@nested_view('stage', view=StageViewSet)
```

(continues on next page)

⁶ <https://www.django-rest-framework.org/api-guide/serializers/#modelserializer>

(continued from previous page)

```
class TaskViewSet (Task.generated_view):
    '''
    Task operations.
    '''
```

More information about Views and ViewSets you can find in [Django REST Framework documentation for views](#)⁷.

1.2.4 Adding Models to API

To add created Models to the API you need to write something like this at the end of your `settings.py` file:

```
'''
Some code generated by VST Utils
'''

'''
Add Task view set to the API
Only 'root' (parent) views should be added there.
Nested views added automatically, that's why there is only Task view.
Stage view is added altogether with Task as nested view.
'''
API[VST_API_VERSION][r'task'] = {
    'view': 'newapp2.views.TaskViewSet'
}

'''
You can add model too.
All model generate base ViewSet with data that they have, if you don't create custom
↳ ViewSet or Serializer
'''
API[VST_API_VERSION][r'task'] = dict(
    model='newapp2.models.Task'
)

# Adds link to the task view to the GUI menu
PROJECT_GUI_MENU.insert(0, {
    'name': 'Task',
    # CSS class of font-awesome icon
    'span_class': 'fa fa-list-alt',
    'url': '/task'
}))
```

1.2.5 Migrations creation

To make migrations open `/{{app_dir}}/{{app_name}}/` directory and execute following command:

```
python -m {{app_name}} makemigrations {{app_name}}
```

More information about Migrations you can find in [Django Migrations documentation](#)⁸.

⁷ <https://www.django-rest-framework.org/api-guide/viewsets/>

⁸ <https://docs.djangoproject.com/en/4.1/topics/migrations/>

1.2.6 Migrations applying

To apply migrations you need to open `{{app_dir}}/{{app_name}}/` directory and execute following command:

```
python -m {{app_name}} migrate
```

1.2.7 Restart of Application

To restart your application, firstly, you need to stop it (if it was started before):

```
python -m {{app_name}} web stop=/tmp/{{app_name}}_web.pid
```

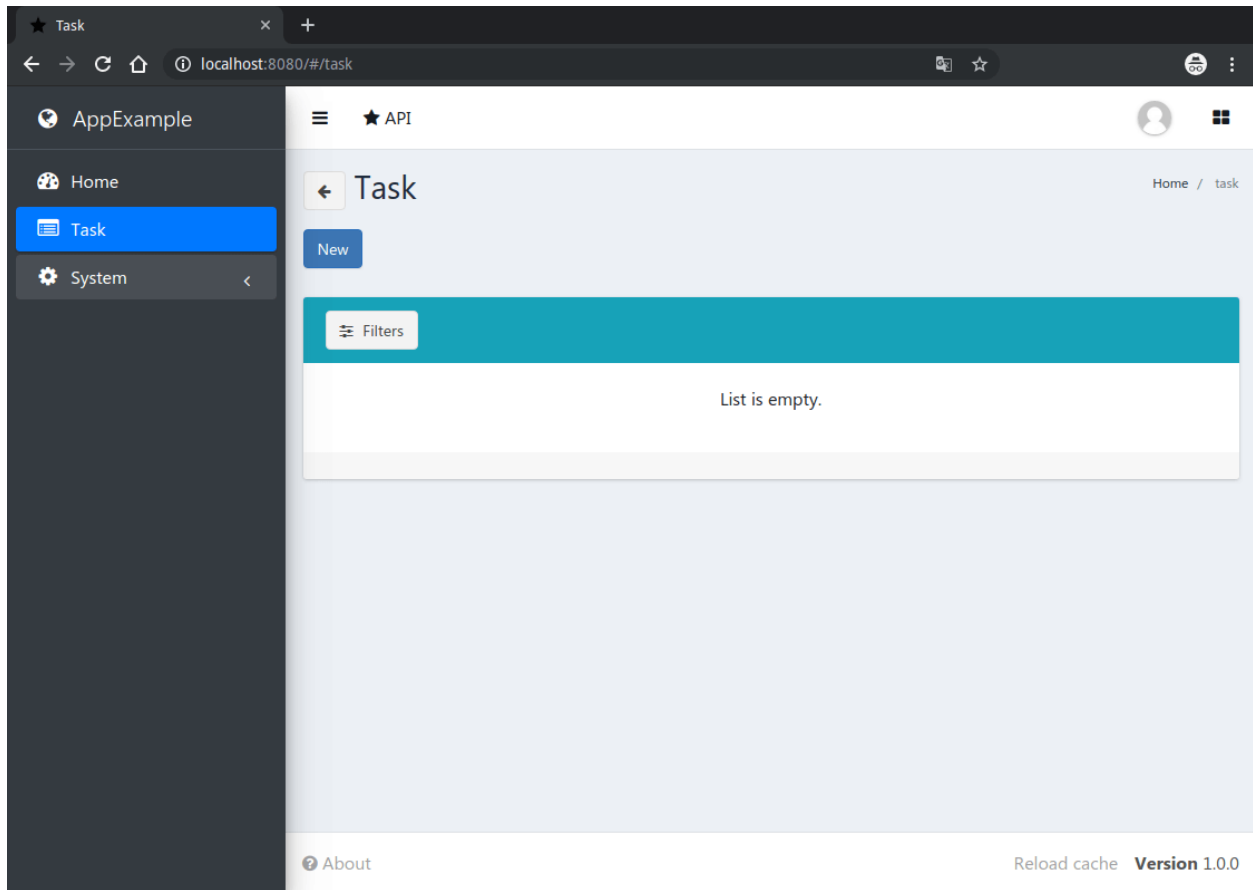
And then start it again:

```
python -m {{app_name}} web
```

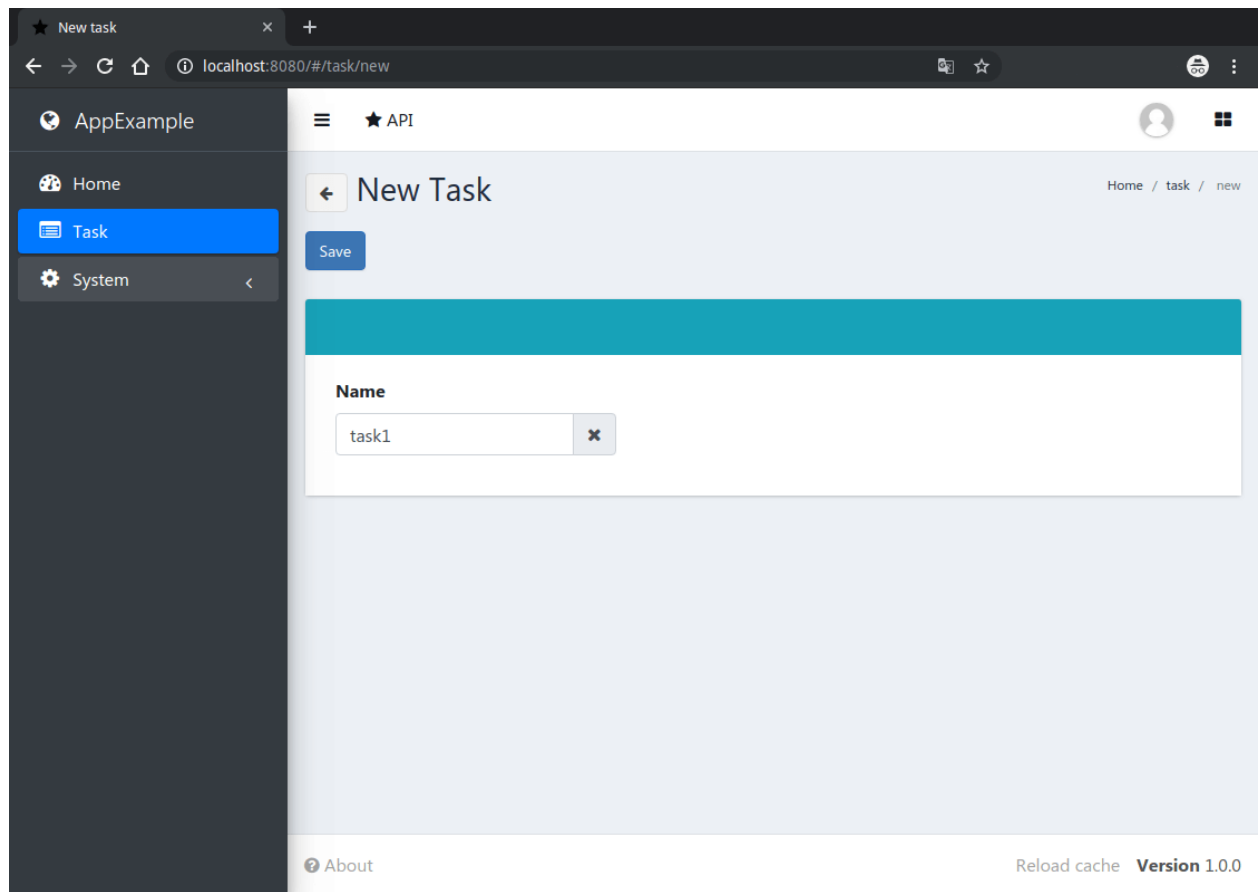
After cache reloading you will see following page:



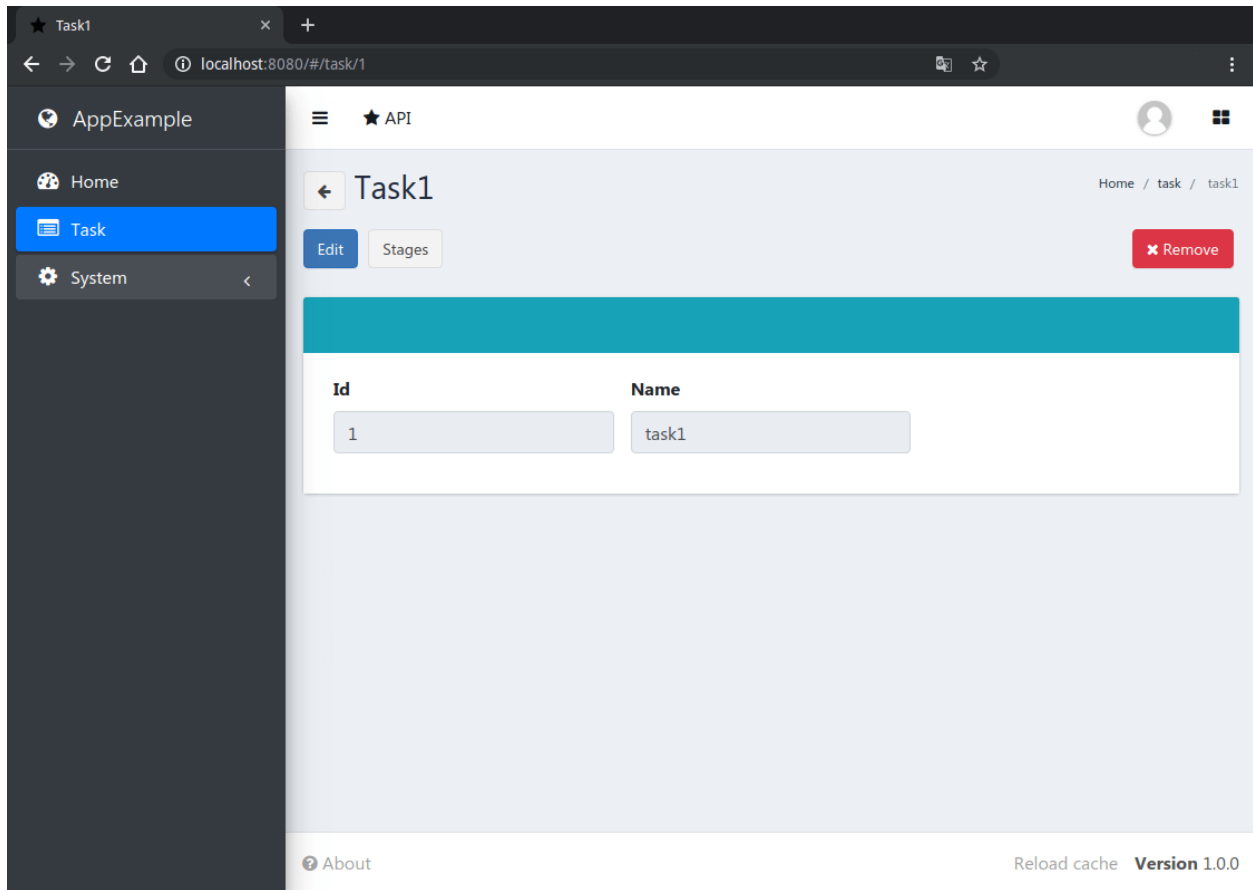
As you can see, link to new Task View has been added to the sidebar menu. Let's click on it.



There is no task instance in your app. Add it using 'new' button.



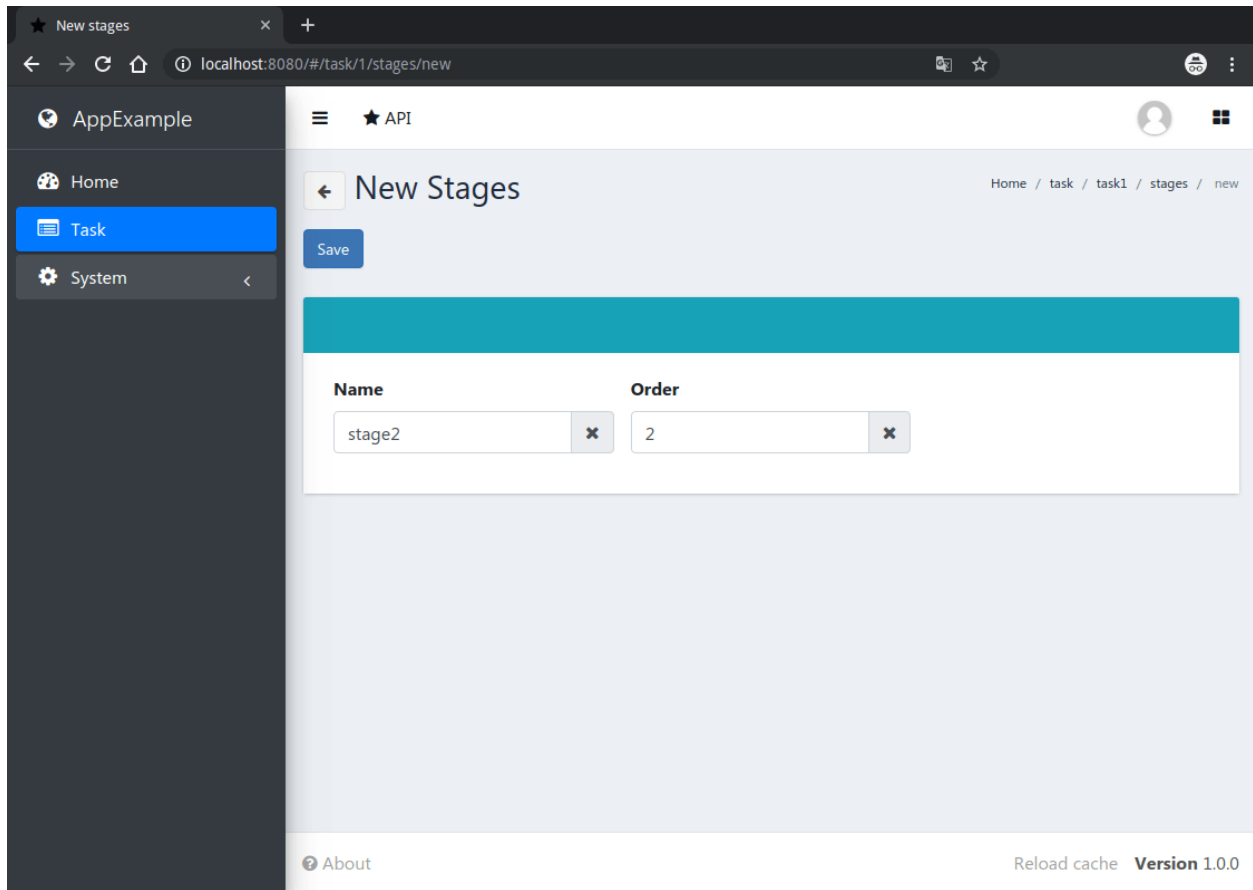
After creating a new task you'll see a following page:



As you can see, there is 'stages' button, that opens page with this task's stages list. Let's click on it.



There is no stage instances in your app. Let's create 2 new stages.





After stages creation page with stages list will looks like this:



Sorting by 'order' field works, as we mentioned in the our `models.py` file for Stage Model.

Additional information about Django and Django REST Framework you can find in [Django documentation](https://docs.djangoproject.com/en/4.1/)⁹ and [Django REST Framework documentation](https://www.django-rest-framework.org/)¹⁰.

⁹ <https://docs.djangoproject.com/en/4.1/>

¹⁰ <https://www.django-rest-framework.org/>

Configuration manual

2.1 Introduction

Though default configuration is suitable for many common cases, vstutils-based applications is highly configurable system. For advanced settings (scalability, dedicated DB, custom cache, logging or directories) you can configure vstutils-based application deeply by tweaking `/etc/{{app_name or app_lib_name}}/settings.ini`.

The most important thing to keep in mind when planning your application architecture is that vstutils-based applications have a service-oriented structure. To build a distributed scalable system you only need to connect to a shared *database*, shared *cache*, *locks* and a shared *rpc* service (MQ such as RabbitMQ, Redis, etc.). A shared file storage may be required in some cases, a but vstutils does not require it.

Let's cover the main sections of the config and its parameters:

2.2 Main settings

Section `[main]`.

This section is intended for settings related to whole vstutils-based application (both worker and web). Here you can specify verbosity level of vstutils-based application during work, which can be useful for troubleshooting (logging level etc). Also there are settings for changing timezone for whole app and allowed domains.

To use LDAP protocol, create following settings in section `[main]`.

```
ldap-server = ldap://server-ip-or-host:port
ldap-default-domain = domain.name
ldap-auth_format = cn=<username>,ou=your-group-name,<domain>
```

`ldap-default-domain` is an optional argument, that is aimed to make user authorization easier (without input of domain name).

`ldap-auth_format` is an optional argument, that is aimed to customize LDAP authorization. Default value: `cn=<username>,<domain>`

In the example above authorization logic will be the following:

1. System checks combination of login:password in database;
2. System checks combination of login:password in LDAP:

- if domain was mentioned, it will be set during authorization (if user enter login without `user@domain.name` or without `DOMAIN\user`);
- if authorization was successful and there is user with entered credentials in database, server creates session that user.
- **debug** - Enable debug mode. Default: false.
- **allowed_hosts** - Comma separated list of domains, which allowed to serve. Default: `*`.
- **first_day_of_week** - Integer value with first day of week. Default: `0`.
- **ldap-server** - LDAP server connection.
- **ldap-default-domain** - Default domain for auth.
- **ldap-auth_format** - Default search request format for auth. Default: `cn=<username>, <domain>`.
- **timezone** - Timezone for web-application. Default: UTC.
- **log_level** - Logging level. Default: WARNING.
- **enable_django_logs** - Enable or disable Django logger to output. Useful for debugging. Default: false.
- **enable_admin_panel** - Enable or disable Django Admin panel. Default: false.
- **enable_registration** - Enable or disable user self-registration. Default: false.
- **enable_user_self_remove** - Enable or disable user self-removing. Default: false.
- **auth-plugins** - Comma separated list of django authentication backends. Authorization attempt is made until the first successful one in order specified in the list.
- **auth-cache-user** - Enable or disable user instance caching. It increases session performance on each request but saves model instance in unsafe storage (default django cache). The instance is serialized to a string using the standard python module `pickle`¹¹ and then encrypted with *Vigenère cipher*¹². Read more in the `vstutils.utils.SecurePickling` documentation. Default: false.

2.3 Databases settings

Section `[databases]`.

The main section that is designed to manage multiple databases connected to the project.

These settings are for all databases and are vendor-independent, with the exception of tablespace management.

- **default_tablespace** - Default tablespace to use for models that don't specify one, if the backend supports it. Read more at [Declaring tablespaces for tables](#)¹³.
- **default_index_tablespace** - Default tablespace to use for indexes on fields that don't specify one, if the backend supports it. Read more at [Declaring tablespaces for indexes](#)¹⁴.
- **databases_without_cte_support** - A comma-separated list of database section names that do not support CTEs (Common Table Expressions).

¹¹ <https://docs.python.org/3.6/library/pickle.html#module-pickle>

¹² https://en.wikipedia.org/wiki/Vigenère_cipher

¹³ <https://docs.djangoproject.com/en/4.2/topics/db/tablespaces/#declaring-tablespaces-for-tables>

¹⁴ <https://docs.djangoproject.com/en/4.2/topics/db/tablespaces/#declaring-tablespaces-for-indexes>

Warning: Although MariaDB supports Common Table Expressions, but database connected to MariaDB still needs to be added to `databases_without_cte_support` list. The problem is that the implementation of recursive queries in the MariaDB does not allow using it in a standard form. MySQL (since 8.0) works as expected.

Also, all subsections of this section are available connections to the DBMS. So the `databases.default` section will be used by django as the default connection.

Here you can change settings related to database, which vstutils-based application will use. vstutils-based application supports all databases supported by django. List of supported out of the box: SQLite (default choice), MySQL, Oracle, or PostgreSQL. Configuration details available at [Django database documentation](#)¹⁵. To run vstutils-based application at multiple nodes (cluster), use client-server database (SQLite not suitable) shared for all nodes.

You can also set the base template for connecting to the database in the database section.

Section `[database]`.

This section is designed to define the basic template for connections to various databases. This can be useful to reduce the list of settings in the `databases.*` subsections by setting the same connection for a different set of databases in the project. For more details read the django docs about [Multiple databases](#)¹⁶

There is a list of settings, required for MySQL/MariaDB database.

Firstly, if you use MySQL/MariaDB and you have set timezone different from “UTC” you should run command below:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
```

Secondly, to use MySQL/MariaDB set following options in `settings.ini` file:

```
[database.options]
connect_timeout = 10
init_command = SET sql_mode='STRICT_TRANS_TABLES', default_storage_engine=INNODB,
↳ NAMES 'utf8', CHARACTER SET 'utf8', SESSION collation_connection = 'utf8_unicode_ci'
```

Finally, add some options to MySQL/MariaDB configuration:

```
[client]
default-character-set=utf8
init_command = SET collation_connection = @@collation_database

[mysqld]
character-set-server=utf8
collation-server=utf8_unicode_ci
```

2.4 Cache settings

Section `[cache]`.

This section is cache backend related settings used by vstutils-based application. vstutils supports all cache backends that Django does. Filesystem, in-memory, memcached are supported out of the box and many others are supported with additional plugins. You can find details about cache configs supported [Django caches documentation](#)¹⁷. In clusters we advice to share cache between nodes to improve performance using client-server cache realizations. We recommend to use Redis in production environments.

¹⁵ <https://docs.djangoproject.com/en/4.2/ref/settings/#databases>

¹⁶ <https://docs.djangoproject.com/en/4.2/topics/db/multi-db/#multiple-databases>

¹⁷ <https://docs.djangoproject.com/en/4.2/ref/settings/#caches>

2.5 Locks settings

Section `[locks]`.

Locks is a system that vstutils-based application uses to avoid damage from parallel actions working on the same entity simultaneously. It is based on Django cache, so there is another bunch of same settings as *cache*. And why there is another section for them, you may ask. Because cache backend is used for locking must provide some guarantees, which do not required to usual cache: it **MUST** be shared for all vstutils-based application threads and nodes. So, for example, in-memory backend is not suitable. In case of clusterization we strongly recommend to use Redis or Memcached as backend for that purpose. Cache and locks backend can be the same, but don't forget about requirement we said above.

2.6 Session cache settings

Section `[session]`.

vstutils-based application store sessions in *database*, but for better performance, we use a cache-based session backend. It is based on Django cache, so there is another bunch of same settings as *cache*. By default, settings are got from *cache*.

2.7 Rpc settings

Section `[rpc]`.

vstutils-based application uses Celery for long-running async tasks. Celery is based on message queue concept, so between web-service and workers running under Celery must be some kind of message broker (RabbitMQ or something). Those settings relate to this broker and Celery itself. Those kinds of settings: broker backend, number of worker-processes per node and some settings used for troubleshoot server-broker-worker interaction problems.

This section require vstutils with *rpc* extra dependency.

- **connection** - Celery *broker connection*¹⁸. Default: `filesystem:///var/tmp`.
- **concurrency** - Count of celery worker threads. Default: 4.
- **heartbeat** - Interval between sending heartbeat packages, which says that connection still alive. Default: 10.
- **enable_worker** - Enable or disable worker with webserver. Default: true.

The following variables from Django settings¹⁹ are also supported (with the corresponding types):

- **prefetch_multiplier** - `CELERYD_PREFETCH_MULTIPLIER`²⁰
- **max_tasks_per_child** - `CELERYD_MAX_TASKS_PER_CHILD`²¹
- **results_expiry_days** - `CELERY_RESULT_EXPIRES`²²
- **default_delivery_mode** - `CELERY_DEFAULT_DELIVERY_MODE`²³
- **task_send_sent_event** - `CELERY_DEFAULT_DELIVERY_MODE`²⁴
- **worker_send_task_events** - `CELERY_DEFAULT_DELIVERY_MODE`²⁵

¹⁸ <http://docs.celeryproject.org/en/latest/userguide/configuration.html#conf-broker-settings>

¹⁹ <http://docs.celeryproject.org/en/latest/userguide/configuration.html#new-lowercase-settings>

²⁰ http://docs.celeryproject.org/en/latest/userguide/configuration.html#std-setting-worker_prefetch_multiplier

²¹ http://docs.celeryproject.org/en/latest/userguide/configuration.html#std-setting-worker_max_tasks_per_child

²² http://docs.celeryproject.org/en/latest/userguide/configuration.html#std-setting-result_expires

²³ <http://docs.celeryproject.org/en/latest/userguide/configuration.html#task-default-delivery-mode>

²⁴ http://docs.celeryproject.org/en/latest/userguide/configuration.html#task_send_sent_event

²⁵ http://docs.celeryproject.org/en/latest/userguide/configuration.html#worker_send_task_events

2.8 Worker settings

Section `[worker]`.

Warning: These settings are needed only for rpc-enabled applications.

Celery worker options:

- **loglevel** - Celery worker log level. Default: from *main* section `log_level`.
- **pidfile** - Celery worker pidfile. Default: `/run/{app_name}_worker.pid`
- **autoscale** - Options for autoscaling. Two comma separated numbers: `max,min`.
- **beat** - Enable or disable celery beat scheduler. Default: `true`.

See other settings via `celery worker --help` command.

2.9 SMTP settings

Section `[mail]`.

Django comes with several email sending backends. With the exception of the SMTP backend (default when `host` is set), these backends are useful only in testing and development.

Applications based on vstutils uses only `smtp` and `console` backends.

- **host** - IP or domain for smtp-server. If it not set vstutils uses `console` backends. Default: `None`.
- **port** - Port for smtp-server connection. Default: `25`.
- **user** - Username for smtp-server connection. Default: `" "`.
- **password** - Auth password for smtp-server connection. Default: `" "`.
- **tls** - Enable/disable tls for smtp-server connection. Default: `False`.
- **send_confirmation** - Enable/disable confirmation message after registration. Default: `False`.
- **authenticate_after_registration** - Enable/disable autologin after registration confirmation. Default: `False`.

2.10 Web settings

Section `[web]`.

These settings are related to web-server. Those settings includes: `session_timeout`, `static_files_url` and pagination limit.

- **allow_cors** - enable cross-origin resource sharing. Default: `False`.
- **cors_allowed_origins**, **cors_allowed_origins_regexes**, **cors_expose_headers**, **cors_allow_methods**, **cors_allow_headers**, **cors_preflight_max_age** - [Settings²⁶](https://github.com/adamchainz/django-cors-headers#configuration) from `django-cors-headers` lib with their defaults.
- **enable_gravatar** - Enable/disable gravatar service using for users. Default: `True`.
- **rest_swagger_description** - Help string in Swagger schema. Useful for dev-integrations.

²⁶ <https://github.com/adamchainz/django-cors-headers#configuration>

- **openapi_cache_timeout** - Cache timeout for storing schema data. Default: 120.
- **health_throttle_rate** - Count of requests to `/api/health/` endpoint. Default: 60.
- **bulk_threads** - Threads count for PATCH `/api/endpoint/` endpoint. Default: 3.
- **session_timeout** - Session lifetime. Default: 2w (two weeks).
- **etag_default_timeout** - Cache timeout for Etag headers to control models caching. Default: 1d (one day).
- **rest_page_limit** and **page_limit** - Default limit of objects in API list. Default: 1000.
- **session_cookie_domain** - The domain to use for session cookies. Read [more](#)²⁷. Default: None.
- **csrf_trusted_origins** - A list of hosts which are trusted origins for unsafe requests. Read [more](#)²⁸. Default: from **session_cookie_domain**.
- **case_sensitive_api_filter** - Enables/disables case sensitive search for name filtering. Default: True.
- **secure_proxy_ssl_header_name** - Header name which activates SSL urls in responses. Read [more](#)²⁹. Default: `HTTP_X_FORWARDED_PROTOCOL`.
- **secure_proxy_ssl_header_value** - Header value which activates SSL urls in responses. Read [more](#)³⁰. Default: `https`.

The following variables from Django settings are also supported (with the corresponding types):

- **secure_browser_xss_filter** - `SECURE_BROWSER_XSS_FILTER`³¹
- **secure_content_type_nosniff** - `SECURE_CONTENT_TYPE_NOSNIFF`³²
- **secure_hsts_include_subdomains** - `SECURE_HSTS_INCLUDE_SUBDOMAINS`³³
- **secure_hsts_preload** - `SECURE_HSTS_PRELOAD`³⁴
- **secure_hsts_seconds** - `SECURE_HSTS_SECONDS`³⁵
- **password_reset_timeout_days** - `PASSWORD_RESET_TIMEOUT_DAYS`³⁶
- **request_max_size** - `DATA_UPLOAD_MAX_MEMORY_SIZE`³⁷
- **x_frame_options** - `X_FRAME_OPTIONS`³⁸
- **use_x_forwarded_host** - `USE_X_FORWARDED_HOST`³⁹
- **use_x_forwarded_port** - `USE_X_FORWARDED_PORT`⁴⁰

The following settings affects prometheus metrics endpoint (which can be used for monitoring application):

- **metrics_throttle_rate** - Count of requests to `/api/metrics/` endpoint. Default: 120.
- **enable_metrics** - Enable/disable `/api/metrics/` endpoint for app. Default: true

²⁷ https://docs.djangoproject.com/en/4.2/ref/settings/#std:setting-SESSION_COOKIE_DOMAIN

²⁸ <https://docs.djangoproject.com/en/4.2/ref/settings/#csrf-trusted-origins>

²⁹ <https://docs.djangoproject.com/en/4.2/ref/settings/#secure-proxy-ssl-header>

³⁰ <https://docs.djangoproject.com/en/4.2/ref/settings/#secure-proxy-ssl-header>

³¹ <https://docs.djangoproject.com/en/4.2/ref/settings/#secure-browser-xss-filter>

³² <https://docs.djangoproject.com/en/4.2/ref/settings/#secure-content-type-nosniff>

³³ <https://docs.djangoproject.com/en/4.2/ref/settings/#secure-hsts-include-subdomains>

³⁴ <https://docs.djangoproject.com/en/4.2/ref/settings/#secure-hsts-preload>

³⁵ <https://docs.djangoproject.com/en/4.2/ref/settings/#secure-hsts-seconds>

³⁶ https://docs.djangoproject.com/en/4.2/ref/settings/#std:setting-PASSWORD_RESET_TIMEOUT

³⁷ https://docs.djangoproject.com/en/4.2/ref/settings/#std:setting-DATA_UPLOAD_MAX_MEMORY_SIZE

³⁸ <https://docs.djangoproject.com/en/4.2/ref/settings/#x-frame-options>

³⁹ <https://docs.djangoproject.com/en/4.2/ref/settings/#use-x-forwarded-host>

⁴⁰ <https://docs.djangoproject.com/en/4.2/ref/settings/#use-x-forwarded-port>

- **metrics_backend** - Python class path with metrics collector backend. Default: `vstutils.api.metrics.DefaultBackend` Default backend collects metrics from uwsgi workers and python version info.

Section `[uvicorn]`.

You can configure the necessary settings to run the uvicorn server. `vstutils` supports almost all options from the cli, except for those that configure the application and connection.

See all available uvicorn settings via `uvicorn --help` command.

2.11 Centrifugo client settings

Section `[centrifugo]`.

To install app with centrifugo client, `[centrifugo]` section must be set. Centrifugo is used by application to auto-update page data. When user change some data, other clients get notification on `subscriptions_update` channel with model label and primary key. Without the service all GUI-clients get page data every 5 seconds (by default).

- **address** - Centrifugo server address.
- **api_key** - API key for clients.
- **token_hmac_secret_key** - API key for jwt-token generation.
- **timeout** - Connection timeout.
- **verify** - Connection verification.
- **subscriptions_prefix** - Prefix used for generating update channels, by default “`{VST_PROJECT}.update`”.

Note: These settings also add parameters to the OpenAPI schema and change how the auto-update system works in the GUI. `token_hmac_secret_key` is used for jwt-token generation (based on session expiration time). Token will be used for Centrifugo-JS client.

2.12 Storage settings

Section `[storages]`.

Applications based on `vstutils` supports filesystem storage out of box. Setup `media_root` and `media_url` in `[storages.filesystem]` section to configure custom media dir and relative url. By default it would be `{/path/to/project/module}/media` and `/media/`.

Applications based on `vstutils` supports store files in external services with [Apache Libcloud](http://libcloud.apache.org/)⁴¹ and [Boto3](https://boto3.amazonaws.com/v1/documentation/api/latest/index.html)⁴².

Apache Libcloud settings grouped by sections named `[storages.libcloud.provider]`, where `provider` is name of storage. Each section has four keys: `type`, `user`, `key` and `bucket`. Read more about the settings in [django-storages libcloud docs](https://django-storages.readthedocs.io/en/latest/backends/apache_libcloud.html#libcloud-providers)⁴³

This setting is required to configure connections to cloud storage providers. Each entry corresponds to a single ‘bucket’ of storage. You can have multiple buckets for a single service provider (e.g., multiple S3 buckets), and you can define buckets at multiple providers.

⁴¹ <http://libcloud.apache.org/>

⁴² <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

⁴³ https://django-storages.readthedocs.io/en/latest/backends/apache_libcloud.html#libcloud-providers

For Boto3 all settings grouped by section named `[storages.boto3]`. Section must contain following keys: `access_key_id`, `secret_access_key`, `storage_bucket_name`. Read more about the settings in [django-storages amazon-S3 docs](https://django-storages.readthedocs.io/en/latest/backends/amazon-S3.html)⁴⁴

Storage has following priority to choose storage engine if multiple was provided:

1. Libcloud store when config contains this section.
2. Boto3 store, when you have section and has all required keys.
3. FileSystem store otherwise.

Once you have defined your Libcloud providers, you have an option of setting one provider as the default provider of Libcloud storage. You can do it by setup `[storages.libcloud.default]` section or `vstutils` will set the first storage as default.

If you configure default libcloud provider, `vstutils` will use it as global file storage. To override it set `default=django.core.files.storage.FileSystemStorage` in `[storages]` section. When `[storages.libcloud.default]` is empty `django.core.files.storage.FileSystemStorage` is used as default. To override it set `default=storages.backends.apache_libcloud.LibCloudStorage` in `[storages]` section and use Libcloud provider as default.

Here is example for boto3 connection to minio cluster with public read permissions, external proxy domain and internal connection support:

```
[storages.boto3]
access_key_id = EXAMPLE_KEY
secret_access_key = EXAMPLEKEY_SECRET
# connection to internal service behind proxy
s3_endpoint_url = http://127.0.0.1:9000/
# external domain to bucket 'media'
storage_bucket_name = media
s3_custom_domain = media-api.example.com/media
# external domain works behind tls
s3_url_protocol = https:
s3_secure_urls = true
# settings to connect as plain http for uploading
s3_verify = false
s3_use_ssl = false
# allow to save files with similar names by adding prefix
s3_file_overwrite = false
# disables query string auth and setup default acl as RO for public users
querystring_auth = false
default_acl = public-read
```

2.13 Throttle settings

Section `[throttle]`.

By including this section to your config, you can setup global and per-view throttle rates. Global throttle rates are specified under root `[throttle]` section. To specify per-view throttle rate, you need to include child section.

For example, if you want to apply throttle to `api/v1/author`:

```
[throttle.views.author]
rate=50/day
actions=create,update
```

⁴⁴ <https://django-storages.readthedocs.io/en/latest/backends/amazon-S3.html>

- **rate** - Throttle rate in format number_of_requests/time_period. Expected time_periods: second/minute/hour/day.
- **actions** - Comma separated list of drf actions. Throttle will be applied only on specified here actions. Default: update, partial_update.

More on throttling at [DRF Throttle docs](#)⁴⁵.

2.14 Production web settings

Section [uwsgi].

Settings related to web-server used by vstutils-based application in production (for deb and rpm packages by default). Most of them related to system paths (logging, PID-file and so on). More settings in [uWSGI docs](#)⁴⁶.

But keep in mind that uWSGI is deprecated and may be removed in future releases. Use the uvicorn settings to manage your app server.

2.15 Configuration options

This section contains additional information for configure additional elements.

1. If you need set https for your web settings, you can do it using HAProxy, Nginx, Traefik or configure it in settings.ini.

```
[uwsgi]
addrport = 0.0.0.0:8443

[uvicorn]
ssl_keyfile = /path/to/key.pem
ssl_certfile = /path/to/cert.crt
```

2. We strictly do not recommend running the web server from root. Use HTTP proxy to run on privileged ports.
3. You can use {ENV[HOME:-value]} (where HOME is environment variable, value is default value) in configuration values.
4. You can use environment variables for setup important settings. But config variables has more priority then env. Available settings are: DEBUG, DJANGO_LOG_LEVEL, TIMEZONE and some settings with [ENV_NAME] prefix.

For project without special settings and project levels named project these variables will start with PROJECT_ prefix. There is a list of these variables: {ENV_NAME}_ENABLE_ADMIN_PANEL, {ENV_NAME}_ENABLE_REGISTRATION, {ENV_NAME}_MAX_TFA_ATTEMPTS, {ENV_NAME}_ETAG_TIMEOUT, {ENV_NAME}_SEND_CONFIRMATION_EMAIL, {ENV_NAME}_SEND_EMAIL_RETRIES, {ENV_NAME}_SEND_EMAIL_RETRY_DELAY, {ENV_NAME}_AUTHENTICATE_AFTER_REGISTRATION, {ENV_NAME}_MEDIA_ROOT (dir with uploads), {ENV_NAME}_GLOBAL_THROTTLE_RATE, and {ENV_NAME}_GLOBAL_THROTTLE_ACTIONS.

There are also URI-specific variables for connecting to various services such as databases and caches. There are DATABASE_URL, CACHE_URL, LOCKS_CACHE_URL, SESSIONS_CACHE_URL and ETAG_CACHE_URL. As you can see from the names, they are closely related to the keys and names of the corresponding config sections.

5. We recommend to install uvloop to your environment and setup loop = uvloop in [uvicorn] section for performance reasons.

⁴⁵ <https://www.django-rest-framework.org/api-guide/throttling/>

⁴⁶ <http://uwsgi-docs.readthedocs.io/en/latest/Configuration.html>

VST Utils framework uses Django, Django Rest Framework, drf-yasg and Celery.

3.1 Models

A model is the single, definitive source of truth about your data. It contains essential fields and behavior for the data you're storing. Usually best practice is to avoid writing views and serializers manually, as BModel provides plenty of Meta attributes to autogenerate serializers and views for many use cases.

Default Django model classes overrides in *vstutils.models* module.

class *vstutils.models.BModel*(*args, **kwargs)

Default model class that generates model viewset, separate serializers for list() and retrieve(), filters, api endpoints and nested views.

Examples:

```
from django.db import models
from rest_framework.fields import ChoiceField
from vstutils.models import BModel

class Stage(BModel):
    name = models.CharField(max_length=256)
    order = models.IntegerField(default=0)

    class Meta:
        default_related_name = "stage"
        ordering = ('order', 'id',)
        # fields which would be showed on list.
        _list_fields = [
            'id',
            'name',
        ]
        # fields which would be showed on detail view and creation.
        _detail_fields = [
            'id',
            'name',
            'order'
        ]
```

(continues on next page)

(continued from previous page)

```

        # make order as choices from 0 to 9
        _override_detail_fields = {
            'order': ChoiceField((str(i) for i in range(10)))
        }

class Task(BModel):
    name = models.CharField(max_length=256)
    stages = models.ManyToManyField(Stage)
    _translate_model = 'Task'

    class Meta:
        # fields which would be showed.
        _list_fields = [
            'id',
            'name',
        ]
        # create nested views from models
        _nested = {
            'stage': {
                'allow_append': False,
                'model': Stage
            }
        }

```

In this case, you create models which could be converted to simple view, where:

- POST/GET to `/api/version/task/` - creates new or get list of tasks
- PUT/PATCH/GET/DELETE to `/api/version/task/:id/` - updates, retrieves or removes instance of task
- POST/GET to `/api/version/task/:id/stage/` - creates new or get list of stages in task
- PUT/PATCH/GET/DELETE to `/api/version/task/:id/stage/:stage_id` - updates, retrieves or removes instance of stage in task.

To attach a view to an API insert the following code in `settings.py`:

```

API[VST_API_VERSION][r'task'] = {
    'model': 'your_application.models.Task'
}

```

For primary access to generated view inherit from `Task.generated_view` property.

To make translation on frontend easier use `_translate_model` attribute with `model_name`.

List of meta-attributes for generating a view:

- `_view_class` - list of additional view classes to inherit from, class or string to import with base class `ViewSet`. Constants are also supported:
 - `read_only` - to create a view only for viewing;
 - `list_only` - to create a view with list only;
 - `history` - to create a view only for viewing and deleting records.

CRUD-view is applied by default.

- `_serializer_class` - class of API serializer; use this attribute to specify parent class for auto-generated serializers. Default is `vstutils.api.serializers.VSTSerializer`. Can take a string to import, serializer class or `django.utils.functional.SimpleLazyObject`.
- `_serializer_class_name` - model name for OpenAPI definitions. This would be a model name in generated admin interface. Default is name of model class.
- `_list_fields` or `_detail_fields` - list of fields which will be listed in entity list or detail view accordingly. Same as DRF serializers meta-attribute “fields”.
- `_override_list_fields` or `_override_detail_fields` - mapping with names and field types that will be redeclared in serializer attributes (think of it as declaring fields in DRF `ModelSerializer`).
- `_properties_groups` - dict with key as group name and value as list of fields (str). Allows to group fields in sections on frontend.
- `_view_field_name` - name of field frontend shows as main view name.
- `_non_bulk_methods` - list of methods which must not be used via bulk requests.
- `_extra_serializer_classes` - mapping with additional serializers in viewset. For example, custom serializer, which will compute smth in action (mapping name). Value can be string for import. Important note: setting `model` attribute to `None` allows to use standard serializer generation mechanism and get fields from a list or detail serializer (set `__inject_from__` serializer’s meta attribute to `list` or `detail` accordingly). In some cases, it is required to pass the model to the serializer. For these purposes, the constant `LAZY_MODEL` can be used as a meta attribute. Each time the serializer is used, the exact model where this serializer was declared will be set.
- `_filterset_fields` - list/dict of filterset names for API-filtering. Default is list of fields in list view. During processing a list of fields checks for the presence of special field names and inherit additional parent classes. If the list contains `id`, class will inherit from `vstutils.api.filters.DefaultIDFilter`. If the list contains `name`, class will inherit from `vstutils.api.filters.DefaultNameFilter`. If both conditions are present, inheritance will be from all of the above classes. Possible values include `list` of fields to filter or `dict` where key is a field name and value is a Filter class. Dict extends attribute functionality and provides ability to override filter field class (`None` value disables overriding).
- `_search_fields` - tuple or list of fields using for search requests. By default (or `None`) get all filterable fields in detail view.
- `_copy_attrs` - list of model-instance attributes indicates that object is copyable with this attrs.
- `_nested` - key-value mapping with nested views (key - nested name, kwargs for `vstutils.api.decorators.nested_view` decorator but supports `model` attribute as nested). `model` can be string for import.
- `_extra_view_attributes` - key-value mapping with additional view attributes, but has less priority over generated attributes.

In common, you can also add custom attributes to override or extend the default list of processing classes. Supported view attributes are `filter_backends`, `permission_classes`, `authentication_classes`, `throttle_classes`, `renderer_classes` and `parser_classes`. List of meta-attributes for settings of view is looks like:

- `_pre_{attribute}` - List of classes included before defaults.
- `_{attribute}` - List of classes included after defaults.
- `_override_{attribute}` - boolean flag indicates that attribute override default viewset (otherwise appends). Default is `False`.

Note: You may need to create an [action](#)⁴⁷ on generated view. Use `vstutils.models.decorators.register_view_action` decorator with the `detail` argument to determine applicability to a list or detail entry. In this case, the decorated method will take an instance of the view object as `self` attribute.

Note: In some cases, inheriting models may require to inherit Meta class from the base model. If the Meta is explicitly declared in the base class, then you can get it through the attribute `OriginalMeta` and use it for inheritance.

Note: Docstring of model will be reused for view descriptions. It is possible to write both a general description for all actions and description for each action using the following syntax:

```
General description for all actions.

action_name:
    Description for this action.

another_action:
    Description for another action.
```

hidden

If `hidden` is set to `True`, entry will be excluded from query in `BQuerySet`.

id

Primary field for select and search in API.

class `vstutils.models.Manager` (*args, **kwargs)

Default VSTUtils manager. Used by *BaseModel* and *BModel*. Uses *BQuerySet* as base.

class `vstutils.models.queryset.BQuerySet` (model=None, query=None, using=None, hints=None)

Represent a lazy database lookup for a set of objects. Allows to override default iterable class by *custom_iterable_class* attribute (class with `__iter__` method which returns generator of model objects) and default query class by *custom_query_class* attribute (class inherited from `django.db.models.sql.query.Query`).

cleared()

Filter queryset for models with attribute 'hidden' and exclude all hidden objects.

get_paginator (*args, **kwargs)

Returns initialized object of `vstutils.utils.Paginator` over current instance's `QuerySet`. All args and kwargs go to to `Paginator`'s constructor.

paged (*args, **kwargs)

Returns paginated data with custom `Paginator`-class. Uses *PAGE_LIMIT* from global settings by default.

class `vstutils.models.decorators.register_view_action` (*args, **kwargs)

Decorator for turning model methods to generated view [actions](#)⁴⁸. The decorated method becomes a method of generated view and *self* is a view object. See supported args in `vstutils.api.decorators.subaction()`.

Note: Sometimes you may need to use proxy models with a common set of actions. To receive the action by the proxy model, pass the named argument `inherit` with `True` value.

⁴⁷ <https://www.django-rest-framework.org/api-guide/viewsets/#marking-extra-actions-for-routing>

Note: Often, an action does not transfer any parameters and requires only sending an empty query. To speed up development, we set the default serializer to `vstutils.api.serializers.EmptySerializer`.

You can also use custom models without using database:

class `vstutils.models.custom_model.ExternalCustomModel` (*args, **kwargs)

This custom model is intended for self-implementation of requests to external services. The model allows you to pass filtering, limiting and sorting parameters to an external request, receiving already limited data.

To start using this model, it is enough to implement the `get_data_generator()` class method, which receives the query object with the necessary parameters as an argument.

class `vstutils.models.custom_model.FileModel` (*args, **kwargs)

Custom model which loads data from YAML-file instead of database. Path to the file stored in `FileModel.file_path` attribute.

Examples:

Source file stored in `/etc/authors.yaml` with content:

```
- name: "Sergey Klyuykov"
- name: "Michael Taran"
```

Example:

```
from vstutils.custom_model import FileModel, CharField

class Authors(FileModel):
    name = CharField(max_length=512)

    file_path = '/etc/authors.yaml'
```

class `vstutils.models.custom_model.ListModel` (*args, **kwargs)

Custom model which uses a list of dicts with data (attribute `ListModel.data`) instead of database records. Useful when you have a simple list of data.

Examples:

```
from vstutils.custom_model import ListModel, CharField

class Authors(ListModel):
    name = CharField(max_length=512)

    data = [
        {"name": "Sergey Klyuykov"},
        {"name": "Michael Taran"},
    ]
```

Sometimes, it may be necessary to switch the data source. For these purposes, you should use the `setup_custom_queryset_kwargs` function, which takes various named arguments, which are also passed to the data initialization function. One such argument for `ListModel` is `date_source`, which takes any iterable object.

Examples:

⁴⁸ <https://www.django-rest-framework.org/api-guide/viewsets/#marking-extra-actions-for-routing>

```

from vstutils.custom_model import ListModel, CharField

class Authors(ListModel):
    name = CharField(max_length=512)

qs = Authors.objects.setup_custom_queryset_kwargs(data_source=[
    {"name": "Sergey Klyuykov"},
    {"name": "Michael Taran"},
])

```

In this case, we setup source list via `setup_custom_queryset_kwargs` function, and any other chained call is going to work with this data.

data = []

List with data dicts. Empty by default.

class `vstutils.models.custom_model.ViewCustomModel` (*args, **kwargs)

This model implements the SQL View programming mechanism over other models. In the `get_view_queryset()` method, a query is prepared, and all further actions are implemented on top of it.

3.1.1 Model Fields

class `vstutils.models.fields.FkModelField` (to, on_delete, related_name=None, related_query_name=None, limit_choices_to=None, parent_link=False, to_field=None, db_constraint=True, **kwargs)

Extends `django.db.models.ForeignKey`⁴⁹. Use this field in `vstutils.models.BModel` to get `vstutils.api.FkModelField` in serializer. To set Foreign Key relation set `to` argument to string path to model or to Model Class as in `django.db.models.ForeignKey`⁵⁰

class `vstutils.models.fields.HTMLField` (*args, db_collation=None, **kwargs)

Extends `django.db.models.TextField`⁵¹. A simple field for storing HTML markup. The field is based on `django.db.models.TextField`⁵², therefore it does not support indexing and is not recommended for use in filters.

class `vstutils.models.fields.MultipleFieldFile` (instance, field, name)

Subclasses `django.db.models.fields.files.FieldFile`⁵³. Provides `MultipleFieldFile.save()` and `MultipleFieldFile.delete()` to manipulate the underlying file, as well as update the associated model instance.

delete (save=True)

Delete file from storage and from object attr.

save (name, content, save=True)

Save changes in file to storage and to object attr.

⁴⁹ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.ForeignKey>

⁵⁰ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.ForeignKey>

⁵¹ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

⁵² <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

⁵³ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.fields.files.FieldFile>

class `vstutils.models.fields.MultipleFileDescriptor` (*field*)

Subclasses `django.db.models.fields.files.FileDescriptor` to handle list of files. Return a list of *MultipleFieldFile* when accessed so you can write code like:

```
from myapp.models import MyModel
instance = MyModel.objects.get(pk=1)
instance.files[0].size
```

get_file (*file, instance*)

Always return valid `attr_class` object. For details on logic see `django.db.models.fields.files.FileDescriptor.__get__()`.

class `vstutils.models.fields.MultipleFileField` (***kwargs*)

Subclasses `django.db.models.fields.files.FileField`. Field for storing a list of Storage-kept files. All args passed to `FileField`.

attr_class

alias of *MultipleFieldFile*

descriptor_class

alias of *MultipleFileDescriptor*

class `vstutils.models.fields.MultipleFileMixin` (***kwargs*)

Mixin suited to use with `django.db.models.fields.files.FieldFile`⁵⁴ to transform it to a Field with list of files.

get_prep_value (*value*)

Prepare value for database insertion

pre_save (*model_instance, add*)

Call `.save()` method on every file in list

class `vstutils.models.fields.MultipleImageField` (***kwargs*)

Field for storing a list of storage-kept images. All args are passed to `django.db.models.fields.files.ImageField`, except `height_field` and `width_field`, they are not currently implemented.

attr_class

alias of *MultipleImageFieldFile*

descriptor_class

alias of *MultipleFileDescriptor*

class `vstutils.models.fields.MultipleImageFieldFile` (*instance, field, name*)

Subclasses *MultipleFieldFile* and *ImageFile* mixin, handles deleting `_dimensions_cache` when file is deleted.

class `vstutils.models.fields.MultipleNamedBinaryFileInJSONField` (**args, db_collation=None, **kwargs*)

Extends `django.db.models.TextField`⁵⁵. Use this field in `vstutils.models.BModel` to get `vstutils.api.MultipleNamedBinaryFileInJSONField` in serializer.

⁵⁴ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.fields.files.FieldFile>

⁵⁵ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

```
class vstutils.models.fields.MultipleNamedBinaryImageInJSONField (*args,  
                                                                    db_collation=None,  
                                                                    **kwargs)
```

Extends `django.db.models.TextField`⁵⁶. Use this field in `vstutils.models.BModel` to get `vstutils.api.MultipleNamedBinaryImageInJSONField` in serializer.

```
class vstutils.models.fields.NamedBinaryFileInJSONField (*args, db_collation=None,  
                                                            **kwargs)
```

Extends `django.db.models.TextField`⁵⁷. Use this field in `vstutils.models.BModel` to get `vstutils.api.NamedBinaryFileInJSONField` in serializer.

```
class vstutils.models.fields.NamedBinaryImageInJSONField (*args, db_collation=None,  
                                                            **kwargs)
```

Extends `django.db.models.TextField`⁵⁸. Use this field in `vstutils.models.BModel` to get `vstutils.api.NamedBinaryImageInJSONField` in serializer.

```
class vstutils.models.fields.WYSIWYGField (*args, db_collation=None, **kwargs)
```

Extends `django.db.models.TextField`⁵⁹. A simple field for storing Markdown data. The field is based on `django.db.models.TextField`⁶⁰, therefore it does not support indexing and is not recommended for use in filters.

3.2 Web API

Web API is based on Django Rest Framework with additional nested functions.

3.2.1 Fields

The Framework includes a list of convenient serializer fields. Some of them take effect only in generated admin interface. Additional serializer fields for generating OpenAPI and GUI.

```
class vstutils.api.fields.AutoCompletionField (*args, **kwargs)
```

Field that provides autocompletion on frontend, using specified list of objects.

Parameters

- **autocomplete** (*list*⁶¹, *tuple*⁶², *str*⁶³) – Autocompletion reference. You can set list/tuple with values or set OpenAPI schema definition name. For definition name GUI will find optimal link and will show values based on `autocomplete_property` and `autocomplete_represent` arguments.
- **autocomplete_property** (*str*⁶⁴) – this argument indicates which attribute will be get from OpenAPI schema definition model as value.
- **autocomplete_represent** – this argument indicates which attribute will be get from OpenAPI schema definition model as represent value.
- **use_prefetch** (*bool*⁶⁵) – prefetch values on frontend at list-view. Default is `True`.

⁵⁶ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

⁵⁷ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

⁵⁸ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

⁵⁹ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

⁶⁰ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

Note: Effective only in GUI. Works similar to *VSTCharField* in API.

class `vstutils.api.fields.Barcode128Field(*args, **kwargs)`

Simple string field. Value must always be a valid ASCII-string. The field is going to represent as Barcode (Code 128) in user interface.

Parameters

child (`rest_framework.fields.Field`) – original data field for serialization or deserialization. Default: `rest_framework.fields.CharField`

class `vstutils.api.fields BinFileInStringField(*args, **kwargs)`

Field extends *FileInStringField*, but works with binary (base64) files.

Parameters

media_types (`tuple`⁶⁶, `list`⁶⁷) – List of MIME types to select on the user's side. Supported syntax using *. Default: `['*/*']`

Note: Effective only in GUI. Works similar to *VSTCharField* in API.

class `vstutils.api.fields.CSVFileField(*args, **kwargs)`

Field extends *FileInStringField*, using for works with csv files. This field provides the display of the loaded data in the form of a table.

Parameters

- **items** (*Serializer*) – The config of the table. This is a drf or vst serializer which includes char fields which are the keys in the dictionaries into which the data from csv is serialized and the names for columns in a table. The fields must be in the order you want them to appear in the table. Following options may be included:
 - `label`: human readable column name
 - `required`: Defines whether the field should be required. False by default.
- **min_column_width** (`int`⁶⁸) – Minimum cell width. Default is 200 px.
- **delimiter** (`str`⁶⁹) – The delimiting character.
- **lineterminator** (`str`⁷⁰) – The newline sequence. Leave blank to auto-detect. Must be one of `\r`, `\n`, or `\r\n`.
- **quotechar** (`str`⁷¹) – The character used to quote fields.
- **escapechar** (`str`⁷²) – The character used to escape the quote character within a field.
- **media_types** (`tuple`⁷³, `list`⁷⁴) – List of MIME types to select on the user's side. Supported syntax using *. Default: `['text/csv']`

⁶¹ <https://docs.python.org/3.6/library/stdtypes.html#list>

⁶² <https://docs.python.org/3.6/library/stdtypes.html#tuple>

⁶³ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁶⁴ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁶⁵ <https://docs.python.org/3.6/library/functions.html#bool>

⁶⁶ <https://docs.python.org/3.6/library/stdtypes.html#tuple>

⁶⁷ <https://docs.python.org/3.6/library/stdtypes.html#list>

class `vstutils.api.fields.CommaMultiSelect` (*args, **kwargs)

Field containing a list of values with specified separator (default: “,”). Gets list of values from another model or custom list. Provides autocompletion as `AutoCompletionField`, but with comma-lists. Suited for property-fields in model where main logic is already implemented or with `model.CharField`.

Parameters

- **select** (`str`⁷⁵, `tuple`⁷⁶, `list`⁷⁷) – OpenAPI schema definition name or list with values.
- **select_separator** (`str`⁷⁸) – separator of values. Default is comma.
- **select_property**, **select_represent** – work as `autocomplete_property` and `autocomplete_represent`. Default is name.
- **use_prefetch** – prefetch values on frontend at list-view. Default is `False`.
- **make_link** – Show value as link to model. Default is `True`.
- **dependence** (`dict`⁷⁹) – Dictionary, where keys are name of field from the same model, and values are name of query filter. If at least one of the fields that we depend on is non nullable, required and set to null, autocompletion list will be empty and field will be disabled.

Note: Effective only in GUI. Works similar to `VSTCharField` in API.

class `vstutils.api.fields.CrontabField` (*args, **kwargs)

Simple crontab-like field which contains the schedule of cron entries to specify time. A crontab field has five fields for specifying day, date and time. * in the value field above means all legal values as in braces for that column.

The value column can have a * or a list of elements separated by commas. An element is either a number in the ranges shown above or two numbers in the range separated by a hyphen (meaning an inclusive range).

The time and date fields are:

field	allowed value
minute	0-59
hour	0-23
day of month	1-31
month	1-12
day of week	0-7 (0 or 7 is Sunday)

Default value of each field if not specified is *.

```

.----- minute (0 - 59)
| .----- hour (0 - 23)
| | .----- day of month (1 - 31)

```

(continues on next page)

⁶⁸ <https://docs.python.org/3.6/library/functions.html#int>
⁶⁹ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁷⁰ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁷¹ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁷² <https://docs.python.org/3.6/library/stdtypes.html#str>
⁷³ <https://docs.python.org/3.6/library/stdtypes.html#tuple>
⁷⁴ <https://docs.python.org/3.6/library/stdtypes.html#list>
⁷⁵ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁷⁶ <https://docs.python.org/3.6/library/stdtypes.html#tuple>
⁷⁷ <https://docs.python.org/3.6/library/stdtypes.html#list>
⁷⁸ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁷⁹ <https://docs.python.org/3.6/library/stdtypes.html#dict>

(continued from previous page)

```

| | | .----- month (1 - 12)
| | | .----- day of week (0 - 6) (Sunday=0 or 7)
| | | |
* * * * *

```

class `vstutils.api.fields.DeepFkField` (*only_last_child=False*, *parent_field_name='parent'*, ***kwargs*)

Extends *FkModelField*, but displays as tree on frontend.

Warning: This field does not support dependence. Use filters at your own risk, as it would rather break the tree structure.

Parameters

- **only_last_child** (*bool*⁸⁰) – if True then only allows a value to be selected if it has no children. Default is *False*
- **parent_field_name** (*str*⁸¹) – name of parent field in model. Default is *parent*

class `vstutils.api.fields.DependEnumField` (**args*, ***kwargs*)

Field extends *DynamicJsonTypeField* but its value is not transformed to json and would be given as is. Useful for *property*⁸² in models or for actions.

Parameters

- **field** (*str*⁸³) – field in model which value change will change type of current value.
- **types** – key-value mapping where key is value of subscribed field and value is type (in OpenAPI format) of current field.
- **choices** (*dict*⁸⁴) – variants of choices for different subscribed field values. Uses mapping where key is value of subscribed field and value is list with values to choice.

Note: Effective only in GUI. In API works similar to *VSTCharField* without value modification.

class `vstutils.api.fields.DependFromFkField` (**args*, ***kwargs*)

Field extends *DynamicJsonTypeField*. Validates field data by *field_attribute* chosen in related model. By default, any value of *field_attribute* validates as *VSTCharField*. To override this behavior set dict attribute `{field_attribute value}_fields_mapping` in related model where:

- **key** - string representation of value type which is received from related instance *field_attribute*.
- **value** - `rest_framework.Field` instance for validation.

Parameters

- **field** (*str*⁸⁵) – field in model which value change changes type of current value. Field must be *FkModelField*.
- **field_attribute** (*str*⁸⁶) – attribute of related model instance with name of type.

⁸⁰ <https://docs.python.org/3.6/library/functions.html#bool>

⁸¹ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁸² <https://docs.python.org/3.6/library/functions.html#property>

⁸³ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁸⁴ <https://docs.python.org/3.6/library/stdtypes.html#dict>

Warning: `field_attribute` in related model must be `rest_framework.fields.ChoiceField` or `GUI` will show field as simple text.

class `vstutils.api.fields.DynamicJsonTypeField(*args, **kwargs)`

Field which type is based on another field. It converts value to internal string and represent field as json object.

Parameters

- **field** ([str](#)⁸⁷) – field in model which value change will change type of current value.
- **types** – key-value mapping where key is value of subscribed field and value is type (in OpenAPI format) of current field.
- **choices** ([dict](#)⁸⁸) – variants of choices for different subscribed field values. Uses mapping where key is value of subscribed field and value is list with values to choice.
- **source_view** ([str](#)⁸⁹) – Allows to use parent views data as source for field creation. Exact view path (`/user/{id}/`) or relative parent specifier (`<<parent>>.<<parent>>.<<parent>>`) can be provided. For example if current page is `/user/1/role/2/` and `source_view` is `<<parent>>.<<parent>>` then data from `/user/1/` will be used. Only detail views if supported.

Note: Effective only in GUI. In API works similar to `VSTCharField` without value modifications.

class `vstutils.api.fields.FileInStringField(*args, **kwargs)`

Field extends `VSTCharField` and saves file's content as string.

Value must be text (not binary) and saves in model as is.

Parameters

media_types ([tuple](#)⁹⁰, [list](#)⁹¹) – List of MIME types to select on the user's side. Supported syntax using `*`. Default: `[' */* ']`

Note: Take effect only in GUI. In API it would behave as `VSTCharField`.

class `vstutils.api.fields.FkField(*args, **kwargs)`

Implementation of `ForeignKeyField`. You can specify which field of a related model will be stored in field (default: "id"), and which will represent field on frontend.

Parameters

- **select** ([str](#)⁹²) – OpenAPI schema definition name.
- **autocomplete_property** ([str](#)⁹³) – this argument indicates which attribute will be get from OpenAPI schema definition model as value. Default is `id`.
- **autocomplete_represent** – this argument indicates which attribute will be get from OpenAPI schema definition model as represent value. Default is `name`.

⁸⁵ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁸⁶ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁸⁷ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁸⁸ <https://docs.python.org/3.6/library/stdtypes.html#dict>

⁸⁹ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁹⁰ <https://docs.python.org/3.6/library/stdtypes.html#tuple>

⁹¹ <https://docs.python.org/3.6/library/stdtypes.html#list>

- **field_type** (*type*⁹⁴) – defines the autocomplete_property type for further definition in the schema and casting to the type from the api. Default is passthroughs but require *int* or *str* objects.
- **use_prefetch** (*bool*⁹⁵) – prefetch values on frontend at list-view. Default is `True`.
- **make_link** (*bool*⁹⁶) – show value as link to model. Default is `True`.
- **dependence** (*dict*⁹⁷) – dictionary, where keys are names of a field from the same model, and values are names of query filter. If at least one of the fields that we depend on is non nullable, required and set to null, autocomplete list will be empty and field will be disabled.

There are some special keys for dependence dictionary to get data that is stored on frontend without additional database query:

```
'<<pk>>' gets primary key of current instance,
'<<view_name>>' gets view name from Vue component,
'<<parent_view_name>>' gets parent view name from Vue component,
'<<view_level>>' gets view level,
'<<operation_id>>' gets operation_id,
'<<parent_operation_id>>' gets parent_operation_id.
```

Examples:

```
field = FkField(select=Category, dependence={'<<pk>>': 'my_filter'})
```

This filter will get pk of current object and make query on frontend `/category?my_filter=3` where `3` is primary key of current instance.

Parameters

filters (*dict*⁹⁸) – dictionary, where keys are names of a field from a related (by this `FkField`) model, and values are values of that field.

Note: Intersection of `dependence.values()` and `filters.keys()` will throw error to prevent ambiguous filtering.

Note: Effective only in GUI. Works similar to `rest_framework.fields.IntegerField` in API.

class `vstutils.api.fields.FkModelField` (**args*, ***kwargs*)

Extends `FkField`, but stores referred model class. This field is useful for `django.db.models.ForeignKey`⁹⁹ fields in model to set.

Parameters

- **select** (`vstutils.models.BModel`, `vstutils.api.serializers.VSTSerializer`) – model class (based on `vstutils.models.BModel`) or serializer class which used in API and has path in OpenAPI schema.

⁹² <https://docs.python.org/3.6/library/stdtypes.html#str>

⁹³ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁹⁴ <https://docs.python.org/3.6/library/functions.html#type>

⁹⁵ <https://docs.python.org/3.6/library/functions.html#bool>

⁹⁶ <https://docs.python.org/3.6/library/functions.html#bool>

⁹⁷ <https://docs.python.org/3.6/library/stdtypes.html#dict>

⁹⁸ <https://docs.python.org/3.6/library/stdtypes.html#dict>

- **autocomplete_property** (*str*¹⁰⁰) – this argument indicates which attribute will be get from OpenAPI schema definition model as value. Default is `id`.
- **autocomplete_represent** – this argument indicates which attribute will be get from OpenAPI schema definition model as represent value. Default is `name`.
- **use_prefetch** – prefetch values on frontend at list-view. Default is `True`.
- **make_link** – Show value as link to model. Default is `True`.

Warning: Model class get object from database during `.to_internal_value` execution. Be careful on mass save executions.

Warning: Permissions to model which is referred by this field, are not to be checked. You should check it manually in signals or validators.

class `vstutils.api.fields.HtmlField(*args, **kwargs)`

Field contains html text and marked as `format:html`. The field does not validate whether its content is HTML.

Warning: To avoid vulnerability, do not allow users to modify this data because users are able to execute their scripts.

Note: Effective only in GUI. Works similar to `VSTCharField` in API.

class `vstutils.api.fields.MaskedField(*args, **kwargs)`

Extends class `'rest_framework.serializers.CharField'`. Field that applies mask to value.

Parameters

mask (*dict*¹⁰¹, *str*¹⁰²) – `IMask`¹⁰³

Note: Effective only on frontend.

class `vstutils.api.fields.MultipleNamedBinaryFileInJsonField(*args, **kwargs)`

Extends `NamedBinaryFileInJsonField` but uses list of JSONs. Allows to operate with multiple files as list of `NamedBinaryFileInJsonField`.

Attrs: `NamedBinaryInJsonField.file`: if `True`, accept only subclasses of `File` as input. If `False`, accept only string input. Default: `False`.

file_field

alias of `MultipleFieldFile`

class `vstutils.api.fields.MultipleNamedBinaryImageInJsonField(*args, **kwargs)`

Extends `MultipleNamedBinaryFileInJsonField` but uses list of JSONs. Used for operating with multiple images and works as list of `NamedBinaryImageInJsonField`.

⁹⁹ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.ForeignKey>

¹⁰⁰ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹⁰¹ <https://docs.python.org/3.6/library/stdtypes.html#dict>

¹⁰² <https://docs.python.org/3.6/library/stdtypes.html#str>

¹⁰³ <https://imask.js.org/guide.html>

Parameters

background_fill_color (*str*¹⁰⁴) – Color to fill area that is not covered by image after cropping. Transparent by default but will be black if image format is not supporting transparency. Can be any valid CSS color.

class `vstutils.api.fields.NamedBinaryFileInJsonField(*args, **kwargs)`

Field that takes JSON with properties: * name - string - name of file; * mediaType - string - MIME type of file; * content - base64 string - content of file.

This field is useful for saving binary files with their names in `django.db.models.CharField`¹⁰⁵ or `django.db.models.TextField`¹⁰⁶ model fields. All manipulations with decoding and encoding binary content data executes on client. This imposes reasonable limits on file size.

Additionally, this field can construct `django.core.files.uploadedfile.SimpleUploadedFile` from incoming JSON and store it as file in `django.db.models.FileField`¹⁰⁷ if *file* argument is set to *True*

Attrs: `NamedBinaryInJsonField.file`: if *True*, accept only subclasses of `File` as input. If *False*, accept only string input. Default: *False*.

Note: Effective only in GUI. Works similar to *VSTCharField* in API.

class `vstutils.api.fields.NamedBinaryImageInJsonField(*args, **kwargs)`

Extends *NamedBinaryFileInJsonField* to represent image on frontend (if binary image is valid). Validate this field with *vstutils.api.validators.ImageValidator*.

Parameters

background_fill_color (*str*¹⁰⁸) – Color to fill area that is not covered by image after cropping. Transparent by default but will be black if image format is not supporting transparency. Can be any valid CSS color.

class `vstutils.api.fields.PasswordField(*args, **kwargs)`

Extends *CharField*¹⁰⁹ but in schema set format to *password*. Show all characters as asterisks instead of real value in GUI.

class `vstutils.api.fields.PhoneField(*args, **kwargs)`

Extends class 'rest_framework.serializers.CharField'. Field for phone in international format

class `vstutils.api.fields.QrCodeField(*args, **kwargs)`

Simple string field. The field is going to represent as *QrCode* in user interface.

Parameters

child (*rest_framework.fields.Field*) – original data field for serialization or deserialization. Default: `rest_framework.fields.CharField`

class `vstutils.api.fields.RatingField(min_value=0, max_value=5, step=1, front_style='stars', **kwargs)`

Extends class 'rest_framework.serializers.FloatField'. This field represents a rating form input on frontend. Grading limits can be specified with 'min_value=' and 'max_value=', defaults are 0 to 5. Minimal step between grades are specified in 'step=', default - 1. Frontend visual representation can be chosen with 'front_style=', available variants are listed in 'self.valid_front_styles'.

¹⁰⁴ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹⁰⁵ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.CharField>

¹⁰⁶ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.TextField>

¹⁰⁷ <https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.FileField>

¹⁰⁸ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹⁰⁹ <https://www.django-rest-framework.org/api-guide/fields/#charfield>

For 'slider' front style, you can specify slider color, by passing valid color to 'color='. For 'fa_icon' front style, you can specify FontAwesome icon that would be used for displaying rating, by passing a valid FontAwesome icon code to 'fa_class='.

Parameters

- **min_value** (*float*¹¹⁰, *int*¹¹¹) – minimal level
- **max_value** (*float*¹¹², *int*¹¹³) – maximal level
- **step** (*float*¹¹⁴, *int*¹¹⁵) – minimal step between levels
- **front_style** (*str*¹¹⁶) – visualization on frontend field. Allowed: ['stars', 'slider', 'fa_icon'].
- **color** (*str*¹¹⁷) – color of rating element (star, icon or slider) in css format
- **fa_class** (*str*¹¹⁸) – FontAwesome icon code

class `vstutils.api.fields.RedirectCharField(*args, **kwargs)`

Field for redirect by string. Often used in actions for redirect after execution.

Note: Effective only in GUI. Works similar to `rest_framework.fields.IntegerField` in API.

class `vstutils.api.fields.RedirectFieldMixin(**kwargs)`

Field mixin indicates that this field is used to send redirect address to frontend after some action.

Parameters

- **operation_name** (*str*¹¹⁹) – prefix for operation_id, for example if operation_id is *history_get* then operation_name is *history*
- **depend_field** (*str*¹²⁰) – name of the field that we depend on, its value will be used for operation_id
- **concat_field_name** (*bool*¹²¹) – if True then name of the field will be added at the end of operation_id

class `vstutils.api.fields.RedirectIntegerField(*args, **kwargs)`

Field for redirect by id. Often used in actions for redirect after execution.

Note: Effective only in GUI. Works similar to `rest_framework.fields.IntegerField` in API.

class `vstutils.api.fields.RelatedListField(related_name, fields, view_type='list',
serializer_class=None, **kwargs)`

Extends class `VSTCharField`. With this field you can output reverse ForeignKey relation as a list of related instances.

¹¹⁰ <https://docs.python.org/3.6/library/functions.html#float>

¹¹¹ <https://docs.python.org/3.6/library/functions.html#int>

¹¹² <https://docs.python.org/3.6/library/functions.html#float>

¹¹³ <https://docs.python.org/3.6/library/functions.html#int>

¹¹⁴ <https://docs.python.org/3.6/library/functions.html#float>

¹¹⁵ <https://docs.python.org/3.6/library/functions.html#int>

¹¹⁶ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹¹⁷ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹¹⁸ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹¹⁹ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹²⁰ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹²¹ <https://docs.python.org/3.6/library/functions.html#bool>

To use it, you specify ‘related_name’ kwarg (related_manager for reverse ForeignKey) and ‘fields’ kwarg (list or tuple of fields from related model, which needs to be included).

By default *VSTCharField* used to serialize all field values and represent it on frontend. You can specify *serializer_class* and override fields as you need. For example title, description and other field properties can be set to customize frontend behavior.

Parameters

- **related_name** (*str*¹²²) – name of a related manager for reverse foreign key
- **fields** (*list*¹²³ [*str*¹²⁴], *tuple*¹²⁵ [*str*¹²⁶]) – list of related model fields.
- **view_type** (*str*¹²⁷) – determines how field are represented on frontend. Must be either ‘list’ or ‘table’.
- **fields_custom_handlers_mapping** (*dict*¹²⁸) – includes custom handlers, where key: field_name, value: callable_obj that takes params: instance[dict], fields_mapping[dict], model, field_name[str]
- **serializer_class** (*type*¹²⁹) – Serializer to customize types of fields, if no serializer provided *VSTCharField* will be used for every field in *fields* list

class vstutils.api.fields.**SecretFileInString** (*args, **kwargs)

Field extends *FileInStringField*, but hides it’s value in admin interface.

Value must be text (not binary) and saves in model as is.

Parameters

- **media_types** (*tuple*¹³⁰, *list*¹³¹) – List of MIME types to select on the user’s side. Supported syntax using *. Default: ['*'/*']

Note: Take effect only in GUI. In API it would behave as *VSTCharField*.

class vstutils.api.fields.**TextareaField** (*args, **kwargs)

Field containing multiline string.

Note: Effective only in GUI. Works similar to *VSTCharField* in API.

class vstutils.api.fields.**UptimeField** (*args, **kwargs)

Time duration field, in seconds. May be used to compute some uptime.

Note: Effective only in GUI. Works similar to *rest_framework.fields.IntegerField* in API.

¹²² <https://docs.python.org/3.6/library/stdtypes.html#str>

¹²³ <https://docs.python.org/3.6/library/stdtypes.html#list>

¹²⁴ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹²⁵ <https://docs.python.org/3.6/library/stdtypes.html#tuple>

¹²⁶ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹²⁷ <https://docs.python.org/3.6/library/stdtypes.html#str>

¹²⁸ <https://docs.python.org/3.6/library/stdtypes.html#dict>

¹²⁹ <https://docs.python.org/3.6/library/functions.html#type>

¹³⁰ <https://docs.python.org/3.6/library/stdtypes.html#tuple>

¹³¹ <https://docs.python.org/3.6/library/stdtypes.html#list>

class `vstutils.api.fields.VSTCharField(*args, **kwargs)`

`CharField` (extends `rest_framework.fields.CharField`). This field translate any json type to string for model.

class `vstutils.api.fields.WYSIWYGField(*args, **kwargs)`

On frontend renders <https://ui.toast.com/tui-editor>. Saves data as markdown and escapes all html tags.

Parameters

escape (`bool`¹³²) – html-escape input. Enabled by default.

3.2.2 Validators

There are validation classes for fields.

class `vstutils.api.validators.FileMediaTypeValidator(extensions=None, **kwargs)`

Base Image Validation class. Validates media types.

Parameters

extensions (`typing.Union`¹³³[`typing.Tuple`¹³⁴, `typing.List`¹³⁵, `None`¹³⁶]) – Tuple or List of file extensions, that should pass the validation

Raises `rest_framework.exceptions.ValidationError`: in case file extension are not in the list

class `vstutils.api.validators.ImageBaseSizeValidator(extensions=None, **kwargs)`

Validates image size. To use this class for validating image width/height, rewrite `self.orientation` to ('height',) or ('width',) or ('height', 'width')

Raises `rest_framework.exceptions.ValidationError`: if `not(min <= (height or width) <= max)`

Parameters

extensions (`typing.Union`¹³⁷[`typing.Tuple`¹³⁸, `typing.List`¹³⁹, `None`¹⁴⁰]) –

class `vstutils.api.validators.ImageHeightValidator(extensions=None, **kwargs)`

Wrapper for `ImageBaseSizeValidator` that validates only height

Parameters

- **min_height** – minimal height of an image being validated
- **max_height** – maximal height of an image being validated
- **extensions** (`typing.Union`¹⁴¹[`typing.Tuple`¹⁴², `typing.List`¹⁴³, `None`¹⁴⁴]) –

¹³² <https://docs.python.org/3.6/library/functions.html#bool>

¹³³ <https://docs.python.org/3.6/library/typing.html#typing.Union>

¹³⁴ <https://docs.python.org/3.6/library/typing.html#typing.Tuple>

¹³⁵ <https://docs.python.org/3.6/library/typing.html#typing.List>

¹³⁶ <https://docs.python.org/3.6/library/constants.html#None>

¹³⁷ <https://docs.python.org/3.6/library/typing.html#typing.Union>

¹³⁸ <https://docs.python.org/3.6/library/typing.html#typing.Tuple>

¹³⁹ <https://docs.python.org/3.6/library/typing.html#typing.List>

¹⁴⁰ <https://docs.python.org/3.6/library/constants.html#None>

¹⁴¹ <https://docs.python.org/3.6/library/typing.html#typing.Union>

¹⁴² <https://docs.python.org/3.6/library/typing.html#typing.Tuple>

¹⁴³ <https://docs.python.org/3.6/library/typing.html#typing.List>

¹⁴⁴ <https://docs.python.org/3.6/library/constants.html#None>

class `vstutils.api.validators.ImageOpenValidator` (*extensions=None, **kwargs*)

Image validator that checks if image can be unpacked from b64 to PIL Image obj. Won't work if Pillow isn't installed.

Raises `rest_framework.exceptions.ValidationError` if PIL throws error when trying to open image

Parameters

extensions (`typing.Union`¹⁴⁵[`typing.Tuple`¹⁴⁶, `typing.List`¹⁴⁷, `None`¹⁴⁸]) –

class `vstutils.api.validators.ImageResolutionValidator` (*extensions=None, **kwargs*)

Wrapper for `ImageBaseSizeValidator` that validates both height and width

Parameters

- **min_height** – minimal height of an image being validated
- **max_height** – maximal height of an image being validated
- **min_width** – minimal width of an image being validated
- **max_width** – maximal width of an image being validated
- **extensions** (`typing.Union`¹⁴⁹[`typing.Tuple`¹⁵⁰, `typing.List`¹⁵¹, `None`¹⁵²]) –

class `vstutils.api.validators.ImageValidator` (*extensions=None, **kwargs*)

Base Image Validation class. Validates image format. Won't work if Pillow isn't installed. Base Image Validation class. Validates media types.

Parameters

extensions (`typing.Union`¹⁵³[`typing.Tuple`¹⁵⁴, `typing.List`¹⁵⁵, `None`¹⁵⁶]) –
 Tuple or List of file extensions, that should pass the validation

Raises `rest_framework.exceptions.ValidationError`: in case file extension are not in the list

property has_pillow

Check if Pillow is installed

class `vstutils.api.validators.ImageWidthValidator` (*extensions=None, **kwargs*)

Wrapper for `ImageBaseSizeValidator` that validates only width

Parameters

- **min_width** – minimal width of an image being validated
- **max_width** – maximal width of an image being validated
- **extensions** (`typing.Union`¹⁵⁷[`typing.Tuple`¹⁵⁸, `typing.List`¹⁵⁹, `None`¹⁶⁰]) –

¹⁴⁵ <https://docs.python.org/3.6/library/typing.html#typing.Union>

¹⁴⁶ <https://docs.python.org/3.6/library/typing.html#typing.Tuple>

¹⁴⁷ <https://docs.python.org/3.6/library/typing.html#typing.List>

¹⁴⁸ <https://docs.python.org/3.6/library/constants.html#None>

¹⁴⁹ <https://docs.python.org/3.6/library/typing.html#typing.Union>

¹⁵⁰ <https://docs.python.org/3.6/library/typing.html#typing.Tuple>

¹⁵¹ <https://docs.python.org/3.6/library/typing.html#typing.List>

¹⁵² <https://docs.python.org/3.6/library/constants.html#None>

¹⁵³ <https://docs.python.org/3.6/library/typing.html#typing.Union>

¹⁵⁴ <https://docs.python.org/3.6/library/typing.html#typing.Tuple>

¹⁵⁵ <https://docs.python.org/3.6/library/typing.html#typing.List>

¹⁵⁶ <https://docs.python.org/3.6/library/constants.html#None>

class `vstutils.api.validators.RegularExpressionValidator` (*regexp=None*)

Class for regular expression based validation

Raises

rest_framework.exceptions.ValidationError – in case value does not match regular expression

Parameters

regexp (`typing.Optional`¹⁶¹`[typing.Pattern`¹⁶²`])` –

class `vstutils.api.validators.UrlQueryStringValidator` (*regexp=None*)

Class for validation url query string, for example a=&b=1

Parameters

regexp (`typing.Optional`¹⁶³`[typing.Pattern`¹⁶⁴`])` –

`vstutils.api.validators.resize_image` (*img, width, height*)

Utility function to resize image proportional to specific values. Can create white margins if it's needed to satisfy required size

Parameters

- **img** (`PIL.Image`) – Pillow Image object
- **width** (`int`¹⁶⁵) – Required width
- **height** (`int`¹⁶⁶) – Required height

Returns

Pillow Image object

Return type

`PIL.Image`

`vstutils.api.validators.resize_image_from_to` (*img, limits*)

Utility function to resize image proportional to values between min and max values for each side. Can create white margins if it's needed to satisfy restrictions

Parameters

- **img** (`PIL.Image`) – Pillow Image object
- **limits** (`dict`¹⁶⁷) – Dict with min/max side restrictions like: `{'width': {'min': 300, 'max': 600}, 'height': {'min': 400, 'max': 800}}`

Returns

Pillow Image object

Return type

`PIL.Image`

¹⁵⁷ <https://docs.python.org/3.6/library/typing.html#typing.Union>

¹⁵⁸ <https://docs.python.org/3.6/library/typing.html#typing.Tuple>

¹⁵⁹ <https://docs.python.org/3.6/library/typing.html#typing.List>

¹⁶⁰ <https://docs.python.org/3.6/library/constants.html#None>

¹⁶¹ <https://docs.python.org/3.6/library/typing.html#typing.Optional>

¹⁶² <https://docs.python.org/3.6/library/typing.html#typing.Pattern>

¹⁶³ <https://docs.python.org/3.6/library/typing.html#typing.Optional>

¹⁶⁴ <https://docs.python.org/3.6/library/typing.html#typing.Pattern>

¹⁶⁵ <https://docs.python.org/3.6/library/functions.html#int>

¹⁶⁶ <https://docs.python.org/3.6/library/functions.html#int>

¹⁶⁷ <https://docs.python.org/3.6/library/stdtypes.html#dict>

3.2.3 Serializers

Default serializer classes for web-api. Read more in Django REST Framework documentation for [Serializers](#)¹⁶⁸.

class `vstutils.api.serializers.BaseSerializer` (*args, **kwargs)

Default serializer with logic to work with objects. Read more in [DRF serializer's documentation](#)¹⁶⁹ how to create Serializers and work with them.

Note: You can also setup `generated_fields` in class attribute `Meta` to get serializer with default `CharField` fields. Setup attribute `generated_field_factory` to change default fabric method.

class `vstutils.api.serializers.EmptySerializer` (*args, **kwargs)

Default serializer for empty responses. In generated GUI this means that action button won't show additional view before execution.

class `vstutils.api.serializers.VSTSerializer` (*args, **kwargs)

Default model serializer based on `rest_framework.serializers.ModelSerializer`. Read more in [DRF documentation](#)¹⁷⁰ how to create Model Serializers. This serializer matches model fields to extended set of serializer fields. List of available pairs specified in `VSTSerializer.serializer_field_mapping`. For example, to set `vstutils.api.fields.FkModelField` in serializer use `vstutils.models.fields.FkModelField` in a model.

3.2.4 Views

Default ViewSets for web-api.

class `vstutils.api.base.CopyMixin` (**kwargs)

Mixin for viewsets which adds *copy* endpoint to view.

copy (request, **kwargs)

Endpoint which copy instance with deps.

Parameters

request (`rest_framework.request.Request`) –

Return type

`vstutils.api.responses.BaseResponseClass`

copy_field_name = 'name'

Name of field which will get a prefix.

copy_prefix = 'copy-'

Value of prefix which will be added to new instance name.

copy_related: `typing.Iterable`¹⁷¹ [`str`¹⁷²] = ()

List of related names which will be copied to new instance.

¹⁶⁸ <https://www.django-rest-framework.org/api-guide/serializers/>

¹⁶⁹ <https://www.django-rest-framework.org/api-guide/serializers/#serializers>

¹⁷⁰ <https://www.django-rest-framework.org/api-guide/serializers/#modelserializer>

¹⁷¹ <https://docs.python.org/3.6/library/typing.html#typing.Iterable>

¹⁷² <https://docs.python.org/3.6/library/stdtypes.html#str>

class `vstutils.api.base.FileResponseRetrieveMixin` (***kwargs*)

ViewSet mixin for retrieving FileResponse from models with file fields data.

Example:

```
import datetime
import os
from django.db import models
from django.db.models.functions import Now
from rest_framework import permissions, fields as drf_fields
from vstutils.api.serializers import BaseSerializer, DataSerializer
from vstutils.models.decorators import register_view_action
from vstutils.models.custom_model import ListModel, FileModel, ViewCustomModel
from vstutils.api import fields, base, responses

from .cacheable import CachableModel

class TestQuerySerializer(BaseSerializer):
    test_value = drf_fields.ChoiceField(required=True, choices=("TEST1", "TEST2"))

class FileViewMixin(base.FileResponseRetrieveMixin):
    # required always
    instance_field_data = 'value'
    # required for response caching in browser
    instance_field_timestamp = 'updated'
    @register_view_action(
        methods=['get'],
        detail=False,
        query_serializer=TestQuerySerializer,
        serializer_class=DataSerializer,
        suffix='Instance'
    )
    def query_serializer_test(self, request):
        query_validated_data = self.get_query_serialized_data(request)
        return responses.HTTP_200_OK(query_validated_data)

    @register_view_action(
        methods=['get'],
        detail=False,
        query_serializer=TestQuerySerializer,
        is_list=True
    )
    def query_serializer_test_list(self, request):
        return self.list(request)
```

serializer_class_retrieve

alias of FileResponse

class `vstutils.api.base.GenericViewSet` (***kwargs*)

The base class for all views. Extends the standard features of the DRF class. Here are some of the possibilities:

- Provides model attribute instead of queryset.
- Provides to set serializers for each action separately through a dictionary `action_serializers` or attributes starting with `serializer_class_[action name]`.

- Provides to specify a serializer for lists and detail views separately.
- Optimizes the database query for GET requests, if possible, by selecting only the fields necessary for the serializer.

create_action_serializer (*args, **kwargs)

A method that implements the standard logic for actions. It relies on the passed arguments to build logic. So, if the named argument data was passed, then the serializer will be validated and saved.

Parameters

- **autosave** (*bool*¹⁷³) – Enables/disables the execution of saving by the serializer if named argument *data* passed. Enabled by default.
- **custom_data** (*dict*¹⁷⁴) – Dict with data which will passed to *validated_data* without validation.
- **serializer_class** (*None, type*¹⁷⁵ [*rest_framework.serializers.Serializer*]) – Serializer class for this execution. May be useful when request and response serializers are different.

Param

data: Default serializer class argument with serializable data. Enables validation and saving.

Param

instance: Default serializer class argument with serializable instance.

Returns

Ready serializer with default logic performed.

Return type

`rest_framework.serializers.Serializer`

get_query_serialized_data (request, query_serializer=None, raise_exception=True)

Get request query data and serialize values if *query_serializer_class* attribute exists or attribute was send.

Parameters

- **request** (`rest_framework.request.Request`) – DRF request object.
- **query_serializer** (*typing.Type*¹⁷⁶ [*rest_framework.serializers.BaseSerializer*]) – serializer class for processing parameters in *query_params*.
- **raise_exception** (*bool*¹⁷⁷) – flag that indicates whether an exception should be thrown during validation in the serializer or not.

Return type

`typing.Union`¹⁷⁸ [`dict`¹⁷⁹, `collections.OrderedDict`¹⁸⁰]

get_serializer (*args, **kwargs)

Return the serializer instance that should be used for validating and deserializing input, and for serializing output.

Provide to use `django.http.StreamingHttpResponse`¹⁸¹ as serializer init.

Parameters

- **args** (*typing.Any*¹⁸²) –
- **kwargs** (*typing.Any*¹⁸³) –

Return type

`rest_framework.serializers.BaseSerializer`

get_serializer_class()

Provides to setup serializer class for each action.

nested_allow_check()

Just raise or pass. Used for nested views for easier access checking.

class `vstutils.api.base.HistoryModelViewSet` (***kwargs*)

Default viewset like `ReadOnlyModelViewSet` but for historical data (allow to delete, but can't create and update). Inherited from `GenericViewSet`.

class `vstutils.api.base.ModelViewSet` (***kwargs*)

A viewset that provides CRUD actions under model. Inherited from `GenericViewSet`.

Variables

- **model** (`vstutils.models.BModel`) – DB model with data.
- **serializer_class** (`vstutils.api.serializers.VSTSerializer`) – Serializer for view of Model data.
- **serializer_class_one** (`vstutils.api.serializers.VSTSerializer`) – Serializer for view one instance of Model data.
- **serializer_class_[ACTION_NAME]** (`vstutils.api.serializers.VSTSerializer`) – Serializer for view of any endpoint like `.create`.

Examples:

```
from vstutils.api.base import ModelViewSet
from . import serializers as sers

class StageViewSet(ModelViewSet):
    # This is difference with DRF:
    # we use model instead of queryset
    model = sers.models.Stage
    # Serializer for list view (view for a list of Model instances
    serializer_class = sers.StageSerializer
    # Serializer for page view (view for one Model instance).
    # This property is not required, if its value is the same as `serializer_
    ↪class`.
    serializer_class_one = sers.StageSerializer
    # Allowed to set decorator to custom endpoint like this:
    # serializer_class_create - for create method
    # serializer_class_copy - for detail endpoint `copy`.
    # etc...
```

¹⁷³ <https://docs.python.org/3.6/library/functions.html#bool>

¹⁷⁴ <https://docs.python.org/3.6/library/stdtypes.html#dict>

¹⁷⁵ <https://docs.python.org/3.6/library/functions.html#type>

¹⁷⁶ <https://docs.python.org/3.6/library/typing.html#typing.Type>

¹⁷⁷ <https://docs.python.org/3.6/library/functions.html#bool>

¹⁷⁸ <https://docs.python.org/3.6/library/typing.html#typing.Union>

¹⁷⁹ <https://docs.python.org/3.6/library/stdtypes.html#dict>

¹⁸⁰ <https://docs.python.org/3.6/library/collections.html#collections.OrderedDict>

¹⁸¹ <https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.StreamingHttpResponse>

¹⁸² <https://docs.python.org/3.6/library/typing.html#typing.Any>

¹⁸³ <https://docs.python.org/3.6/library/typing.html#typing.Any>

class `vstutils.api.base.ReadOnlyModelViewSet` (***kwargs*)

Default viewset like `vstutils.api.base.ModelViewSet` for readonly models. Inherited from `GenericViewSet`.

class `vstutils.api.decorators.nested_view` (*name, arg=None, methods=None, *args, **kwargs*)

By default, DRF does not support nested views. This decorator solves this problem.

You need two or more models with nested relationship (Many-to-Many or Many-to-One) and two viewsets. Decorator nests viewset to parent viewset class and generate paths in API.

Parameters

- **name** (*str*¹⁸⁴) – Name of nested path. Also used as default name for related queryset (see *manager_name*).
- **arg** (*str*¹⁸⁵) – Name of nested primary key field.
- **view** (`vstutils.api.base.ModelViewSet`, `vstutils.api.base.HistoryModelViewSet`, `vstutils.api.base.ReadOnlyModelViewSet`) – Nested viewset class.
- **allow_append** (*bool*¹⁸⁶) – Flag for allowing to append existed instances.
- **manager_name** (*str*¹⁸⁷, *Callable*¹⁸⁸) – Name of model-object attr which contains nested queryset.
- **methods** (*list*¹⁸⁹) – List of allowed methods to nested view endpoints.
- **subs** (*list*¹⁹⁰, *None*) – List of allowed subviews or actions to nested view endpoints.
- **queryset_filters** – List of callable objects which returns filtered queryset of main.

Note: Some view methods will not call for performance reasons. This also applies to some of the class attributes that are usually initialized in the methods. For example, `.initial()` will never call. Each viewset wrapped by nested class with additional logic.

Example:

```
from vstutils.api.decorators import nested_view
from vstutils.api.base import ModelViewSet
from . import serializers as sers

class StageViewSet(ModelViewSet):
    model = sers.models.Stage
    serializer_class = sers.StageSerializer

nested_view('stages', 'id', view=StageViewSet)
class TaskViewSet(ModelViewSet):
    model = sers.models.Task
    serializer_class = sers.TaskSerializer
```

This code generates api paths:

- `/tasks/` - GET, POST
- `/tasks/{id}/` - GET, PUT, PATCH, DELETE
- `/tasks/{id}/stages/` - GET, POST
- `/tasks/{id}/stages/{stages_id}/` - GET, PUT, PATCH, DELETE

`vstutils.api.decorators.subaction(*args, **kwargs)`

Decorator which wrap object method to subaction of viewset.

Parameters

- **methods** – List of allowed HTTP-request methods. Default is `["post"]`.
- **detail** – Flag to set method execution to one instance.
- **serializer_class** – Serializer for this action.
- **permission_classes** – Tuple or list permission classes.
- **url_path** – API-path name for this action.
- **description** – Description for this action in OpenAPI.
- **multiaction** – Allow to use this action in multiactions. Works only with `vstutils.api.serializers.EmptySerializer` as response.
- **require_confirmation** – Sets whether the action must be confirmed before being executed.
- **is_list** – Mark this action as paginated list with all rules and parameters.
- **title** – Override action title.
- **icons** – Setup action icon classes.

3.2.5 Actions

Vstutils has the advanced system of working with actions. REST API works with data through verbs, which are called methods. However, to work with one or a list of entities, such actions may not be enough.

To expand the set of actions, you need to create an action that will work with some aspect of the described model. For these purposes, there is a standard `rest_framework.decorators.action()`, which can also be extended using the scheme. But for the greater convenience, there is a set of decorator objects in vstutils to eliminate the routine of writing boilerplate code.

The main philosophy for these wrappers is that the developer writes business logic without being distracted by the boilerplate code. Often, most of the errors in the code appear precisely because of the blurry look from the routine writing of the code.

```
class vstutils.api.actions.Action (detail=True, methods=None, serializer_class=<class
                                'vstutils.api.serializers.DataSerializer'>,
                                result_serializer_class=None, query_serializer=None, multi=False,
                                title=None, icons=None, is_list=False, hidden=False, **kwargs)
```

Base class of actions. Has minimal of required functionality to create an action and write only business logic. This decorator is suitable in cases where it is not possible to implement the logic using `SimpleAction` or the algorithm is much more complicated than standard CRUD.

Examples:

```
184 https://docs.python.org/3.6/library/stdtypes.html#str
185 https://docs.python.org/3.6/library/stdtypes.html#str
186 https://docs.python.org/3.6/library/functions.html#bool
187 https://docs.python.org/3.6/library/stdtypes.html#str
188 https://docs.python.org/3.6/library/typing.html#typing.Callable
189 https://docs.python.org/3.6/library/stdtypes.html#list
190 https://docs.python.org/3.6/library/stdtypes.html#list
```

```

...
from vstutils.api.fields import VSTCharField
from vstutils.api.serializers import BaseSerializer
from vstutils.api.base import ModelViewSet
from vstutils.api.actions import Action
...

class RequestSerializer(BaseSerializer):
    data_field1 = ...
    ...

class ResponseSerializer(BaseSerializer):
    detail = VSTCharField(read_only=True)

class AuthorViewSet(ModelViewSet):
    model = ...
    ...

    @Action(serializer_class=RequestSerializer, result_serializer_
↪class=ResponseSerializer, ...)
    def profile(self, request, *args, **kwargs):
        ''' Got `serializer_class` body and response with `result_
↪serializer_class`. '''
        serializer = self.get_serializer(self.get_object(), data=request.
↪data)
        serializer.is_valid(raise_exception=True)
        return {"detail": "Executed!"}

```

Parameters

- **detail** (`bool`¹⁹¹) – Flag indicating which type of action is used: on a list or on a single entity. Affects where this action will be displayed - on a detailed record or on a list of records.
- **methods** (`typing.List`¹⁹²`[str`¹⁹³`])` – List of available HTTP-methods for this action. Default has only *POST* method.
- **serializer_class** (`typing.Type`¹⁹⁴`[rest_framework.serializers.Serializer]`) – Request body serializer. Also used for default response.
- **result_serializer_class** (`typing.Type`¹⁹⁵`[rest_framework.serializers.Serializer]`) – Response body serializer. Required, when request and response has different set of fields.
- **query_serializer** (`typing.Type`¹⁹⁶`[rest_framework.serializers.Serializer]`) – GET-request query data serializer. It is used when it is necessary to get valid data in the query data of a GET-request and cast it to the required type.
- **multi** (`bool`¹⁹⁷) – Used only with non-GET requests and notify GUI, that this action should be rendered over the selected list items.
- **title** (`str`¹⁹⁸) – Title for action in UI. For non-GET actions, title is generated from method's name.
- **icons** (`typing.Union`¹⁹⁹`[str`²⁰⁰`, typing.Iterable`²⁰¹`])` – List of icons for UI button.
- **is_list** (`bool`²⁰²) – Flag indicating whether the action type is a list or a single entity. Typically used with GET actions.

- **kwargs** – Set of named arguments for `rest_framework.decorators.action()`.
- **hidden** (`bool`²⁰³) –

class `vstutils.api.actions.EmptyAction` (***kwargs*)

In this case, actions on an object do not require any data and manipulations with them. For such cases, there is a standard method that allows you to simplify the scheme and code to work just with the object.

Optionally, you can also overload the response serializer to notify the interface about the format of the returned data.

Examples:

```
...
from vstutils.api.fields import RedirectIntegerField
from vstutils.api.serializers import BaseSerializer
from vstutils.api.base import ModelViewSet
from vstutils.api.actions import EmptyAction
...

class ResponseSerializer(BaseSerializer):
    id = RedirectIntegerField(operation_name='sync_history')

class AuthorViewSet(ModelViewSet):
    model = ...
    ...

    @EmptyAction(result_serializer_class=ResponseSerializer, ...)
    def sync_data(self, request, *args, **kwargs):
        ''' Example of action which get object, sync data and redirect_
↪user to another view. '''
        sync_id = self.get_object().sync().id
        return {"id": sync_id}
```

```
...
from django.http.response import FileResponse
from vstutils.api.base import ModelViewSet
from vstutils.api.actions import EmptyAction
...

class AuthorViewSet(ModelViewSet):
    model = ...
    ...

    @EmptyAction(result_serializer_class=ResponseSerializer, ...)
```

(continues on next page)

¹⁹¹ <https://docs.python.org/3.6/library/functions.html#bool>
¹⁹² <https://docs.python.org/3.6/library/typing.html#typing.List>
¹⁹³ <https://docs.python.org/3.6/library/stdtypes.html#str>
¹⁹⁴ <https://docs.python.org/3.6/library/typing.html#typing.Type>
¹⁹⁵ <https://docs.python.org/3.6/library/typing.html#typing.Type>
¹⁹⁶ <https://docs.python.org/3.6/library/typing.html#typing.Type>
¹⁹⁷ <https://docs.python.org/3.6/library/functions.html#bool>
¹⁹⁸ <https://docs.python.org/3.6/library/stdtypes.html#str>
¹⁹⁹ <https://docs.python.org/3.6/library/typing.html#typing.Union>
²⁰⁰ <https://docs.python.org/3.6/library/stdtypes.html#str>
²⁰¹ <https://docs.python.org/3.6/library/typing.html#typing.Iterable>
²⁰² <https://docs.python.org/3.6/library/functions.html#bool>
²⁰³ <https://docs.python.org/3.6/library/functions.html#bool>

(continued from previous page)

```

def resume(self, request, *args, **kwargs):
    ''' Example of action which response with generated resume in pdf.
    → '''
    instance = self.get_object()

    return FileResponse(
        streaming_content=instance.get_pdf(),
        as_attachment=True,
        filename=f'{instance.last_name}_{instance.first_name}_resume.
    → pdf'
    )

```

class `vstutils.api.actions.SimpleAction(*args, **kwargs)`

The idea of this decorator is to get the full CRUD for the instance in a minimum of steps. The instance is the object that was returned from the method being decorated. The whole mechanism is very similar to the standard property decorator, with a description of a getter, setter, and deleter.

If you're going to create an entry point for working with a single object, then you do not need to define methods. The presence of a getter, setter, and deleter will determine which methods will be available.

In the official documentation of Django, an example is given with moving data that is not important for authorization to the Profile model. To work with such data that is outside the main model, there is this action object, which implements the basic logic in the most automated way.

It covers most of the tasks for working with such data. By default, it has a GET method instead of POST. Also, for better organization of the code, it allows you to change the methods that will be called when modifying or deleting data.

When assigning an action on an object, the list of methods is also filled with the necessary ones.

Examples:

```

...
from vstutils.api.fields import PhoneField
from vstutils.api.serializers import BaseSerializer
from vstutils.api.base import ModelViewSet
from vstutils.api.actions import Action
...

class ProfileSerializer(BaseSerializer):
    phone = PhoneField()

class AuthorViewSet(ModelViewSet):
    model = ...
    ...

    @SimpleAction(serializer_class=ProfileSerializer, ...)
    def profile(self, request, *args, **kwargs):
        ''' Get profile data to work. '''
        return self.get_object().profile

    @profile.setter
    def profile(self, instance, request, serializer, *args, **kwargs):
        instance.save(update_fields=['phone'])

    @profile.deleter

```

(continues on next page)

(continued from previous page)

```
def profile(self, instance, request, serializer, *args, **kwargs):
    instance.phone = ''
    instance.save(update_fields=['phone'])
```

3.2.6 Filtersets

For greater development convenience, the framework provides additional classes and functions for filtering elements by fields.

class `vstutils.api.filters.DefaultIDFilter` (*data=None, queryset=None, *, request=None, prefix=None*)

Basic filterset to search by id. Provides a search for multiple values separated by commas. Uses `extra_filter()` in fields.

class `vstutils.api.filters.DefaultNameFilter` (*data=None, queryset=None, *, request=None, prefix=None*)

Basic filterset to search by part of name. Uses *LIKE* DB condition by `name_filter()`.

class `vstutils.api.filters.FkFilterHandler` (*related_pk='id', related_name='name', pk_handler=<class 'int'>*)

Simple handler for filtering by relational fields.

Parameters

- **related_pk** (`str`²⁰⁴) – Field name of related model’s primary key. Default is ‘id’.
- **related_name** (`str`²⁰⁵) – Field name of related model’s charfield. Default is ‘name’.
- **pk_handler** (`typing.Callable`²⁰⁶) – Changes handler for checking value before search. Sends “0” if handler falls. Default is ‘int()’.

Example:

```
class CustomFilterSet(filters.FilterSet):
    author = CharFilter(method=vst_filters.FkFilterHandler(related_pk='pk',
↵related_name='email'))
```

Where `author` is `ForeignKey` to `User` and you want to search by primary key and email.

`vstutils.api.filters.extra_filter` (*queryset, field, value*)

Method for searching values in a comma-separated list.

Parameters

- **queryset** (`django.db.models.query.QuerySet`²⁰⁷) – model queryset for filtration.
- **field** (`str`²⁰⁸) – field name in `FilterSet`. Also supports `__not` suffix.
- **value** (`str`²⁰⁹) – comma separated list of searching values.

Returns

filtered queryset.

²⁰⁴ <https://docs.python.org/3.6/library/stdtypes.html#str>

²⁰⁵ <https://docs.python.org/3.6/library/stdtypes.html#str>

²⁰⁶ <https://docs.python.org/3.6/library/typing.html#typing.Callable>

Return type`django.db.models.query.QuerySet`²¹⁰`vstutils.api.filters.name_filter(queryset, field, value)`

Method for searching by part of name. Uses *LIKE* DB condition or *contains* qs-expression.

Parameters

- **queryset** (`django.db.models.query.QuerySet`²¹¹) – model queryset for filtration.
- **field** (`str`²¹²) – field name in FilterSet. Also supports `__not` suffix.
- **value** (`str`²¹³) – searching part of name.

Returns

filtered queryset.

Return type`django.db.models.query.QuerySet`²¹⁴

3.2.7 Responses

DRF provides a standard set of variables whose names correspond to the human-readable name of the HTTP code. For convenience, we have dynamically wrapped it in a set of classes that have appropriate names and additionally provides following capabilities:

- String responses are wrapped in json like { "detail": "string response" }.
- Attribute timings are kept for further processing in middleware.
- Status code is set by class name (e.g. `HTTP_200_OK` or `Response200` has code 200).

All classes inherit from:

```
class vstutils.api.responses.BaseResponseClass (*args, **kwargs)
```

API response class with default status code.

Variables

- **status_code** (`int`²¹⁵) – HTTP status code.
- **timings** (`int`²¹⁶, `None`) – Response timings.

Parameters

timings – Response timings.

²⁰⁷ <https://docs.djangoproject.com/en/4.2/ref/models/querysets/#django.db.models.query.QuerySet>

²⁰⁸ <https://docs.python.org/3.6/library/stdtypes.html#str>

²⁰⁹ <https://docs.python.org/3.6/library/stdtypes.html#str>

²¹⁰ <https://docs.djangoproject.com/en/4.2/ref/models/querysets/#django.db.models.query.QuerySet>

²¹¹ <https://docs.djangoproject.com/en/4.2/ref/models/querysets/#django.db.models.query.QuerySet>

²¹² <https://docs.python.org/3.6/library/stdtypes.html#str>

²¹³ <https://docs.python.org/3.6/library/stdtypes.html#str>

²¹⁴ <https://docs.djangoproject.com/en/4.2/ref/models/querysets/#django.db.models.query.QuerySet>

²¹⁵ <https://docs.python.org/3.6/library/functions.html#int>

²¹⁶ <https://docs.python.org/3.6/library/functions.html#int>

3.2.8 Middleware

By default, Django [supposes](#)²¹⁷ that a developer creates Middleware class manually, but it's often a routine. The `vstutils` library offers a convenient request handler class for elegant OOP development. Middleware is used to process incoming requests and send responses before they reach final destination.

class `vstutils.middleware.BaseMiddleware` (*get_response*)

Middleware base class for handling:

- Incoming requests by `BaseMiddleware.request_handler()`;
- Outgoing response before any calling on server by `BaseMiddleware.get_response_handler()`;
- Outgoing responses by `BaseMiddleware.handler()`.

Middleware must be added to `MIDDLEWARE` list in settings.

Example:

```
from vstutils.middleware import BaseMiddleware
from django.http import HttpResponse

class CustomMiddleware(BaseMiddleware):
    def request_handler(self, request):
        # Add header to request
        request.headers['User-Agent'] = 'Mozilla/5.0'
        return request

    def get_response_handler(self, request):
        if not request.user.is_stuff:
            # Return 403 HTTP status for non-stuff users.
            # This request never gets in any view.
            return HttpResponse(
                "Access denied!",
                content_type="text/plain",
                status_code=403
            )
        return super().get_response_handler(request)

    def handler(self, request, response):
        # Add header to response
        response['Custom-Header'] = 'Some value'
        return response
```

Parameters

get_response (`typing.Callable`²¹⁸) –

get_response_handler (*request*)

Entrypoint for breaking or continuing request handling. This function must return `django.http.HttpResponse` object or result of parent class calling.

Since the release of 5.3, it has been possible to write this method as asynchronous. This should be used in cases where the middleware makes queries to the database or cache. However, such a middleware should be excluded from bulk requests.

²¹⁷ <https://docs.djangoproject.com/en/4.1/topics/http/middleware/#writing-your-own-middleware>

Warning: Never do asynchronous middleware in dependent chains. They are designed to send independent requests to external sources.

Set `async_capable` to `True` and `sync_capable` to `False` for such middleware.

Parameters

request (`django.http.HttpRequest`²¹⁹) – HTTP-request object which is wrapped from client request.

Return type

`django.http.HttpResponse`²²⁰

handler (*request, response*)

The response handler. Method to process responses.

Parameters

- **request** (`django.http.HttpRequest`²²¹) – HTTP-request object.
- **response** (`django.http.HttpResponse`²²²) – HTTP-response object which will be sended to client.

Returns

Handled response object.

Return type

`django.http.HttpResponse`²²³

request_handler (*request*)

The request handler. Called before request is handled by a view.

Parameters

request (`django.http.HttpRequest`²²⁴) – HTTP-request object which is wrapped from client request.

Returns

Handled request object.

Return type

`django.http.HttpRequest`²²⁵

²¹⁸ <https://docs.python.org/3.6/library/typing.html#typing.Callable>

²¹⁹ <https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpRequest>

²²⁰ <https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpResponse>

²²¹ <https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpRequest>

²²² <https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpResponse>

²²³ <https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpResponse>

²²⁴ <https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpRequest>

²²⁵ <https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpRequest>

3.2.9 Filter Backends

Filter Backends²²⁶ are used to modify model queryset. To create custom filter backend to, (i.g. annotate model queryset), you should inherit from `vstutils.api.filter_backends.VSTFilterBackend` and override `vstutils.api.filter_backends.VSTFilterBackend.filter_queryset()` and in some cases `vstutils.api.filter_backends.VSTFilterBackend.get_schema_fields()`.

class `vstutils.api.filter_backends.DeepViewFilterBackend`

Backend that filters queryset by column from `deep_parent_field` property of the model. Value for filtering must be provided in query param `__deep_parent`.

If param is missing then no filtering is applied.

If param is empty value (`/?__deep_parent=`) then objects with no parent (the value of the field whose name is stored in the property `deep_parent_field` of the model is `None`) returned.

This filter backend and nested view is automatically added when model has `deep_parent_field` property.

Example:

```
from django.db import models
from vstutils.models import BModel

class DeepNestedModel(BModel):
    name = models.CharField(max_length=10)
    parent = models.ForeignKey('self', null=True, default=None, on_
    ↪delete=models.CASCADE)

    deep_parent_field = 'parent'
    deep_parent_allow_append = True

    class Meta:
        default_related_name = 'deepnested'
```

In example above if we add this model under path `'deep'`, following views will be created: `/deep/` and `/deep/{id}/deepnested/`.

Filter backend can be used as `/deep/?__deep_parent=1` and will return all `DeepNestedModel` objects whose parent's primary key is `1`.

You can also use generic DRF views, for that you still must set `deep_parent_field` to your model and manually add `DeepViewFilterBackend` to `filter_backends`²²⁷ list.

class `vstutils.api.filter_backends.HideHiddenFilterBackend`

Filter Backend that hides all objects with `hidden=True` from the queryset

filter_queryset (*request, queryset, view*)

Clear objects with hidden attr from queryset.

class `vstutils.api.filter_backends.SelectRelatedFilterBackend`

Filter Backend that will automatically call `prefetch_related` and `select_related` on all relations in queryset.

filter_queryset (*request, queryset, view*)

Select+prefetch related in queryset.

class `vstutils.api.filter_backends.VSTFilterBackend`

A base filter backend class to be inherited from. Example:

²²⁶ <https://www.django-rest-framework.org/api-guide/filtering/#djangofilterbackend>

²²⁷ <https://www.django-rest-framework.org/api-guide/filtering/#djangofilterbackend>

```

from django.utils import timezone
from django.db.models import Value, DateTimeField

from vstutils.api.filter_backends import VSTFilterBackend

class CurrentTimeFilterBackend(VSTFilterBackend):
    def filter_queryset(self, request, queryset, view):
        return queryset.annotate(current_time=Value(timezone.now()),
↪output_field=DateTimeField()))

```

In this example Filter Backend annotates time in current timezone to any connected model's queryset.

In some cases it may be necessary to provide a parameter from a query of request. To define this parameter in the schema, you must overload the `get_schema_operation_parameters` function and specify a list of parameters to use.

Example:

```

from django.utils import timezone
from django.db.models import Value, DateTimeField

from vstutils.api.filter_backends import VSTFilterBackend

class ConstantCurrentTimeForQueryFilterBackend(VSTFilterBackend):
    query_param = 'constant'

    def filter_queryset(self, request, queryset, view):
        if self.query_param in request.query_params and request.query_
↪params[self.query_param]:
            queryset = queryset.annotate(**{
                request.query_params[self.query_param]: Value(timezone.
↪now(), output_field=DateTimeField())
            })
        return queryset

    def get_schema_operation_parameters(self, view):
        return [
            {
                "name": self.query_param,
                "required": False,
                "in": openapi.IN_QUERY,
                "description": "Annotate value to queryset",
                "schema": {
                    "type": openapi.TYPE_STRING,
                }
            },
        ]

```

In this example Filter Backend annotates time in current timezone to any connected model's queryset with field name from query *constant*.

get_schema_operation_parameters (view)

You can also make the filter controls available to the schema autogeneration that REST framework provides, by implementing this method. The method should return a list of OpenAPI schema mappings.

3.3 Celery

Celery is a distributed task queue. It's used to execute some actions asynchronously in a separate worker. For more details on Celery, check it's official [docs](https://docs.celeryproject.org/en/stable/)²²⁸. For Celery related vstutils features to work, you need to specify `[rpc]` and `[worker]` sections in `settings.ini`. Also you need to include extra `[rpc]` requirements.

3.3.1 Tasks

class `vstutils.tasks.TaskClass`

Wrapper for Celery BaseTask class. Usage is same as Celery standard class, but you can execute task without creating instance with `TaskClass.do()` method.

Example:

```
from vstutils.environment import get_celery_app
from vstutils.tasks import TaskClass

app = get_celery_app()

class Foo(TaskClass):
    def run(*args, **kwargs):
        return 'Foo task has been executed'

app.register_task(Foo())
```

Now you can call your task with various methods:

- by executing `Foo.do(*args, **kwargs)`
- get registered task instance like that - `app.tasks['full_path.to.task.class.Foo']`

Also you can make your registered task periodic, by adding it to `CELERY_BEAT_SCHEDULE` in `settings.py`:

```
CELERY_BEAT_SCHEDULE = {
    'foo-execute-every-month': {
        'task': 'full_path.to.task.class.Foo',
        'schedule': crontab(day_of_month=1),
    },
}
```

classmethod `do(*args, **kwargs)`

Method which send signal to celery for start remote task execution. All arguments will passed to the task `TaskClass.run()` method.

Return type

`celery.result.AsyncResult`

property `name`

property for proper Celery task execution, needed for `TaskClass.do()` method to work

run `(*args, **kwargs)`

The body of the task executed by workers.

²²⁸ <https://docs.celeryproject.org/en/stable/>

3.4 Endpoint

Endpoint view has two purposes: bulk requests execution and providing OpenAPI schema.

Endpoint url is `{API_URL}/endpoint/`, for example value with default settings is `/api/endpoint/`.

API_URL can be changed in `settings.py`.

class `vstutils.api.endpoint.EndpointViewSet` (***kwargs*)

Default API-endpoint viewset.

get (*request*)

Returns response with swagger ui or openapi json schema if `?format=openapi`

Parameters

request (`vstutils.api.endpoint.BulkRequestType`) –

Return type

`django.http.response.HttpResponse`

get_client (*request*)

Returns test client and guarantees that if bulk request comes authenticated than test client will be authenticated with the same user

Parameters

request (`vstutils.api.endpoint.BulkRequestType`) –

Return type

`vstutils.api.endpoint.BulkClient`

get_serializer (**args, **kwargs*)

Return the serializer instance that should be used for validating and deserializing input, and for serializing output.

Return type

`vstutils.api.endpoint.OperationSerializer`

get_serializer_context (*context*)

Extra context provided to the serializer class.

Return type

`dict`²²⁹

operate (*operation_data, context*)

Method used to handle one operation and return result of it

Parameters

• **operation_data** (`typing.Dict`²³⁰) –

• **context** (`typing.Dict`²³¹) –

Return type

`typing.Tuple`²³²[`typing.Dict`²³³, `typing.SupportsFloat`²³⁴]

post (*request*)

Execute transactional bulk request

Parameters

request (`vstutils.api.endpoint.BulkRequestType`) –

Return type*vstutils.api.responses.BaseResponseClass***put** (*request*, *allow_fail=True*)

Execute non transaction bulk request

Parameters**request** (*vstutils.api.endpoint.BulkRequestType*) –**Return type***vstutils.api.responses.BaseResponseClass***serializer_class**

One operation serializer class.

alias of `OperationSerializer`**versioning_class**alias of `QueryParameterVersioning`

3.4.1 Bulk requests

Bulk request allows you send multiple requests to api at once, it accepts json list of operations.

Method	Transactional (all operations in one transaction)	Synchronous (operations executed one by one in given order)
PUT /{API_URL}/ endpoint/	NO	YES
POST /{API_URL}/ endpoint/	YES	YES
PATCH /{API_URL}/ endpoint/	NO	NO

Parameters of one operation (required parameter marked by *):

- `method*` - http method of request
- `path*` - path of request, can be `str` or `list`
- `data` - data to send
- `query` - query parameters as `str`
- `let` - string with name of variable (used for access to response result in templates)
- `headers` - dict with headers which will be sent, names of headers must follow [CGI specification](#)²³⁵ (e.g., `CONTENT_TYPE`, `GATEWAY_INTERFACE`, `HTTP_*`).
- `version` - `str` with specified version of api, if not provided then `VST_API_VERSION` will be used

In any request parameter you can insert result value of previous operations (`<<{OPERATION_NUMBER or LET_VALUE}[path][to][value]>>`), for example:

²²⁹ <https://docs.python.org/3.6/library/stdtypes.html#dict>

²³⁰ <https://docs.python.org/3.6/library/typing.html#typing.Dict>

²³¹ <https://docs.python.org/3.6/library/typing.html#typing.Dict>

²³² <https://docs.python.org/3.6/library/typing.html#typing.Tuple>

²³³ <https://docs.python.org/3.6/library/typing.html#typing.Dict>

²³⁴ <https://docs.python.org/3.6/library/typing.html#typing.SupportsFloat>

²³⁵ <https://www.w3.org/CGI/>


```
[
  {"method": "post", "path": "user", "data": {"name": "User 1"}},
  {"method": "delete", "version": "v2", "path": ["user", "<<0[data][id]>>"]}
]
```

Result of bulk request is json list of objects for operation:

- `method` - http method
- `path` - path of request, always str
- `data` - data that needs to be sent
- `status` - response status code

Transactional bulk request returns 502 `BAG GATEWAY` and does rollback after first failed request.

Warning: If you send non-transactional bulk request, you will get 200 status and must validate statuses on each operation responses.

3.4.2 OpenAPI schema

Request on `GET /{API_URL}/endpoint/` returns Swagger UI.

Request on `GET /{API_URL}/endpoint/?format=openapi` returns OpenAPI schema in json format. Also you can specify required version of schema using `version` query parameter (e.g., `GET /{API_URL}/endpoint/?format=openapi&version=v2`).

To change the schema after generating and before sending to user use hooks. Define one or more function, each taking 2 named arguments:

- `request` - user request object.
- `schema` - ordered dict with OpenAPI schema.

Note: Sometimes hooks may raise an exception; in order to keep a chain of data modification, such exceptions are handled. The changes made to the schema before the exception however, are saved.

Example hook:

```
def hook_add_username_to_guiname(request, schema):
    schema['info']['title'] = f"{request.username} - {schema['info']['title']}"
```

To connect hook(s) to your app add function import name to the `OPENAPI_HOOKS` list in `settings.py`

```
OPENAPI_HOOKS = [
    '{{appName}}.openapi.hook_add_username_to_guiname',
]
```

3.5 Testing Framework

VST Utils Framework includes a helper in base test case class and improves support for making API requests. That means if you want make bulk request to endpoint you don't need create and init test client, but just need to call:

```
endpoint_results = self.bulk([
    # list of endpoint requests
])
```

3.5.1 Creating test case

test.py module contains test case classes based on `vstutils.tests.BaseTestCase`. At the moment, we officially support two styles of writing tests: classic and simple query wrappers with run check and runtime optimized bulk queries with manual value checking.

3.5.2 Simple example with classic tests

For example, if you have api endpoint like `/api/v1/project/` and model Project you can write test case like this:

```
from vstutils.tests import BaseTestCase

class ProjectTestCase(BaseTestCase):
    def setUp(self):
        super(ProjectTestCase, self).setUp()
        # init demo project
        self.initial_project = self.get_model_class('project.Test').objects.
        ↪ create(name="Test")

    def tearDown(self):
        super(ProjectTestCase, self).tearDown()
        # remove it after test
        self.initial_project.delete()

    def test_project_endpoint(self):
        # Test checks that api returns valid values
        self.list_test('/api/v1/project/', 1)
        self.details_test(
            ["project", self.initial_project.id],
            name=self.initial_project.name
        )
        # Try to create new projects and check list endpoint
        test_data = [
            {"name": f"TestProject{i}"}
            for i in range(2)
        ]
        id_list = self.mass_create("/api/v1/project/", test_data, 'name')
        self.list_test('/api/v1/project/', 1 + len(id_list))
```

This example demonstrates functionality of default test case class. Default projects are initialized for the fastest and most efficient result. We recommend to divide tests for different entities into different classes. This example demonstrate classic style of testing, but you can use bulks in your test cases.

3.5.3 Bulk requests in tests

Bulk query system is well suited for testing and executing valid queries. Previous example could be rewritten as follows:

```
from vstutils.tests import BaseTestCase

class ProjectTestCase(BaseTestCase):
    def setUp(self):
        super(ProjectTestCase, self).setUp()
        # init demo project
        self.initial_project = self.get_model_class('project.Test').objects.  
→create(name="Test")

    def tearDown(self):
        super(ProjectTestCase, self).tearDown()
        # remove it after test
        self.initial_project.delete()

    def test_project_endpoint(self):
        test_data = [
            {"name": f"TestProject{i}"}
            for i in range(2)
        ]
        bulk_data = [
            {"method": "get", "path": ["project"]},
            {"method": "get", "path": ["project", self.initial_project.id]}
        ]
        bulk_data += [
            {"method": "post", "path": ["project"], "data": i}
            for i in test_data
        ]
        bulk_data.append(
            {"method": "get", "path": ["project"]}
        )
        results = self.bulk_transactional(bulk_data)

        self.assertEqual(results[0]['status'], 200)
        self.assertEqual(results[0]['data']['count'], 1)
        self.assertEqual(results[1]['status'], 200)
        self.assertEqual(results[1]['data']['name'], self.initial_project.name)

        for pos, result in enumerate(results[2:-1]):
            self.assertEqual(result['status'], 201)
            self.assertEqual(result['data']['name'], test_data[pos]['name'])

        self.assertEqual(results[-1]['status'], 200)
        self.assertEqual(results[-1]['data']['count'], 1 + len(test_data))
```

In this case, you have more code, but your tests are closer to GUI workflow, because vstutils-projects uses `/api/endpoint/` for requests. Either way, bulk queries are much faster due to optimization; Testcase execution time is less comparing to non-bulk requests.

3.5.4 Test case API

class `vstutils.tests.BaseTestCase` (*methodName='runTest'*)

Main test case class extends `django.test.TestCase`²³⁶.

assertCheckDict (*first, second, msg=None*)

Fail if the two fields in dicts are unequal as determined by the '==' operator. Checks if first not contains or not equal field in second

Parameters

- **first** (`typing.Dict`²³⁷) –
- **second** (`typing.Dict`²³⁸) –
- **msg** (`str`²³⁹) –

assertCount (*iterable, count, msg=None*)

Calls `len()`²⁴⁰ over iterable and checks equality with count.

Parameters

- **iterable** (`typing.Sized`²⁴¹) – any iterable object which could be send to `len()`²⁴².
- **count** (`int`²⁴³) – expected result.
- **msg** (`typing.Any`²⁴⁴) – error message

assertRCode (*resp, code=200, *additional_info*)

Fail if response code is not equal. Message is response body.

Parameters

- **resp** (`django.http.HttpResponse`²⁴⁵) – response object
- **code** (`int`²⁴⁶) – expected code

bulk (*data, code=200, **kwargs*)

Makes non-transactional bulk request and asserts status code (default is 200)

Parameters

- **data** (`typing.Union`²⁴⁷[`typing.List`²⁴⁸[`typing.Dict`²⁴⁹[`str`²⁵⁰, `typing.Any`²⁵¹]], `str`²⁵², `bytes`²⁵³, `bytearray`²⁵⁴]) – request data
- **code** (`int`²⁵⁵) – http status to assert
- **kwargs** – named arguments for `get_result()`

Return type

`typing.Union`²⁵⁶[`typing.List`²⁵⁷[`typing.Dict`²⁵⁸[`str`²⁵⁹, `typing.Any`²⁶⁰]], `str`²⁶¹, `bytes`²⁶², `bytearray`²⁶³, `typing.Dict`²⁶⁴, `typing.Sequence`²⁶⁵[`typing.Union`²⁶⁶[`typing.List`²⁶⁷[`typing.Dict`²⁶⁸[`str`²⁶⁹, `typing.Any`²⁷⁰]], `str`²⁷¹, `bytes`²⁷², `bytearray`²⁷³]]]

Returns

bulk response

bulk_transactional (*data, code=200, **kwargs*)

Make transactional bulk request and assert status code (default is 200)

Parameters

- **data** (`typing.Union274[typing.List275[typing.Dict276[str277, typing.Any278]], str279, bytes280, bytearray281])` – request data
- **code** (`int282`) – http status to assert
- **kwargs** – named arguments for `get_result()`

Return type

`typing.Union283[typing.List284[typing.Dict285[str286, typing.Any287]], str288, bytes289, bytearray290, typing.Dict291, typing.Sequence292[typing.Union293[typing.List294[typing.Dict295[str296, typing.Any297]], str298, bytes299, bytearray300]]]`

Returns

bulk response

call_registration (`data, **kwargs`)

Function for calling registration. Just got form data and headers.

Parameters

- **data** (`dict301`) – Registration form data.
- **kwargs** – named arguments with request headers.

details_test (`url, **kwargs`)

Test for get details of model. If you setup additional named arguments, the method check their equality with response data. Uses `get_result()` method.

Parameters

- **url** – url to detail record. For example: `/api/v1/project/1/` (where 1 is uniq id of project). You can use `get_url()` for building url.
- **kwargs** – params that's should be checked (key - field name, value - field value).

endpoint_call (`data=None, method='get', code=200, **kwargs`)

Makes request to endpoint and asserts response status code if specified (default is 200). Uses `get_result()` method for execution.

Parameters

- **data** (`typing.Union302[typing.List303[typing.Dict304[str305, typing.Any306]], str307, bytes308, bytearray309])` – request data
- **method** (`str310`) – http request method
- **code** (`int311`) – http status to assert
- **query** – dict with query data (working only with `get`)

Return type

`typing.Union312[typing.List313[typing.Dict314[str315, typing.Any316]], str317, bytes318, bytearray319, typing.Dict320, typing.Sequence321[typing.Union322[typing.List323[typing.Dict324[str325, typing.Any326]], str327, bytes328, bytearray329]]]`

Returns

bulk response

endpoint_schema (`**kwargs`)

Make request to schema. Returns dict with swagger data.

Parameters

version – API version for schema parser.

get_count (*model*, ***kwargs*)

Simple wrapper over `get_model_filter()` which returns counter of items.

Parameters

- **model** (*str*³³⁰, `django.db.models.Model`³³¹) – string which contains model name (if attribute `model` is set to the test case class), module import, `app.ModelName` or `django.db.models.Model`³³².
- **kwargs** – named arguments to `django.db.models.query.QuerySet.filter()`³³³.

Returns

number of instances in database.

Return type

`int`³³⁴

get_model_class (*model*)

Getting model class by string or return model arg.

Parameters

model (*str*³³⁵, `django.db.models.Model`³³⁶) – string which contains model name (if attribute `model` is set to the test case class), module import, `app.ModelName` or `django.db.models.Model`³³⁷.

Returns

Model class.

Return type

`django.db.models.Model`³³⁸

get_model_filter (*model*, ***kwargs*)

Simple wrapper over `get_model_class()` which returns filtered queryset from model.

Parameters

- **model** (*str*³³⁹, `django.db.models.Model`³⁴⁰) – string which contains model name (if attribute `model` is set to the test case class), module import, `app.ModelName` or `django.db.models.Model`³⁴¹.
- **kwargs** – named arguments to `django.db.models.query.QuerySet.filter()`³⁴².

Return type

`django.db.models.query.QuerySet`³⁴³

get_result (*rtype*, *url*, *code=None*, **args*, ***kwargs*)

Executes and tests response code on request with returning parsed result of request. The method uses the following procedure:

- Test client authorization (with user which creates in `setUp()`).
- Executing a request (sending args and kwargs to request method).
- Parsing the result (converts json string to python-object).
- Checking the http status code with `assertRCode()` (if you have not specified it, the code will be selected in accordance with the request method from the standard set `std_codes`).

- Logout client.
- Return parsed result.

Parameters

- **rtype** – request type (methods from Client cls): get, post etc.
- **url** – requested url string or tuple for `get_url()`. You can use `get_url()` for url building or setup it as full string.
- **code** (`int`³⁴⁴) – expected return code from request.
- **relogin** – execute force login and logout on each call. Default is `True`.
- **args** – extra-args for Client class request method.
- **kwargs** – extra-kwargs for Client class request method.

Return type

```
typing.Union345[typing.List346[typing.Dict347[str348, typing.Any349]], str350, bytes351, bytearray352, typing.Dict353, typing.Sequence354[typing.Union355[typing.List356[typing.Dict357[str358, typing.Any359]], str360, bytes361, bytearray362]]]
```

Returns

result of request.

`get_url (*items)`

Function for creating url path based on `VST_API_URL` and `VST_API_VERSION` settings. Without arguments returns path to default version of api.

Return type

```
str363
```

Returns

string like `/api/v1/.../.../` where `...` is args of function.

`list_test (url, count)`

Test for get list of models. Checks only list count. Uses `get_result()` method.

Parameters

- **url** – url to abstract layer. For example: `/api/v1/project/`. You can use `get_url()` for building url.
- **count** – count of objects in DB.

`models = None`

Attribute with default project models module.

`classmethod patch (*args, **kwargs)`

Simple `unittest.mock.patch()`³⁶⁴ class-method wrapper.

Return type

```
typing.ContextManager365[unittest.mock.Mock366]
```

`classmethod patch_field_default (model, field_name, value)`

This method helps to path default value in the model's field. It's very useful for `DateTime` fields where `django.utils.timezone.now()`³⁶⁷ is used in defaults.

Parameters

- **model** (`django.db.models.base.Model`) –
- **field_name** (`str`³⁶⁸) –
- **value** (`typing.Any`³⁶⁹) –

Return type

`typing.ContextManager`³⁷⁰[`unittest.mock.Mock`³⁷¹]

random_name ()

Simple function which returns uuid1 string.

Return type

`str`³⁷²

std_codes: `typing.Dict`³⁷³[`str`³⁷⁴, `int`³⁷⁵] = {'delete': 204, 'get': 200, 'patch': 200, 'post': 201}

Default http status codes for different http methods. Uses in `get_result()`

class user_as (*testcase, user*)

Context for execute bulk or something as user. The context manager overrides `self.user` in `TestCase` and revert this changes on exit.

Parameters

user (`django.contrib.auth.models.AbstractUser`³⁷⁶) – new user object for execution.

3.5 Testing Framework

3.6 Utils

This is tested set of development utilities. Utilities include a collection of code that will be useful in one way or another for developing the application. Vstutils uses mosts of these functions under the hood.

class vstutils.utils.**BaseEnum** (*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

BaseEnum extends *Enum* class and used to create enum-like objects that can be used in django serializers or django models.

Example:

```
from vstutils.models import BModel

class ItemClasses(BaseEnum):
    FIRST = BaseEnum.SAME
    SECOND = BaseEnum.SAME
    THIRD = BaseEnum.SAME

class MyDjangoModel(BModel):
    item_class = models.CharField(max_length=ItemClasses.max_len,
    ↪choices=ItemClasses.to_choices())

    @property
    def is_second(self):
        # Function check is item has second class of instance
        return ItemClasses.SECOND.is_equal(self.item_class)
```

Note: For special cases, when value must be in lower or upper case, you can setup value as `BaseEnum.LOWER`` or ``BaseEnum.UPPER`. But in default cases we recommend use `BaseEnum.SAME` for memory optimization.

class vstutils.utils.**BaseVstObject**

Default mixin-class for custom objects which needed to get settings and cache.

classmethod **get_django_settings** (*name, default=None*)

Get params from Django settings.

Parameters

- **name** (*str*³⁷⁷) – name of param
- **default** (*object*³⁷⁸) – default value of param

Returns

Param from Django settings or default.

class vstutils.utils.**Dict**

Wrapper over *dict* which return JSON on conversion to string.

class vstutils.utils.**Executor** (*stdout=-1, stderr=-2, **environ_variables*)

Command executor with realtime output write and line handling. By default and by design executor initialize string

³⁷⁷ <https://docs.python.org/3.6/library/stdtypes.html#str>

³⁷⁸ <https://docs.python.org/3.6/library/functions.html#object>

attribute `output` which will be modified by `+=` operator with new lines by `Executor.write_output()` procedure. Override the method if you want change behavior.

`Executor` class supports periodically (0.01 sec) handling process and execute some checks by overriding `Executor.working_handler()` procedure method. If you want disable this behavior override the method by `None` value or use `UnhandledExecutor`.

Parameters

`environ_variables` (`str`³⁷⁹) –

exception CalledProcessError (`returncode`, `cmd`, `output=None`, `stderr=None`)

Raised when `run()` is called with `check=True` and the process returns a non-zero exit status.

Attributes:

`cmd`, `returncode`, `stdout`, `stderr`, `output`

property stdout

Alias for `output` attribute, to match `stderr`

async aexecute (`cmd`, `cwd`, `env=None`)

Executes commands and outputs its result. Asynchronous implementation.

Parameters

- **cmd** – list of `cmd` command and arguments
- **cwd** – workdir for executions
- **env** – extra environment variables which overrides defaults

Returns

– string with full output

execute (`cmd`, `cwd`, `env=None`)

Executes commands and outputs its result.

Parameters

- **cmd** – list of `cmd` command and arguments
- **cwd** – workdir for executions
- **env** – extra environment variables which overrides defaults

Returns

– string with full output

async post_execute (`cmd`, `cwd`, `env`, `return_code`)

Runs after execution end.

Parameters

- **cmd** – list of `cmd` command and arguments
- **cwd** – workdir for executions
- **env** – extra environment variables which overrides defaults
- **return_code** – return code of executed process

async pre_execute (`cmd`, `cwd`, `env`)

Runs before execution starts.

Parameters

- **cmd** – list of `cmd` command and arguments

- **cwd** -- workdir for executions
- **env** -- extra environment variables which overrides defaults

async working_handler (*proc*)

Additional handler for executions.

Parameters

proc (*asyncio.subprocess.Process*) – running process

write_output (*line*)

Parameters

line (*str*³⁸⁰) -- line from command output

Returns

None

Return type

None

class *vstutils.utils.KVExchanger* (*key, timeout=None*)

Class for transmit data using key-value fast (cache-like) storage between services. Uses same cache-backend as Lock.

class *vstutils.utils.Lock* (*id, payload=1, repeat=1, err_msg="", timeout=None*)

Lock class for multi-jobs workflow. Based on *KVExchanger*. The Lock allows only one thread to enter the part that's locked and shared between apps using one locks cache (see also [locks]).

Parameters

- **id** (*int*³⁸¹, *str*³⁸²) -- unique id for lock.
- **payload** -- lock additional info. Should be any boolean True value.
- **repeat** (*int*³⁸³) -- time to wait lock.release. Default 1 sec.
- **err_msg** (*str*³⁸⁴) -- message for AcquireLockException error.

Note:

- Used django.core.cache lib and settings in *settings.py*
 - Have Lock.SCHEDULER and Lock.GLOBAL id
-

Example:

```
from vstutils.utils import Lock

with Lock("some_lock_identifier", repeat=30, err_msg="Locked by another_
↪process") as lock:
    # where
    # ``"some_lock_identifier"`` is unique id for lock and
    # ``30`` seconds lock is going wait until another process will release_
↪lock id.
    # After 30 sec waiting lock will raised with :class:`.Lock.
↪AcquireLockException`
```

(continues on next page)

³⁷⁹ <https://docs.python.org/3.6/library/stdtypes.html#str>

³⁸⁰ <https://docs.python.org/3.6/library/stdtypes.html#str>

(continued from previous page)

```
# and ``err_msg`` value as text.
some_code_execution()
# ``lock`` object will has been automatically released after
# exiting from context.
```

Another example without context manager:

```
from vstutils.utils import Lock

# locked block after locked object created
lock = Lock("some_lock_identifier", repeat=30, err_msg="Locked by another_
↳process")
# deleting of object calls ``lock.release()`` which release and remove lock_
↳from id.
del lock
```

exception AcquireLockException

Exception which will be raised on unreleased lock.

class vstutils.utils.**ModelHandlers** (type_name, err_message=None)

Handlers for some models like 'INTEGRATIONS' or 'REPO_BACKENDS'. Based on *ObjectHandlers* but more specific for working with models. All handlers backends get by first argument model object.

Attributes:**Parameters**

- **objects** (*dict*³⁸⁵) -- dict of objects like: {<name>: <backend_class>}
- **keys** (*list*³⁸⁶) -- names of supported backends
- **values** (*list*³⁸⁷) -- supported backends classes
- **type_name** -- type name for backends. Like name in dict.

get_object (name, obj)

Parameters

- **name** -- string name of backend
- **name** -- str
- **obj** (*django.db.models.Model*³⁸⁸) -- model object

Returns

backend object

Return type

*object*³⁸⁹

³⁸¹ <https://docs.python.org/3.6/library/functions.html#int>

³⁸² <https://docs.python.org/3.6/library/stdtypes.html#str>

³⁸³ <https://docs.python.org/3.6/library/functions.html#int>

³⁸⁴ <https://docs.python.org/3.6/library/stdtypes.html#str>

³⁸⁵ <https://docs.python.org/3.6/library/stdtypes.html#dict>

³⁸⁶ <https://docs.python.org/3.6/library/stdtypes.html#list>

³⁸⁷ <https://docs.python.org/3.6/library/stdtypes.html#list>

³⁸⁸ <https://docs.djangoproject.com/en/4.2/ref/models/instances/#django.db.models.Model>

³⁸⁹ <https://docs.python.org/3.6/library/functions.html#object>

class `vstutils.utils.ObjectHandlers` (*type_name*, *err_message=None*)

Handlers wrapper for get objects from some settings structure.

Example:

```
from vstutils.utils import ObjectHandlers

'''
In `settings.py` you should write some structure:

SOME_HANDLERS = {
    "one": {
        "BACKEND": "full.python.path.to.module.SomeClass"
    },
    "two": {
        "BACKEND": "full.python.path.to.module.SomeAnotherClass",
        "OPTIONS": {
            "some_named_arg": "value"
        }
    }
}
'''

handlers = ObjectHandlers('SOME_HANDLERS')

# Get class handler for 'one'
one_backend_class = handlers['one']
# Get object of backend 'two'
two_obj = handlers.get_object()
# Get object of backend 'two' with overriding constructor named arg
two_obj_overrided = handlers.get_object(some_named_arg='another_value')
```

Parameters

type_name (*str*³⁹⁰) – type name for backends. Like name in dict.

backend (*name*)

Get backend class

Parameters

name (*str*³⁹¹) – name of backend type

Returns

class of backend

Return type

*type*³⁹², *types.ModuleType*³⁹³, *object*³⁹⁴

class `vstutils.utils.Paginator` (*qs*, *chunk_size=None*)

Class for fragmenting the query for small queries.

class `vstutils.utils.SecurePickling` (*secure_key=None*)

Secured pickle wrapper by Vigenère cipher.

³⁹⁰ <https://docs.python.org/3.6/library/stdtypes.html#str>

³⁹¹ <https://docs.python.org/3.6/library/stdtypes.html#str>

³⁹² <https://docs.python.org/3.6/library/functions.html#type>

³⁹³ <https://docs.python.org/3.6/library/types.html#types.ModuleType>

³⁹⁴ <https://docs.python.org/3.6/library/functions.html#object>

Warning: Do not use it with untrusted transport anyway.

Example:

```
from vstutils.utils import SecurePickling

serializer = SecurePickling('password')

# Init secret object
a = {"key": "value"}
# Serialize object with secret key
pickled = serializer.dumps(a)
# Deserialize object
unpickled = serializer.loads(pickled)

# Check, that object is correct
assert a == unpickled
```

class vstutils.utils.**URLHandlers** (*type_name='URLS', *args, **kwargs*)

Object handler for GUI views. Uses *GUI_VIEWS* from settings.py. Based on *ObjectHandlers* but more specific to urlpatterns.

Example:

```
from vstutils.utils import URLHandlers

# By default gets from `GUI_VIEWS` in `settings.py`
urlpatterns = list(URLHandlers())
```

Parameters

type_name – type name for backends. Like name in dict.

get_object (*name, *argv, **kwargs*)

Get url object tuple for urls.py

Parameters

- **name** (*str*³⁹⁵) – url regexp from
- **argv** – overridden args
- **kwargs** – overridden kwargs

Returns

url object

Return type

django.urls.re_path

class vstutils.utils.**UnhandledExecutor** (*stdout=-1, stderr=-2, **environ_variables*)

Class based on *Executor* but disables *working_handler*.

Parameters

environ_variables (*str*³⁹⁶) –

³⁹⁵ <https://docs.python.org/3.6/library/stdtypes.html#str>

class `vstutils.utils.apply_decorators` (**decorators*)

Decorator which apply list of decorators on method or class.

Example:

```
from vstutils.utils import apply_decorators

def decorator_one(func):
    print(f"Decorated {func.__name__} by first decorator.")
    return func

def decorator_two(func):
    print(f"Decorated {func.__name__} by second decorator.")
    return func

@apply_decorators(decorator_one, decorator_two)
def decorated_function():
    # Function decorated by both decorators.
    print("Function call.")
```

class `vstutils.utils.classproperty` (*fget*, *fset=None*)

Decorator which makes class method as class property.

Example:

```
from vstutils.utils import classproperty

class SomeClass(metaclass=classproperty.meta):
    # Metaclass is needed for set attrs in class
    # instead of and not only object.

    some_value = None

    @classproperty
    def value(cls):
        return cls.some_value

    @value.setter
    def value(cls, new_value):
        cls.some_value = new_value
```

Parameters

- **fget** – function for getting an attribute value.
- **fset** – function for setting an attribute value.

`vstutils.utils.create_view` (*model*, ***meta_options*)

A simple function for getting the generated view by standard means, but with overloaded meta-parameters. This method can completely get rid of the creation of proxy models.

Example:

```
from vstutils.utils import create_view

from .models import Host
```

(continues on next page)

³⁹⁶ <https://docs.python.org/3.6/library/stdtypes.html#str>

(continued from previous page)

```
# Host model has full :class:`vstutils.api.base.ModelViewSet` view.
# For overriding and create simple list view just setup this:
HostListViewSet = create_view(
    HostList,
    view_class='list_only'
)
```

Note: This method is also recommended in cases where there is a problem of recursive imports.

Parameters

model (*Type*[*vstutils.models.BaseModel*]) – Model class with *.get_view_class* method. This method also has *vstutils.models.BModel*.

Return type

vstutils.api.base.GenericViewSet

vstutils.utils.decode (*key*, *enc*)

Decode string from encoded by Vigenère cipher.

Parameters

- **key** (*str*³⁹⁷) – secret key for encoding
- **enc** (*str*³⁹⁸) – encoded string for decoding

Returns

– decoded string

Return type

*str*³⁹⁹

vstutils.utils.deprecated (*func*)

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

Parameters

func – any callable that will be wrapped and will issue a deprecation warning when called.

vstutils.utils.encode (*key*, *clear*)

Encode string by Vigenère cipher.

Parameters

- **key** (*str*⁴⁰⁰) – secret key for encoding
- **clear** (*str*⁴⁰¹) – clear value for encoding

Returns

– encoded string

Return type

*str*⁴⁰²

³⁹⁷ <https://docs.python.org/3.6/library/stdtypes.html#str>

³⁹⁸ <https://docs.python.org/3.6/library/stdtypes.html#str>

³⁹⁹ <https://docs.python.org/3.6/library/stdtypes.html#str>

`vstutils.utils.get_render(name, data, trans='en')`

Render string from template.

Parameters

- **name** ([str](#)⁴⁰³) – full template name
- **data** ([dict](#)⁴⁰⁴) – dict of rendered vars
- **trans** ([str](#)⁴⁰⁵) – translation for render. Default 'en'.

Returns

– rendered string

Return type

[str](#)⁴⁰⁶

`vstutils.utils.lazy_translate(text)`

The `lazy_translate` function has the same behavior as `translate()`, but wraps it in a lazy promise.

This is very useful, for example, for translating error messages in class attributes before the language code is known.

Parameters

text – Text message which should be translated.

`vstutils.utils.list_to_choices(items_list, response_type=<class 'list'>)`

Method to create django model choices from flat list of values.

Parameters

- **items_list** – list of flat values.
- **response_type** – casting type of returned mapping

Returns

list of tuples from `items_list` values

`class vstutils.utils.model_lock_decorator(**kwargs)`

Decorator for functions where 'pk' kwarg exist for lock by id.

Warning:

- On locked error raised `Lock.AcquireLockException`
- Method must have and called with `pk` named arg.

`class vstutils.utils.raise_context(*args, **kwargs)`

Context for exclude exceptions.

`class vstutils.utils.raise_context_decorator_with_default(*args, **kwargs)`

Context for exclude errors and return default value.

Example:

⁴⁰⁰ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁴⁰¹ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁴⁰² <https://docs.python.org/3.6/library/stdtypes.html#str>
⁴⁰³ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁴⁰⁴ <https://docs.python.org/3.6/library/stdtypes.html#dict>
⁴⁰⁵ <https://docs.python.org/3.6/library/stdtypes.html#str>
⁴⁰⁶ <https://docs.python.org/3.6/library/stdtypes.html#str>

```

from yaml import load
from vstutils.utils import raise_context_decorator_with_default

@raise_context_decorator_with_default(default={})
def get_host_data(yaml_path, host):
    with open(yaml_path, 'r') as fd:
        data = load(fd.read(), Loader=Loader)
    return data[host]
    # This decorator used when you must return some value even on error
    # In log you also can see traceback for error if it occur

def clone_host_data(host):
    bs_data = get_host_data('inventories/aws/hosts.yml', 'build_server')
    ...

```

class `vstutils.utils.redirect_stdany` (*new_stream=<io.StringIO object>, streams=None*)
Context for redirect any output to own stream.

Note:

- On context returns stream object.
 - On exit returns old streams.
-

`vstutils.utils.send_mail` (*subject, message, from_email, recipient_list, fail_silently=False, auth_user=None, auth_password=None, connection=None, html_message=None, **kwargs*)

Wrapper over `django.core.mail.send_mail()`⁴⁰⁷ which provide additional named arguments.

`vstutils.utils.send_template_email` (*sync=False, **kwargs*)

Function executing sync or async email sending; according *sync* argument and settings variable “RPC_ENABLED”. If you don’t set settings for celery or don’t have celery it sends synchronously mail. If celery is installed and configured and *sync* argument of the function is set to *False*, it sends asynchronously email.

Parameters

- **sync** – argument for determining how send email, asynchronously or synchronously
- **subject** – mail subject.
- **email** – list of strings or single string, with email addresses of recipients
- **template_name** – relative path to template in *templates* directory, must include extension in file name.
- **context_data** – dictionary with context for rendering message template.

`vstutils.utils.send_template_email_handler` (*subject, email_from, email, template_name, context_data=None, **kwargs*)

Function for email sending. The function convert recipient to list and set context before sending if it possible.

Parameters

- **subject** – mail subject.
- **email_from** – sender that be setup in email.
- **email** – list of strings or single string, with email addresses of recipients

⁴⁰⁷ https://docs.djangoproject.com/en/4.2/topics/email/#django.core.mail.send_mail

- **template_name** – relative path to template in *templates* directory, must include extension in file name.
- **context_data** – dictionary with context for rendering message template.
- **kwargs** – additional named arguments for *send_mail*

Returns

Number of emails sent.

class `vstutils.utils.tmp_file` (*data=""*, *mode='w'*, *bufsize=-1*, ***kwargs*)

Temporary file with name generated and auto removed on close.

Attributes:**Parameters**

- **data** (*str*⁴⁰⁸) – string to write in tmp file.
- **mode** (*str*⁴⁰⁹) – file open mode. Default 'w'.
- **bufsize** (*int*⁴¹⁰) – buffer size for `tempfile.NamedTemporaryFile`
- **kwargs** – other kwargs for `tempfile.NamedTemporaryFile`

write (*wr_string*)

Write to file and flush

Parameters

wr_string (*str*⁴¹¹) – writable string

Returns

None

Return type

None

class `vstutils.utils.tmp_file_context` (**args*, ***kwargs*)

Context object for work with `tmp_file`. Auto close on exit from context and remove if file still exist.

This context manager over *tmp_file*

`vstutils.utils.translate` (*text*)

The `translate` function supports translation message dynamically with standard i18n vstutils'es mechanisms usage.

Uses `django.utils.translation.get_language()`⁴¹² to get the language code and tries to get the translation from the list of available ones.

Parameters

text – Text message which should be translated.

⁴⁰⁸ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁴⁰⁹ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁴¹⁰ <https://docs.python.org/3.6/library/functions.html#int>

⁴¹¹ <https://docs.python.org/3.6/library/stdtypes.html#str>

⁴¹² https://docs.djangoproject.com/en/4.2/ref/utils/#django.utils.translation.get_language

Frontend Quickstart

VST utils framework uses Vue ecosystem to render frontend. The quickstart manual will guide you through the most important steps to customize frontend features. App installation and setting up described in - [Backend Section](#) of this docs.

There are several stages in vstutils app:

1. Before app started:

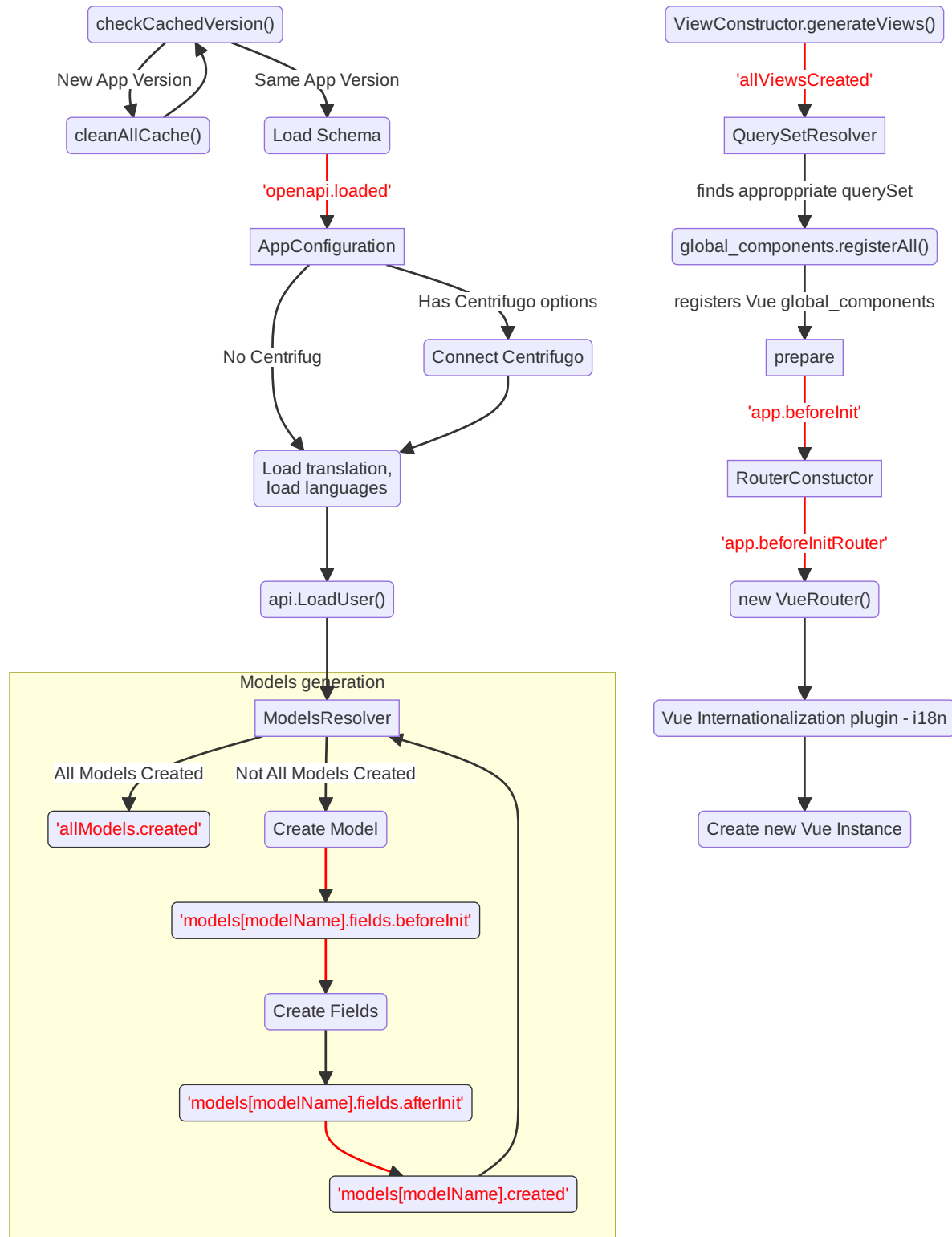
- *checkCacheVersions()* checks if app version has been changed since last visit and cleans all cached data if so;
- loading open api schema from backend. Emits 'openapi.loaded' signal;
- loading all static files from *SPA_STATIC* in *setting.py*;
- sets *AppConfiguration* from OpenAPI schema;

2. App started:

- if there is *centrifugoClient* in *settings.py* connects it. To read more about centrifugo configuration check "[Centrifugo client settings](#)";
- downloading a list of available languages and translations;
- *api.loadUser()* returns user data;
- *ModelsResolver* creates models from schema, emits signal *models[\${modelName}].created* for each created model and *allModels.created* when all models created;
- *ViewConstructor.generateViews()* inits *View* fieldClasses and modelClasses;
- *QuerySetsResolver* finds appropriate queryset by model name and view path;
- *global_components.registerAll()* registers Vue *global_components*;
- *prepare()* emits *app.beforeInit* with { app: this };
- initialize model with *LocalSettings*. Find out more about this in the section [LocalSettings](#);
- creates routerConstructor from *this.views*, emits 'app.beforeInitRouter' with { routerConstructor } and gets new *VueRouter*({this.routes});
- inits application *Vue()* from schema.info, pinia store and emits 'app.afterInit' with {app: this};

3. Application mounted.

There is a flowchart representing application initialization process (signal names have red font):



4.1 Field customization

To add custom script to the project, set script name in settings.py

```
SPA_STATIC += [
    {'priority': 101, 'type': 'js', 'name': 'main.js', 'source': 'project_lib'},
]
```

and put the script (*main.js*) in *{appName}/static/* directory.

1. In *main.js* create new field by extending it from *BaseField* (or any other appropriate field)

For example lets create a field that renders HTML h1 element with 'Hello World!' text:

```
class CustomField extends spa.fields.base.BaseField {
  static get mixins() {
    return super.mixins.concat({
      render(createElement) {
        return createElement('h1', {}, 'Hello World!');
      },
    });
  }
}
```

Or render person's name with some prefix

```
class CustomField extends spa.fields.base.BaseField {
  static get mixins() {
    return super.mixins.concat({
      render(h) {
        return h("h1", {}, `Mr ${this.$props.data.name}`);
      },
    });
  }
}
```

2. Register this field to *app.fieldsResolver* to provide appropriate field format and type to a new field

```
const customFieldFormat = 'customField';
app.fieldsResolver.registerField('string', customFieldFormat, CustomField);
```

3. Listen for a appropriate *models[ModelWithFieldToChange].fields.beforeInit* signal to change field Format

```
spa.signals.connect(`models[ModelWithFieldToChange].fields.beforeInit`, (fields) => {
  fields.fieldToChange.format = customFieldFormat;
});
```

List of models and their fields is available during runtime in console at *app.modelsClasses*

To change Filed behavior, create new field class with a desired logic. Let's say you need to send number of milliseconds to API, user however wants to type in number of seconds. A solution would be to override field's *toInner* and *toRepresent* methods.

```
class MilliSecondsField extends spa.fields.numbers.integer.IntegerField {
  toInner(data) {
    return super.toInner(data) * 1000;
  }
  toRepresent(data) {
```

(continues on next page)

(continued from previous page)

```

    return super.toRepresent(data)/1000;
  }
}

const milliSecondsFieldFormat = 'milliSeconds'
app.fieldsResolver.registerField('integer', milliSecondsFieldFormat, ↵
  ↵MilliSecondsField);
spa.signals.connect(`models[OneAllFields].fields.beforeInit`, (fields) => {
  fields.integer.format = milliSecondsFieldFormat;
});

```

Now you have field that show seconds, but saves/receives data in milliseconds on detail view of AllFieldsModel.

Note: If you need to show some warning or error to developer console you can use field *warn* and *error* methods. You can pass some message and it will print it with field type, model name and field name.

4.2 Change path to FkField

Sometime you may need to request different set of objects for FkField. For example to choose from only famous authors, create *famous_author* endpoint on backend and set FkField request path to *famous_author*. Listen for *app.beforeInit* signal.

```

spa.signals.connect('app.beforeInit', ({ app }) => {
  app.modelsResolver.get('OnePost').fields.get('author').querysets.get('/post/new/
  ↵') [0].url = '/famous_author/'
});

```

Now when we create new post on */post/* endpoint Author FkField makes get request to */famous_author/* instead of */author/*. It's useful to get different set of authors (that may have been previously filtered on backend).

4.3 CSS Styling

1. Like scripts, css files may be added to SPA_STATIC in setting.py

```

SPA_STATIC += [

  {'priority': 101, 'type': 'css', 'name': 'style.css', 'source': 'project_lib'},
]

```

Let's inspect page and find css class for our customField. It is *column-format-customField* and generated with *column-format-{Field.format}* pattern.

2. Use regular css styling to change appearance of the field.

```

.column-format-customField:hover {
  background-color: orangered;
  color: white;
}

```

Other page elements are also available for styling: for example, to hide certain column set corresponding field to none.


```
.column-format-customField {
  display: none;
}
```

4.4 Show primary key column on list

Every pk column has *pk-column* CSS class and hidden by default (using *display: none;*).

For example this style will show pk column on all list views of *Order* model:

```
.list-Order .pk-column {
  display: table-cell;
}
```

4.5 View customization

Listen for signal “*allViews.created*” and add new custom mixin to the view.

Next code snippet depicts rendering new view instead of default view.

```
spa.signals.once('allViews.created', ({ views }) => {
  const AuthorListView = views.get('/author/');
  AuthorListView.mixins.push({
    render(h) {
      return h('h1', {}, `Custom view`);
    },
  });
});
```

Learn more about Vue *render()* function at [Vue documentation](https://v3.vuejs.org/guide/render-function.html)⁴¹³.

It is also possible to fine tune View by overriding default computed properties and methods of existing mixins. For example, override breadcrumbs computed property to turn off breadcrumbs on Author list View

```
import { ref } from 'vue';

spa.signals.once("allViews.created", ({ views }) => {
  const AuthorListView = views.get("/author/");
  AuthorListView.extendStore((store) => {
    return {
      ...store,
      breadcrumbs: ref([]),
    };
  });
});
```

Sometimes you may need to hide detail page for some reason, but still want all actions and sublinks to be accessible from list page. To do it you also should listen signal “*allViews.created*” and change parameter *hidden* from default *false* to *true*, for example:

⁴¹³ <https://v3.vuejs.org/guide/render-function.html>

```
spa.signals.once('allViews.created', ({ views }) => {  
  const authorView = views.get('/author/{id}/');  
  authorView.hidden = true;  
});
```

4.6 Changing title of the view

To change title and string displayed in the breadcrumbs change *title* property of the view or method *getTitle* for more complex logic.

```
spa.signals.once('allViews.created', ({ views }) => {  
  const usersList = views.get('/user/');  
  usersList.title = 'Users list';  
  
  const userDetails = views.get('/user/{id}/');  
  userDetails.getTitle = (state) => (state?.instance ? `User: ${state.instance.id}`  
    ↪: 'User');  
});
```

4.7 Basic Webpack configuration

To use webpack in your project rename *webpack.config.js.default* to *webpack.config.js*. Every project based on vst-utils contains *index.js* in */frontend_src/app/* directory. This file is intended for your code. Run *yarn* command to install all dependencies. Then run *yarn devBuild* from root dir of your project to build static files. Final step is to add built file to *SPA_STATIC* in *settings.py*

```
SPA_STATIC += [  
  {'priority': 101, 'type': 'js', 'name': '{AppName}/bundle/app.js', 'source':  
    ↪ 'project_lib'},  
]
```

Webpack configuration file allows to add more static files. In *webpack.config.js* add more entries

```
const config = {  
  mode: setMode(),  
  entry: {  
    'app': entrypoints_dir + "/app/index.js" // default,  
    'myapp': entrypoints_dir + "/app/myapp.js" // just added  
  },  
};
```

Output files will be built into *frontend_src/{AppName}/static/{AppName}/bundle* directory. Name of output file corresponds to name of entry in *config*. In the example above output files will have names *app.js* and *myapp.js*. Add all of these files to *STATIC_SPA* in *settings.py*. During vstutils installation through *pip* frontend code are being built automatically, so you may need to add *bundle* directory to *gitignore*.

4.8 Page store

Every page has store that can be accessed globally *app.store.page* or from page component using *this.store*.

View method *extendStore* can be used to add custom logic to page's store.

```
import { computed } from 'vue';

spa.signals.once('allViews.created', ({ views }) => {
  views.get('/user/{id}/').extendStore((store) => {
    // Override title of current page using computed value
    const title = computed(() => `Current page has ${store.instances.length}
    ↪instances`);

    async function fetchData() {
      await store.fetchData(); // Call original fetchData
      await callSomeExternalApi(store.instances.value);
    }

    return {
      ...store,
      title,
      fetchData,
    };
  });
});
```

4.9 Overriding root component

Root component of the application can be overridden using *app.beforeInit* signal. This can be useful for such things as changing layout CSS classes, back button behaviour or main layout components.

Example of customizing sidebar component:

```
const CustomAppRoot = {
  components: { Sidebar: CustomSidebar },
  mixins: [spa.AppRoot],
};

spa.signals.once('app.beforeInit', ({ app }) => {
  app.appRootComponent = CustomAppRoot;
});
```

4.10 Translating values of fields

Values tha displayed by *FKField* of *ChoicesField* can be translated using standard translations files.

Translation key must be defined as *:model:<ModelName>:<fieldName>:<value>*. For example:

```
TRANSLATION = {
  ':model:Category:name:Category 1': 'Категория 1',
}
```

Translation of values can be taxing as every model on backend usually generates more than one model on frontend, To avoid this, add *_translate_model = 'Category'* attribute to model on backend. It shortens

```
' :model:Category:name:Category 1': 'Категория 1',  
' :model:OneCategory:name:Category 1': 'Категория 1',  
' :model:CategoryCreate:name:Category 1': 'Категория 1',
```

to

```
' :model:Category:name:Category 1': 'Категория 1',
```

For *FKField* name of the related model is used. And *fieldName* should be equal to *viewField*.

4.11 Changing actions or sublinks

Sometimes using only schema for defining actions or sublinks is not enough.

For example we have an action to make user a superuser (*/user/{id}/make_superuser/*) and we want to hide that action if user is already a superuser (*is_superuser* is *true*). *<\${PATH}>filterActions* signal can be used to achieve such result.

```
spa.signals.connect('</user/{id}/make_superuser/>filterActions', (obj) => {  
  if (obj.data.is_superuser) {  
    obj.actions = obj.actions.filter((action) => action.name !== 'make_superuser'  
    ↪});  
  }  
});
```

1. *<\${PATH}>filterActions* receives {actions, data}
2. *<\${PATH}>filterSublinks* receives {sublinks, data}

Data property will contain instance's data. Actions and sublinks properties will contain arrays with default items (not hidden action or sublinks), it can be changed or replaced completely.

4.12 LocalSettings

This model's fields are displayed in the left sidebar. All data from this model saves in browser Local Storage. If you want to add another options, you can do it using *beforeInit* signal, for example:

```
spa.signals.once('models[_LocalSettings].fields.beforeInit', (fields) => {  
  const cameraField = new spa.fields.base.BaseField({ name: 'camera' });  
  // You can add some logic here  
  fields.camera = cameraField;  
});
```

4.13 Store

There are three ways to store data:

- *userSettingsStore* - saves data on the server. By default, there are options for changing language and a button to turn on/off the dark mode. Data to *userSettingsStore* comes from schema.
- *localSettingsStore* - saves data in the browser Local Storage. This is where you can store your own fields, as described in [LocalSettings](#).

- *store* - stores current page data.

To use any of this stores you need to run the following command: `app.[storeName]`, for example: `app.userSettingsStore`.

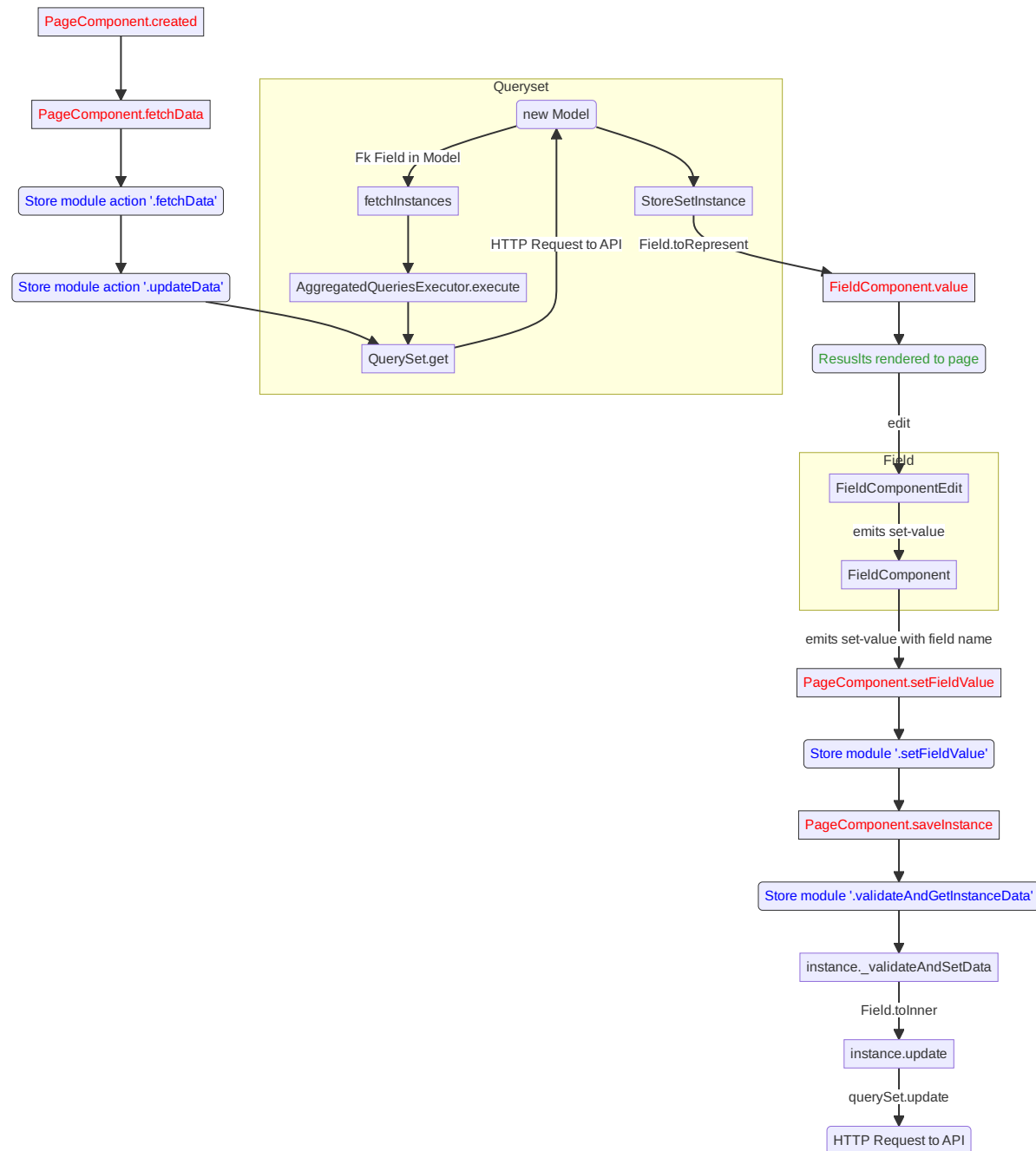
Note: If you are accessing the `userSettingsStore` from within the component then you need to use `this.$app` instead `app`.

From *app.store* you may need:

- *viewsItems* and *viewItemsMap* - stores information about parent views for this page. It is used for example in breadcrumbs. The difference between them is only in the way information is stored: *viewItems* is an Array of Objects and *viewItemsMap* is a Map.
- *page* - saves all information about current page.
- *title* - title of current page.

5.1 API Flowchart

This flowchart shows how data goes through application from and to API.



5.2 Signals

System of signals is a mechanism, that VST Utils uses for app customization.

Let's look how it works.

Very often you need to modify something after some event has occurred. But how can you know about this event? And what if you need to know about this event in several blocks of code?

To solve this problem VST Utils uses system of signals, where:

- you can emit some signal, which tells all subscribers, that some event has occurred, and pass some data/variables from the context, where this event has occurred;
- you can subscribe to some signal, that notifies you about some event, and also you can pass some callback (handler) that can do something with data/variables, that were passed from the context, where event had occurred.

5.2.1 Emit signal

To emit some signal you need to write following in you code:

```
tabSignal.emit(name_of_signal, context);
```

where:

- **name_of_signal** - string, which stores name of signal (event);
- **context** - some variable of any type, that will be passed to the callback (handler) during connection to this signal.

Example of signal emitting:

```
let app = {
  name: 'example of app';
};

tabSignal.emit('app.created', app);
```

5.2.2 Connect to signal

To connect to some signal you need to write following in you code:

```
tabSignal.connect(name_of_signal, callback);
```

where:

- **name_of_signal** - string, which stores name of signal (event);
- **callback** - function, that can do something with variables, which will be passed from event's context to this callback as arguments.

Example of connecting to signal:

```
/* ... */
function callback(app) {
  app.title = 'example of app title';
}

tabSignal.connect('app.created', callback);
/* ... */
```

5.3 List of signals in VST Utils

VST Utils has some signals, that are emitting during application work. If you need to customize something in you project you can subscribe to these signals and add callback function with desired behavior. Also you can emit you own signals in your project.

5.3.1 openapi.loaded

Signal name: “openapi.loaded”.

Context argument: openapi - {object} - OpenAPI schema loaded from API.

Description: This signal is emitted after OpenAPI schema was loaded. You can use this signal if you need to change something in the OpenAPI schema, before it was parsed.

5.3.2 resource.loaded

Signal name: “resource.loaded”.

Context argument: None.

Description: This signal is emitted after all static files were successfully loaded and added to the page.

5.3.3 app.version.updated

Signal name: “app.version.updated”.

Context argument: None.

Description: This signal is emitted during app loading if VST Utils detects, that version of your project was updated.

5.3.4 app.beforeInitRouter

Signal name: “app.beforeInitRouter”.

Context argument: obj - {object} - Object with following structure: {routerConstructor: RouterConstructor}, where routerConstructor is an instance of RouterConstructor.

Description: This signal is emitted after creation of RouterConstructor instance and before app creation

5.3.5 app.beforeInit

Signal name: “app.beforeInit”.

Context argument: obj - {object} - Object with following structure: {app: app}, where app is an instance of App class.

Description: This signal is emitted after app variable initialization (OpenAPI schema was parsed, models and views were created), but before app was mounted to the page.

5.3.6 app.afterInit

Signal name: “app.afterInit”.

Context argument: obj - {object} - Object with following structure: {app: app}, where app is an instance of App class.

Description: This signal is emitted after app was mounted to the page.

5.3.7 app.language.changed

Signal name: “app.language.changed”.

Context argument: obj - {object} - Object with following structure: {lang: lang}, where lang is an code of applied language.

Description: This signal is emitted after app interface language was changed.

5.3.8 models[model_name].fields.beforeInit

Signal name: “models[” + model_name + “].fields.beforeInit”. For example, for User model: “models[User].fields.beforeInit”.

Context argument: fields - {object} - Object with pairs of key, value, where key - name of field, value - object with it options. On this moment, field - is just object with options, it is not guiFields instance.

Description: This signal is emitted before creation of guiFields instances for Model fields.

5.3.9 models[model_name].fields.afterInit

Signal name: “models[” + model_name + “].fields.afterInit”. For example, for User model: “models[User].fields.afterInit”.

Context argument: fields - {object} - Object with pairs of key, value, where key - name of field, value - guiFields instance.

Description: This signal is emitted after creation of guiFields instances for Model fields.

5.3.10 models[model_name].created

Signal name: “models[” + model_name + “].created”. For example, for User model: “models[User].created”.

Context argument: obj - {object} - Object with following structure: {model: model}, where model is the created Model.

Description: This signal is emitted after creation of Model object.

5.3.11 allModels.created

Signal name: “allModels.created”.

Context argument: obj - {object} - Object with following structure: {models: models}, where models is the object, storing Models objects.

Description: This signal is emitted after all models were created.

5.3.12 allViews.created

Signal name: “allViews.created”.

Context argument: obj - {object} - Object with following structure: {views: views}, where views - object with all View Instances.

Description: This signal is emitted after creation of all View Instances, with set actions / child_links / multi_actions / operations / sublinks properties.

5.3.13 routes[name].created

Signal name: “routes[” + name + “].created”. For example, for /user/ view: “routes[/user/].created”.

Context argument: route - {object} - Object with following structure: {name: name, path: path, component: component}, where name - name of route, path - template of route’s path, component - component, that will be rendered for current route.

Description: This signal will be emitted after route was formed and added to routes list.

5.3.14 allRoutes.created

Signal name: “allRoutes.created”.

Context argument: routes - {array} - Array with route objects with following structure: {name: name, path: path, component: component}, where name - name of route, path - template of route’s path, component - component, that will be rendered for current route.

Description: This signal is emitted after all routes was formed and added to routes list.

5.3.15 <\${PATH}>filterActions

Signal name: “<\${PATH}>filterActions”.

Context argument: obj - {actions: Object[], data} - Actions is array of action objects. Data represents current instance’s data.

Description: This signal will be executed to filter actions.

5.3.16 <\${PATH}>filterSublinks

Signal name: “<\${PATH}>filterSublinks”.

Context argument: obj - {sublinks: Object[], data} - Actions is array of sublink objects. Data represents current instance's data.

Description: This signal will be executed to filter sublinks.

5.4 Field Format

Very often during creation of some new app developers need to make common fields of some base types and formats (string, boolean, number and so on). Create everytime similar functionality is rather boring and ineffective, so we tried to solve this problem with the help of VST Utils.

VST Utils has set of built-in fields of the most common types and formats, that can be used for different cases. For example, when you need to add some field to you web form, that should hide value of inserted value, just set appropriate field format to `password` instead of `string` to show stars instead of actual characters.

Field classes are used in Model Instances as fields and also are used in Views Instances of `list` type as filters.

All available fields classes are stored in the `guiFields` variable. There are 44 fields formats in VST Utils:

- **base** - base field, from which the most other fields are inherited;
- **string** - string field, for inserting and representation of some short 'string' values;
- **textarea** - string field, for inserting and representation of some long 'string' values;
- **number** - number field, for inserting and representation of 'number' values;
- **integer** - number field, for inserting and representation of values of 'integer' format;
- **int32** - number field, for inserting and representation of values of 'int32' format;
- **int64** - number field, for inserting and representation of values of 'int64' format;
- **double** - number field, for inserting and representation of values of 'double' format;
- **float** - number field, for inserting and representation of values of 'float' format;;
- **boolean** - boolean field, for inserting and representation of 'boolean' values;
- **choices** - string field, with strict set of preset values, user can only choose one of the available value variants;
- **autocomplete** - string field, with set of preset values, user can either choose one of the available value variants or insert his own value;
- **password** - string field, that hides inserted value by '*' symbols;
- **file** - string field, that can read content of the file;
- **secretfile** - string field, that can read content of the file and then hide it from representation;
- **binfile** - string field, that can read content of the file and convert it to the 'base64' format;
- **namedbinfile** - field of JSON format, that takes and returns JSON with 2 properties: name (string) - name of file and content(base64 string) - content of file;
- **namedbinimage** - field of JSON format, that takes and returns JSON with 2 properties: name (string) - name of image and content(base64 string) - content of image;
- **multiplenamedbinfile** - field of JSON format, that takes and returns array with objects, consisting of 2 properties: name (string) - name of file and content(base64 string) - content of file;

- **multiplenamebinimage** - field of JSON format, that takes and returns array with objects, consisting of 2 properties: name (string) - name of image and content(base64 string) - content of image;
- **text_paragraph** - string field, that is represented as text paragraph (without any inputs);
- **plain_text** - string field, that saves all non-printing characters during representation;
- **html** - string field, that contents different html tags and that renders them during representation;
- **date** - date field, for inserting and representation of 'date' values in 'YYYY-MM-DD' format;
- **date_time** - date field, for inserting and representation of 'date' values in 'YYYY-MM-DD HH:mm' format;
- **uptime** - string field, that converts time duration (amount of seconds) into one of the most appropriate variants (23:59:59 / 01d 00:00:00 / 01m 30d 00:00:00 / 99y 11m 30d 22:23:24) due to the it's value size;
- **time_interval** - number field, that converts time from milliseconds into seconds;
- **crontab** - string field, that has additional form for creation schedule in 'crontab' format;
- **json** - field of JSON format, during representation it uses another guiFields for representation of current field properties;
- **api_object** - field, that is used for representation of some Model Instance from API (value of this field is the whole Model Instance data). This is read only field;
- **fk** - field, that is used for representation of some Model Instance from API (value of this field is the Model Instance Primary Key). During edit mode this field has strict set of preset values to choose;
- **fk_autocomplete** - field, that is used for representation of some Model Instance from API (value of this field is the Model Instance Primary Key or some string). During edit mode user can either choose of the preset values from autocomplete list or insert his own value;
- **fk_multi_autocomplete** - field, that is used for representation of some Model Instance from API (value of this field is the Model Instance Primary Key or some string). During edit mode user can either choose of the preset values from modal window or insert his own value;
- **color** - string field, that stores HEX code of selected color;
- **inner_api_object** - field, that is linked to the fields of another model;
- **api_data** - field for representing some data from API;
- **dynamic** - field, that can change its format depending on the values of surrounding fields;
- **hidden** - field, that is hidden from representation;
- **form** - field, that combines several other fields and stores those values as one JSON, where key - name of form field, value - value of form field;
- **button** - special field for form field, imitates button in form;
- **string_array** - field, that converts array with strings into one string;
- **string_id** - string field, that is supposed to be used in URLs as 'id' key. It has additional validation, that checks, that field's value is not equal to some other URL keys (new/ edit/ remove).

5.5 Layout customization with CSS

If you need to customize elements with css we have some functionality for it. There are classes applied to root elements of `EntityView` (if it contains `ModelField`), `ModelField`, `ListTableRow` and `MultiActions` depending on the fields they contain. Classes are formed for the fields with “boolean” and “choices” types. Also classes apply to operations buttons and links.

Classes generation rules

- `EntityView`, `ModelField` and `ListTableRow` - `field-[field_name]-[field-value]`

Example:

- `field-active-true` for model that contains “boolean” field with name “active” and value “true”
- `field-tariff_type-WAREHOUSE` for model that contains “choices” field with name “tariff_type” and value “WAREHOUSE”

- `MultiActions` - `selected__field-[field_name]-[field-value]`

Example:

`selected__field-tariff_type-WAREHOUSE` and `selected__field-tariff_type-SLIDE` if selected 2 `ListTableRow` that contains “choices” field with name “tariff_type” and values “WAREHOUSE” and “SLIDE” respectively.

- `Operation` - `operation__[operation_name]`

Warning

If you hide operations using CSS classes and for example all actions were hidden then Actions dropdown button will still be visible.

For better control over actions and sublinks see [Changing actions or sublinks](#)

Example:

`operation__pickup_point` if operation button or link has name `pickup_point`

Based on these classes, you can change the styles of various elements.

A few use cases:

- If you need to hide the button for the “change_category” action on a product detail view when product is not “active”, you can do so by adding a CSS selector:

```
.field-status-true .operation__change_category {
    display: none;
}
```

- Hide the button for the “remove” action in `MultiActions` menu if selected at least one product with status “active”:

```
.selected__field-status-true .operation__remove {
    display: none;
}
```

- If you need to change `background-color` to red for order with status “CANCELLED” on `ListView` component do this:

```
.item-row.field-status-CANCELLED {
    background-color: red;
}
```

In this case, you need to use the extra class “item-row” (Used for example, you can choose another one) for specify the element to be selected in the selector, because the class “field-status-CANCELLED” is added in different places on the page.

V

- `vstutils.api.actions`, [56](#)
- `vstutils.api.base`, [51](#)
- `vstutils.api.decorators`, [55](#)
- `vstutils.api.endpoint`, [67](#)
- `vstutils.api.fields`, [38](#)
- `vstutils.api.filter_backends`, [64](#)
- `vstutils.api.filters`, [60](#)
- `vstutils.api.responses`, [61](#)
- `vstutils.api.serializers`, [51](#)
- `vstutils.api.validators`, [48](#)
- `vstutils.middleware`, [62](#)
- `vstutils.models`, [31](#)
- `vstutils.models.custom_model`, [35](#)
- `vstutils.models.decorators`, [34](#)
- `vstutils.models.fields`, [36](#)
- `vstutils.models.queryset`, [34](#)
- `vstutils.tasks`, [66](#)
- `vstutils.tests`, [72](#)
- `vstutils.utils`, [78](#)

A

Action (class in *vstutils.api.actions*), 56
 aexecute() (*vstutils.utils.Executor* method), 79
 apply_decorators (class in *vstutils.utils*), 84
 assertCheckDict() (*vstutils.tests.BaseTestCase* method), 72
 assertCount() (*vstutils.tests.BaseTestCase* method), 72
 assertRCode() (*vstutils.tests.BaseTestCase* method), 72
 attr_class (*vstutils.models.fields.MultipleFileField* attribute), 37
 attr_class (*vstutils.models.fields.MultipleImageField* attribute), 37
 AutoCompletionField (class in *vstutils.api.fields*), 38

B

backend() (*vstutils.utils.ObjectHandlers* method), 82
 Barcode128Field (class in *vstutils.api.fields*), 39
 BaseEnum (class in *vstutils.utils*), 78
 BaseMiddleware (class in *vstutils.middleware*), 62
 BaseResponseClass (class in *vstutils.api.responses*), 61
 BaseSerializer (class in *vstutils.api.serializers*), 51
 BaseTestCase (class in *vstutils.tests*), 72
 BaseTestCase.user_as (class in *vstutils.tests*), 76
 BaseVstObject (class in *vstutils.utils*), 78
 BinFileInStringField (class in *vstutils.api.fields*), 39
 BModel (class in *vstutils.models*), 31
 BQuerySet (class in *vstutils.models.queryset*), 34
 bulk() (*vstutils.tests.BaseTestCase* method), 72
 bulk_transactional() (*vstutils.tests.BaseTestCase* method), 72

C

call_registration() (*vstutils.tests.BaseTestCase* method), 73
 classproperty (class in *vstutils.utils*), 84

cleared() (*vstutils.models.queryset.BQuerySet* method), 34
 CommaMultiSelect (class in *vstutils.api.fields*), 40
 copy() (*vstutils.api.base.CopyMixin* method), 51
 copy_field_name (*vstutils.api.base.CopyMixin* attribute), 51
 copy_prefix (*vstutils.api.base.CopyMixin* attribute), 51
 copy_related (*vstutils.api.base.CopyMixin* attribute), 51
 CopyMixin (class in *vstutils.api.base*), 51
 create_action_serializer() (*vstutils.api.base.GenericViewSet* method), 53
 create_view() (in module *vstutils.utils*), 84
 CrontabField (class in *vstutils.api.fields*), 40
 CSVFileField (class in *vstutils.api.fields*), 39

D

data (*vstutils.models.custom_model.ListModel* attribute), 36
 decode() (in module *vstutils.utils*), 85
 DeepFkField (class in *vstutils.api.fields*), 41
 DeepViewFilterBackend (class in *vstutils.api.filter_backends*), 64
 DefaultIDFilter (class in *vstutils.api.filters*), 60
 DefaultNameFilter (class in *vstutils.api.filters*), 60
 delete() (*vstutils.models.fields.MultipleFieldFile* method), 36
 DependEnumField (class in *vstutils.api.fields*), 41
 DependFromFkField (class in *vstutils.api.fields*), 41
 deprecated() (in module *vstutils.utils*), 85
 descriptor_class (*vstutils.models.fields.MultipleFileField* attribute), 37
 descriptor_class (*vstutils.models.fields.MultipleImageField* attribute), 37

- `details_test()` (*vstutils.tests.BaseTestCase* method), 73
- `Dict` (class in *vstutils.utils*), 78
- `do()` (*vstutils.tasks.TaskClass* class method), 66
- `DynamicJsonTypeField` (class in *vstutils.api.fields*), 42
- ## E
- `EmptyAction` (class in *vstutils.api.actions*), 58
- `EmptySerializer` (class in *vstutils.api.serializers*), 51
- `encode()` (in module *vstutils.utils*), 85
- `endpoint_call()` (*vstutils.tests.BaseTestCase* method), 73
- `endpoint_schema()` (*vstutils.tests.BaseTestCase* method), 73
- `EndpointViewSet` (class in *vstutils.api.endpoint*), 67
- `execute()` (*vstutils.utils.Executor* method), 79
- `Executor` (class in *vstutils.utils*), 78
- `Executor.CalledProcessError`, 79
- `ExternalCustomModel` (class in *vstutils.models.custom_model*), 35
- `extra_filter()` (in module *vstutils.api.filters*), 60
- ## F
- `file_field` (*vstutils.api.fields.MultipleNamedBinaryFileInJsonField* attribute), 44
- `FileInStringField` (class in *vstutils.api.fields*), 42
- `FileMediaTypeValidator` (class in *vstutils.api.validators*), 48
- `FileModel` (class in *vstutils.models.custom_model*), 35
- `FileResponseRetrieveMixin` (class in *vstutils.api.base*), 51
- `filter_queryset()` (*vstutils.api.filter_backends.HideHiddenFilterBackend* method), 64
- `filter_queryset()` (*vstutils.api.filter_backends.SelectRelatedFilterBackend* method), 64
- `FkField` (class in *vstutils.api.fields*), 42
- `FkFilterHandler` (class in *vstutils.api.filters*), 60
- `FkModelField` (class in *vstutils.api.fields*), 43
- `FkModelField` (class in *vstutils.models.fields*), 36
- ## G
- `GenericViewSet` (class in *vstutils.api.base*), 52
- `get()` (*vstutils.api.endpoint.EndpointViewSet* method), 67
- `get_client()` (*vstutils.api.endpoint.EndpointViewSet* method), 67
- `get_count()` (*vstutils.tests.BaseTestCase* method), 74
- `get_django_settings()` (*vstutils.utils.BaseVstObject* class method), 78
- `get_file()` (*vstutils.models.fields.MultipleFileDescriptor* method), 37
- `get_model_class()` (*vstutils.tests.BaseTestCase* method), 74
- `get_model_filter()` (*vstutils.tests.BaseTestCase* method), 74
- `get_object()` (*vstutils.utils.ModelHandlers* method), 81
- `get_object()` (*vstutils.utils.URLHandlers* method), 83
- `get_paginator()` (*vstutils.models.queryset.BQuerySet* method), 34
- `get_prep_value()` (*vstutils.models.fields.MultipleFileMixin* method), 37
- `get_query_serialized_data()` (*vstutils.api.base.GenericViewSet* method), 53
- `get_render()` (in module *vstutils.utils*), 86
- `get_response_handler()` (*vstutils.middleware.BaseMiddleware* method), 62
- `get_result()` (*vstutils.tests.BaseTestCase* method), 74
- `get_schema_operation_parameters()` (*vstutils.api.filter_backends.VSTFilterBackend* method), 65
- `get_serializer()` (*vstutils.api.base.GenericViewSet* method), 53
- `get_serializer()` (*vstutils.api.endpoint.EndpointViewSet* method), 67
- `get_serializer_class()` (*vstutils.api.base.GenericViewSet* method), 53
- `get_serializer_context()` (*vstutils.api.endpoint.EndpointViewSet* method), 67
- `get_url()` (*vstutils.tests.BaseTestCase* method), 75
- ## H
- `handler()` (*vstutils.middleware.BaseMiddleware* method), 63
- `has_pillow` (*vstutils.api.validators.ImageValidator* property), 49
- `hidden` (*vstutils.models.BModel* attribute), 34
- `HideHiddenFilterBackend` (class in *vstutils.api.filter_backends*), 64
- `HistoryModelViewSet` (class in *vstutils.api.base*), 54
- `HtmlField` (class in *vstutils.api.fields*), 44
- `HTMLField` (class in *vstutils.models.fields*), 36
- ## I
- `id` (*vstutils.models.BModel* attribute), 34
- `ImageBaseSizeValidator` (class in *vstutils.api.validators*), 48
- `ImageHeightValidator` (class in *vstutils.api.validators*), 48
- `ImageOpenValidator` (class in *vstutils.api.validators*), 48

ImageResolutionValidator (class in *vstutils.api.validators*), 49
 ImageValidator (class in *vstutils.api.validators*), 49
 ImageWidthValidator (class in *vstutils.api.validators*), 49

K

KVExchanger (class in *vstutils.utils*), 80

L

lazy_translate() (in module *vstutils.utils*), 86
 list_test() (*vstutils.tests.BaseTestCase* method), 75
 list_to_choices() (in module *vstutils.utils*), 86
 ListModel (class in *vstutils.models.custom_model*), 35
 Lock (class in *vstutils.utils*), 80
 Lock.AcquireLockException, 81

M

Manager (class in *vstutils.models*), 34
 MaskedField (class in *vstutils.api.fields*), 44
 model_lock_decorator (class in *vstutils.utils*), 86
 ModelHandlers (class in *vstutils.utils*), 81
 models (*vstutils.tests.BaseTestCase* attribute), 75
 ModelAndViewSet (class in *vstutils.api.base*), 54
 module

- vstutils.api.actions*, 56
- vstutils.api.base*, 51
- vstutils.api.decorators*, 55
- vstutils.api.endpoint*, 67
- vstutils.api.fields*, 38
- vstutils.api.filter_backends*, 64
- vstutils.api.filters*, 60
- vstutils.api.responses*, 61
- vstutils.api.serializers*, 51
- vstutils.api.validators*, 48
- vstutils.middleware*, 62
- vstutils.models*, 31
- vstutils.models.custom_model*, 35
- vstutils.models.decorators*, 34
- vstutils.models.fields*, 36
- vstutils.models.queryset*, 34
- vstutils.tasks*, 66
- vstutils.tests*, 72
- vstutils.utils*, 78

MultipleFieldFile (class in *vstutils.models.fields*), 36

MultipleFileDescriptor (class in *vstutils.models.fields*), 36

MultipleFileField (class in *vstutils.models.fields*), 37

MultipleFileMixin (class in *vstutils.models.fields*), 37

MultipleImageField (class in *vstutils.models.fields*), 37

MultipleImageFieldFile (class in *vstutils.models.fields*), 37

MultipleNamedBinaryFileInJsonField (class in *vstutils.api.fields*), 44

MultipleNamedBinaryFileInJSONField (class in *vstutils.models.fields*), 37

MultipleNamedBinaryImageInJsonField (class in *vstutils.api.fields*), 44

MultipleNamedBinaryImageInJSONField (class in *vstutils.models.fields*), 37

N

name (*vstutils.tasks.TaskClass* property), 66

name_filter() (in module *vstutils.api.filters*), 61

NamedBinaryFileInJsonField (class in *vstutils.api.fields*), 45

NamedBinaryFileInJSONField (class in *vstutils.models.fields*), 38

NamedBinaryImageInJsonField (class in *vstutils.api.fields*), 45

NamedBinaryImageInJSONField (class in *vstutils.models.fields*), 38

nested_allow_check() (*vstutils.api.base.GenericViewSet* method), 54

nested_view (class in *vstutils.api.decorators*), 55

O

ObjectHandlers (class in *vstutils.utils*), 81

operate() (*vstutils.api.endpoint.EndpointViewSet* method), 67

P

paged() (*vstutils.models.queryset.BQuerySet* method), 34

Paginator (class in *vstutils.utils*), 82

PasswordField (class in *vstutils.api.fields*), 45

patch() (*vstutils.tests.BaseTestCase* class method), 75

patch_field_default() (*vstutils.tests.BaseTestCase* class method), 75

PhoneField (class in *vstutils.api.fields*), 45

post() (*vstutils.api.endpoint.EndpointViewSet* method), 67

post_execute() (*vstutils.utils.Executor* method), 79

pre_execute() (*vstutils.utils.Executor* method), 79

pre_save() (*vstutils.models.fields.MultipleFileMixin* method), 37

put() (*vstutils.api.endpoint.EndpointViewSet* method), 68

Q

QrCodeField (class in *vstutils.api.fields*), 45

R

raise_context (class in *vstutils.utils*), 86

`raise_context_decorator_with_default`
(class in `vstutils.utils`), 86

`random_name()` (`vstutils.tests.BaseTestCase` method), 76

`RatingField` (class in `vstutils.api.fields`), 45

`ReadOnlyModelViewSet` (class in `vstutils.api.base`), 54

`redirect_stdany` (class in `vstutils.utils`), 87

`RedirectCharField` (class in `vstutils.api.fields`), 46

`RedirectFieldMixin` (class in `vstutils.api.fields`), 46

`RedirectIntegerField` (class in `vstutils.api.fields`), 46

`register_view_action` (class in `vstutils.models.decorators`), 34

`RegularExpressionValidator` (class in `vstutils.api.validators`), 50

`RelatedListField` (class in `vstutils.api.fields`), 46

`request_handler()` (`vstutils.middleware.BaseMiddleware` method), 63

`resize_image()` (in module `vstutils.api.validators`), 50

`resize_image_from_to()` (in module `vstutils.api.validators`), 50

`run()` (`vstutils.tasks.TaskClass` method), 66

S

`save()` (`vstutils.models.fields.MultipleFieldFile` method), 36

`SecretFileInString` (class in `vstutils.api.fields`), 47

`SecurePickling` (class in `vstutils.utils`), 82

`SelectRelatedFilterBackend` (class in `vstutils.api.filter_backends`), 64

`send_mail()` (in module `vstutils.utils`), 87

`send_template_email()` (in module `vstutils.utils`), 87

`send_template_email_handler()` (in module `vstutils.utils`), 87

`serializer_class` (`vstutils.api.endpoint.EndpointViewSet` attribute), 68

`serializer_class_retrieve` (`vstutils.api.base.FileResponseRetrieveMixin` attribute), 52

`SimpleAction` (class in `vstutils.api.actions`), 59

`std_codes` (`vstutils.tests.BaseTestCase` attribute), 76

`stdout` (`vstutils.utils.Executor.CalledProcessError` property), 79

`subaction()` (in module `vstutils.api.decorators`), 56

T

`TaskClass` (class in `vstutils.tasks`), 66

`TextareaField` (class in `vstutils.api.fields`), 47

`tmp_file` (class in `vstutils.utils`), 88

`tmp_file_context` (class in `vstutils.utils`), 88

`translate()` (in module `vstutils.utils`), 88

U

`UnhandledExecutor` (class in `vstutils.utils`), 83

`UptimeField` (class in `vstutils.api.fields`), 47

`URLHandlers` (class in `vstutils.utils`), 83

`UrlQueryStringValidator` (class in `vstutils.api.validators`), 50

V

`versioning_class` (`vstutils.api.endpoint.EndpointViewSet` attribute), 68

`ViewCustomModel` (class in `vstutils.models.custom_model`), 36

`VSTCharField` (class in `vstutils.api.fields`), 47

`VSTFilterBackend` (class in `vstutils.api.filter_backends`), 64

`VSTSerializer` (class in `vstutils.api.serializers`), 51

`vstutils.api.actions`
module, 56

`vstutils.api.base`
module, 51

`vstutils.api.decorators`
module, 55

`vstutils.api.endpoint`
module, 67

`vstutils.api.fields`
module, 38

`vstutils.api.filter_backends`
module, 64

`vstutils.api.filters`
module, 60

`vstutils.api.responses`
module, 61

`vstutils.api.serializers`
module, 51

`vstutils.api.validators`
module, 48

`vstutils.middleware`
module, 62

`vstutils.models`
module, 31

`vstutils.models.custom_model`
module, 35

`vstutils.models.decorators`
module, 34

`vstutils.models.fields`
module, 36

`vstutils.models.queryset`
module, 34

`vstutils.tasks`
module, 66

`vstutils.tests`
module, 72

`vstutils.utils`

module, [78](#)

W

`working_handler()` (*vstutils.utils.Executor method*),
[80](#)

`write()` (*vstutils.utils.tmp_file method*), [88](#)

`write_output()` (*vstutils.utils.Executor method*), [80](#)

`WYSIWYGField` (*class in vstutils.api.fields*), [48](#)

`WYSIWYGField` (*class in vstutils.models.fields*), [38](#)