Vulyk Documentation

Release 0.5.1

Dmytro Hambal

Jun 18, 2020

CONTENTS

Contents:

CHAPTER

ONE

CROWDSOURCING PLATFORM FOR VARIOUS TASKS

1.1 License

- Free software: BSD license
- Documentation: https://vulyk.readthedocs.org.

1.2 What is it?

We have a lot of tasks that can only be done manually. Those includes digitizing of scanned assets declarations, manual verification of results produced by different NLP software, stalking and so on. So, we decided to generalize those tasks a bit and build a platform with basic support for crowdsourcing using knowledge and chunks of code from the unshred.it project. We don't use anything extraordinary: Flask, MongoDB, Bootstrap, jQuery etc (all trademarks are the property of their respective owners, don't forget that).

1.3 How it's built

Vulyk itself is a platform that can be stuffed with various plugins (check this two for example).

Vulyk providing basic facilities to manage tasks, users, has simple but effective system of groups and permissions, also can load tasks in format of json lines and export results. For admin purposes we have a nice (not really) CLI tool that gives admin an access to users management, tasks management, results management, stats, etc.

Vulyk is also doing dirty job to collect assets for plugin, to provide registration/login via social networks for end users and has leaderboard.

1.4 What for?

- digitize assets declarations of ukrainian officials for Declarations project
- improve the current state of Ukrainian NLP by creating a simple and robust solution for various NLP tasks that require human input.
- process and manually verify existing results from PullEnti (by Konstantin Kuznetsov) and morphological analyzer (by Andriy Rysin, Mariana Romanyshyn, et al.).
- build tagged corpora of different kinds using manually processed results.
- for the sake of The Great Justice of course!

1.5 But, God, how?

We provide some kind of playground, where any interested (or procrastination-addicted) person could spend some time crunching different tasks. Site will show you, mr. Solver, some tasks of a given type, which you'll need to solve.

Leaderboard is already in place!

1.6 How could I participate?

You just need to contact me mr_hambal@outlook.com or @dchaplinsky via chaplinsky.dmitry@gmail.com. One day we'll find one brave heart who'll create a list of issues so the process will be simplified. But not now...

1.7 Running it locally

You'll need MongoDB, Python 3.5+ and virtualenv and with little bit of instructions you'll be able to run the Beast (with two real plugins)!

First of all, check out all required components:

```
mkdir vulyk
git clone https://github.com/mrgambal/vulyk.git
git clone https://github.com/hotsyk/vulyk-declaration.git
git clone https://github.com/hotsyk/vulyk-tagging.git
```

Then create virtual environment and install all three of them there in editable mode (unfortunately we don't have any of them released on PyPI yet)

```
mkdir sandbox
cd sandbox
virtualenv venv && source venv/bin/activate
pip install -e ../vulyk
pip install -e ../vulyk-declaration
pip install -e ../vulyk-tagging
```

Then let's set things up. Edit local_settings.py and add some stuff into it:

```
# This one only works to log in via http://localhost:5000
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = ''
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = ''

# This one works for both, http://localhost:5000 and http://127.0.0.1:5000
SOCIAL_AUTH_TWITTER_KEY = ''
SOCIAL_AUTH_TWITTER_SECRET = ''

# This one only works to log in via http://localhost:5000
SOCIAL_AUTH_FACEBOOK_KEY = ''
SOCIAL_AUTH_FACEBOOK_SECRET = ''

# This one only works to log in via http://localhost:5000
SOCIAL_AUTH_VK_OAUTH2_KEY = ''
SOCIAL_AUTH_VK_OAUTH2_SECRET = ''


MONGODB_SETTINGS = {
    'DB': 'vulyk',
}

ENABLED_TASKS = {
    'vulyk_declaration': 'DeclarationTaskType',
    'vulyk_tagging': 'TaggingTaskType',
}
```

You'll need to register you localhost app in one of social networks (and fill corresponding credentials in local_settings.py!) to make it work locally.

Then you should be able to init the app using CLI, load some tasks and run it locally.

```
cp `which manage.py` .  # FUgly, I know!
python ./manage.py  init declaration_task tagging_task
```

That'll create default user group and give users of this group an access to two task types that we've installed before.

Then:

```
python ./manage.py db load declaration_task --batch 01_declaration decl_tasks.json
python ./manage.py db load tagging_task --batch 01_tagging tagging_tasks.json
```

And finally you should create run.py and put some stuff into it:

```
from vulyk.app import app

if __name__ == '__main__':
    app.run()
```

```
python run.py
```

Then open http://localhost:5000 and you are set!

Easy, isn't it?! Well, we'll smooth some rough edges soon, we promise.

CHAPTER

TWO

INSTALLATION

At the command line:

```
$ easy_install vulyk
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv vulyk
$ pip install vulyk
```

CHAPTER

THREE

USAGE

To use Vulyk in a project:

```
from vulyk.app import app

    if __name__ == '__main__':
        app.config.from_object('local_settings')
        app.run(host='0.0.0.0', port=5000)
```

As long as we use `python-social-auth` in vulyk you have to specify credentials for some auth providers supported:

```
-*- coding: utf8 -*-
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = '<your key here>'
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = '<your secret here>'

SOCIAL_AUTH_TWITTER_KEY = '<your key here>'
SOCIAL_AUTH_TWITTER_SECRET = '<your secret here>'

SOCIAL_AUTH_FACEBOOK_KEY = '<your key here>'
SOCIAL_AUTH_FACEBOOK_SECRET = '<your secret here>'

SOCIAL_AUTH_VK_OAUTH2_KEY = '<your key here>'
```

Needless to say that your settings must contain a dict of plugins (task types) you allow:

```
ENABLED_TASKS = {
    "vulyk_declaration": "DeclarationTaskType"
}
```

Two more options may be put in in case if want to use Flask-Collect alongside with Flask-Assets capabilities to get all your static files collected in single directory:

```
# path to the directory you want to get static collected in
COLLECT_STATIC_ROOT = "/var/lib/www-data/vulyk-static"
# our own Storage for Flask-Collect. Allows collecting files from plugin
# subfolders.
COLLECT_STORAGE = 'vulyk.ext.storage'
```

3.1 Import data

Data loading that's been created to add some life to your plugin should be performed via embedded CLI tool. To plug CLI into an application I create file `control.py` with similar content:

```
#!/usr/bin/env python
from vulyk.control import cli

if __name__ == '__main__':
    cli()
```

Firstly we need to initiate some bootstrap data:

```
$ chmod +x control.py
$ ./control.py  init <task_type_1> <task_type_2>
```

Having that done we're able to load tasks from datafiles (also supports gzip and bz2 archives with data). Datafile should contain a bunch of valid JSON-objects with arbitary content with line-separators in between. Loading process may be initiated by calling:

```
./control.py db load <task_type> --batch "<batch_name>" <name or wildcard>.js
# or
./control.py db load <task_type> --batch "<batch_name>" <name or wildcard>.bz2
# or
./control.py db load <task_type> --batch "<batch_name>" <name or wildcard>.gz
```

Task type – is an internal name of type that will be assigned to your tasks. Batch describes just a category of tasks you may want to be in there to simplify management and stats collecting (optional). You could omit batch name, thus all tasks you load will get 'default' batch specified in settings.

CHAPTER

FOUR

WRITING YOUR PLUGIN FOR VULYK

Vulyk plugins are the python package, installable in the same environment where Vulyk is installed.

Your plugin for Vulyk should follow the standard structure and rules of writing Python package.

To create new plugin you can use http://cookiecutter.readthedocs.org/en/latest/ [[!insert link]] app with special template [[!link to plugin template]], by

4.1 Plugin structure

- **name_of_plugin**
  - __init__.py
  - settings.py
  - **models**
    - __init__.py
    - task_types.py
    - tasks.py
  - **static**
    - images
    - scripts
    - styles
  - **templates**
    - *.html

Main entrance point to plugin will be `AbstractTaskType.get_task()`

4.2 Configuration

After plugin is installed in Vulyk environment, you need to enable it in Vulyk settings, by adding plugin's name to ENABLED_PLUGINS and name of the task to ENABLED_TASKS.

After Vulyk restart, plugin will be enabled.

CHAPTER

FIVE

VULYK

5.1 vulyk package

5.1.1 Subpackages

vulyk.admin package

Submodules

vulyk.admin.models module

class vulyk.admin.models.**AuthModelView**(*model*, *name=None*, *category=None*, *endpoint=None*, *url=None*, *static_folder=None*, *menu_class_name=None*, *menu_icon_type=None*, *menu_icon_value=None*)

Bases: flask_admin.contrib.mongoengine.view.ModelView

Model view that requires authentication and admin status Comes with useful extra of wysiwyg field

action_view()
Mass-model action view.

ajax_lookup()

ajax_update()
Edits a single column of a record in list view.

api_file_view()

create_view()
Create model view

delete_view()
Delete model view. Only POST method is allowed.

details_view()
Details model view

edit_view()
Edit model view

export(*export_type*)

extra_js = ['//cdn.ckeditor.com/4.7.1/standard/ckeditor.js']

index_view()
List view

is_accessible() → bool
Override this method to add permission checks.

Flask-Admin does not make any assumptions about the authentication system used in your application, so it is up to you to implement it.

By default, it will allow access for everyone.

Module contents

vulyk.blueprints package

Subpackages

vulyk.blueprints.gamification package

Subpackages

vulyk.blueprints.gamification.core package

Submodules

vulyk.blueprints.gamification.core.events module

Here go all types of events to happen during the project's lifecycle.

By default all 'on task done' events add some non-zero amount of coins and points. Achievements and levels are conditional things.

Another type of events is donations to some funds. These ones must contain a link to a fund and negative amount of coins, along with zero points, no achievements (at least – for now) and no level changes.

exception vulyk.blueprints.gamification.core.events.**InvalidEventException**
Bases: BaseException

Represents all possible errors during new event construction.

class vulyk.blueprints.gamification.core.events.**Event**(*timestamp: datetime.datetime*, *user: vulyk.models.user.User*, *answer: Optional[vulyk.models.tasks.AbstractAnswer]*, *points_given: decimal.Decimal*, *coins: decimal.Decimal*, *achievements: List*, *acceptor_fund: Optional[vulyk.blueprints.gamification.core.foundations.*
level_given: Optional[int], *viewed: bool*)
Bases: object

Generic gamification system event representation.

Could reflect different type of events: task is done, user is being given points/money/achievements/new level etc, user donates coins to some fund.

acceptor_fund

achievements

answer

classmethod build(*timestamp: datetime.datetime*, *user: vulyk.models.user.User*, *answer: Optional[vulyk.models.tasks.AbstractAnswer]*, *points_given: decimal.Decimal*, *coins: decimal.Decimal*, *achievements: List*, *acceptor_fund: Optional[vulyk.blueprints.gamification.core.foundations.Fund]*, *level_given: Optional[int]*, *viewed: bool*)

Fabric method

Return type *Event*

coins

level_given

points_given

timestamp

to_dict(*ignore_answer=False*) → Dict[str, Union[List, int, str]]

Could be used as a source for JSON or any other representation format :param ignore_answer: Shows if the method should return answer as a part of a dict :type ignore_answer: bool

Returns Dict-ized object view

Return type Dict[str, Union[List, int, str]]

user

viewed

class vulyk.blueprints.gamification.core.events.**NoAchievementsEvent**(*timestamp: datetime.datetime*, *user: vulyk.models.user.User*, *answer: Optional[vulyk.models.tasks.Abstra*, *points_given: decimal.Decimal*, *coins: decimal.Decimal*, *viewed: bool*)

Bases: *vulyk.blueprints.gamification.core.events.Event*

Regular 'on task done' event that contains no new level or achievements assigned to user.

acceptor_fund

achievements

answer

coins

level_given

points_given

timestamp

user

viewed

class vulyk.blueprints.gamification.core.events.**AchievementsEvent**(*timestamp: datetime.datetime*, *user: vulyk.models.user.User*, *answer: Optional[vulyk.models.tasks.AbstractA*, *points_given: decimal.Decimal*, *coins: decimal.Decimal*, *achievements: List*, *viewed: bool*)

Bases: *vulyk.blueprints.gamification.core.events.Event*

Regular 'on task done' event that contains only new achievements assigned to user.

acceptor_fund

achievements

answer

coins

level_given

points_given

timestamp

user

viewed

class vulyk.blueprints.gamification.core.events.**AchievementsLevelEvent**(*timestamp: datetime.datetime*, *user: vulyk.models.user.User*, *answer: vulyk.models.tasks.AbstractAns*, *points_given: decimal.Decimal*, *coins: decimal.Decimal*, *achievements: List*, *level_given: Optional[int]*, *viewed: bool*)

Bases: *vulyk.blueprints.gamification.core.events.Event*

Regular 'on task done' event that contains both level and achievements assigned to user.

acceptor_fund

achievements

answer

coins

level_given

points_given

timestamp

user

viewed

class vulyk.blueprints.gamification.core.events.**LevelEvent**(*timestamp: datetime.datetime*, *user: vulyk.models.user.User*, *answer: Optional[vulyk.models.tasks.AbstractAnswer]*, *points_given: decimal.Decimal*, *coins: decimal.Decimal*, *level_given: Optional[int]*, *viewed: bool*)

Bases: *vulyk.blueprints.gamification.core.events.Event*

Regular 'on task done' event that contains only new level assigned to user.

acceptor_fund

achievements

answer

coins

level_given

points_given

timestamp

user

viewed

class vulyk.blueprints.gamification.core.events.**DonateEvent**(*timestamp: datetime.datetime*, *user: vulyk.models.user.User*, *coins: decimal.Decimal*, *acceptor_fund: vulyk.blueprints.gamification.core.foundations.F*

Bases: *vulyk.blueprints.gamification.core.events.Event*

Regular 'on donate' event that contains negative amount of coins and a fund user donated to. *Viewed*-field is set to True as user itself triggers the action.

acceptor_fund

achievements

answer

coins

level_given

points_given

timestamp

user

viewed

vulyk.blueprints.gamification.core.foundations module

Foundations

class vulyk.blueprints.gamification.core.foundations.**Fund**(*fund_id: str*, *name: str*, *description: str*, *site: str*, *email: str*, *logo: io.IOBase*, *donatable: bool*)

Bases: `object`

Outer representation of different foundations we should keep: donatable or not they are.

description

donatable

email

id

logo

name

site

to_dict() → Dict[str, str]
Could be used as a source for JSON or any other representation format

Returns Dict-ized object view

Return type dict

vulyk.blueprints.gamification.core.parsing module

All available parsers, that convert raw representation could be received from any external source, are and should be kept here.

class vulyk.blueprints.gamification.core.parsing.**JsonRuleParser**
Bases: *vulyk.blueprints.gamification.core.parsing.RuleParser*

Basic JSON parser.

static parse(*json_string: str*) → *vulyk.blueprints.gamification.core.rules.Rule*
Actually perform parsing from JSON-encoded string to an actual rule.

Parameters **json_string** (*str*) – JSON dict with all the data about the achievement.

Returns Fully parsed rule object.

Return type *Rule*

Exception RuleParsingException

class vulyk.blueprints.gamification.core.parsing.**RuleParser**
Bases: object

Just a stub in case if we want to extend parsing sources.

exception vulyk.blueprints.gamification.core.parsing.**RuleParsingException**
Bases: Exception

Basic exception for all types of rule parsing errors

vulyk.blueprints.gamification.core.queries module

The factory of achievements. Classes below allow us to query data source we use as a source of truth.

class vulyk.blueprints.gamification.core.queries.**MongoRuleExecutor**
Bases: object

Simple query runner that uses pymongo's BaseQuerySet instance to aggregate user's stats.

static achieved(*user_id: bson.objectid.ObjectId*, *rule: vulyk.blueprints.gamification.core.rules.Rule*, *collection: mongoengine.queryset.base.BaseQuerySet*) → bool

Determines if given user has achieved a new prize.

Parameters

- **user_id** (*bson.ObjectId*) – Current user ID
- **rule** (*Rule*) – Rule to be applied
- **collection** (*BaseQuerySet*) – WorkSession querySet

Returns True if user stats comply to the rule

Return type bool

class vulyk.blueprints.gamification.core.queries.**MongoRuleQueryBuilder**(*rule: vulyk.blueprints.gamification.c*

Bases: *vulyk.blueprints.gamification.core.queries.RuleQueryBuilder*

Implementation of RuleQueryBuilder, bound to MongoDB.

build_for(*user_id: bson.objectid.ObjectId*) → List[Dict[str, Dict]]

Prepares a pipeline of actions to be passed to MongoDB Aggregation Framework.

Parameters **user_id** (*bson.ObjectId*) – Current user ID

Returns List of actions to be done within the aggregation.

Return type List[Dict[str, Dict]]

class vulyk.blueprints.gamification.core.queries.**RuleQueryBuilder**

Bases: object

Abstract query builder. Takes an instance of Rule-type and converts its properties into fully fledged queries to a data source.

build_for(*user_id: bson.objectid.ObjectId*) → List[Dict[str, Dict]]

Prepares a pipeline of actions to be passed to data source.

Parameters **user_id** (*bson.ObjectId*) – Current user ID

Returns List of actions to be done.

Return type List[Dict[str, Dict]]

vulyk.blueprints.gamification.core.rules module

class vulyk.blueprints.gamification.core.rules.**ProjectRule**(*rule_id: str*, *task_type_name: str*, *badge: str*, *name: str*, *description: str*, *bonus: int*, *tasks_number: int*, *days_number: int*, *is_weekend: bool*, *is_adjacent: bool*)

Bases: *vulyk.blueprints.gamification.core.rules.Rule*

Container for project specific rules.

badge

bonus

description

classmethod from_rule(*rule:* vulyk.blueprints.gamification.core.rules.Rule, *task_type_name: str*) → *vulyk.blueprints.gamification.core.rules.Rule*

Factory method to extend regular Rule and promote it to ProjectRule

Parameters

- **rule** (Rule) – Basic Rule instance to copy info from
- **task_type_name** (*str*) – ID of the project/task type.

Returns Fully formed ProjectRule

Return type *ProjectRule*

name

property task_type_name

to_dict() → Dict[str, Union[str, int, bool]]

Could be used as a source for JSON or any other representation format

Returns Dict-ized object view

Return type Dict[str, Union[str, int, bool]]

class vulyk.blueprints.gamification.core.rules.**Rule**(*rule_id: str*, *badge: str*, *name: str*, *description: str*, *bonus: int*, *tasks_number: int*, *days_number: int*, *is_weekend: bool*, *is_adjacent: bool*)

Bases: object

Base class for all rule containers. Those rules may be interpreted like the following:

- you closed n tasks
- you closed n tasks in m days
- you closed n tasks in weekends
- you've been working for m weekends
- you've been working for m days in a row
- you've been working for m weekends in a row

Also additional bonuses to be given after user gets the achievement are included as well as achievement and its badge is.

badge

bonus

property days_number

description

property id

property is_adjacent

property is_weekend

property limit
The vital characteristic of the rule: the limit member should surpass to get the achievement. At current stage the property relies upon two values: tasks done by user and days he spent working on these tasks. The priority is following: if tasks number is specified, it always supersedes days.

E.g.: if rule has *tasks_number=20* and *days_number=7* – limit is 20. We take the number of tasks done in 7 days, and compare it to 20. Otherwise, if only *days_number=7* is specified, limit is 7. We group all tasks were done not earlier than 7 days ago, group them by day and check if 7 items is returned from aggregation pipeline.

Returns The value to be compared to aggregation results.

Return type int

name

property tasks_number

to_dict() → Dict[str, Union[str, int, bool]]
Could be used as a source for JSON or any other representation format

Returns Dict-ized object view

Return type Dict[str, Union[str, int, bool]]

exception vulyk.blueprints.gamification.core.rules.**RuleValidationException**
Bases: Exception

Basic exception class for all types of rule validation violations

vulyk.blueprints.gamification.core.state module

The package contains user state model and everything that will belong to this part of domain.

class vulyk.blueprints.gamification.core.state.**UserState**(*user: vulyk.models.user.User*, *level: int*, *points: decimal.Decimal*, *actual_coins: decimal.Decimal*, *potential_coins: decimal.Decimal*, *achievements: list*, *last_changed: datetime.datetime*)
Bases: object

An aggregation of all the stuff user has gotten working on projects so far.

achievements

actual_coins

last_changed

level

points

potential_coins

to_dict() → dict
Could be used as a source for JSON or any other representation format

Returns Dict-ized object view

Return type dict

user

Module contents

A cask full of wonders: all of rule parsing and gamification logic.

vulyk.blueprints.gamification.models package

Submodules

vulyk.blueprints.gamification.models.events module

Contains all DB models related to game events.

class vulyk.blueprints.gamification.models.events.**EventModel**(**args*, ***values*)
Bases: flask_mongoengine.Document

Database-specific gamification system event representation

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

acceptor_fund
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a Document which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the LazyReferenceField.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a ListField of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

achievements

A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

---

Note: Required means it cannot be empty - as the default for ListFields is []

---

classmethod amount_of_money_donated(*user: Optional[*vulyk.models.user.User*]*) → float

Amount of money donated by current user or total donations if None passed.

Parameters **user** (*Optional[User]*) – User instance

Returns Amount of money

Return type float

classmethod amount_of_money_earned(*user: Optional[*vulyk.models.user.User*]*) → float

Amount of money earned by current user or total amount earned if None is passed.

Parameters **user** (*Optional[User]*) – User instance

Returns Amount of money

Return type float

answer

A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a `Document` which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the `LazyReferenceField`.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

classmethod batches_user_worked_on(*user:* vulyk.models.user.User) → Generator[vulyk.models.tasks.Batch, None, None]

Returns an iterable of deduplicated batches user has worked on before.

Parameters **user** (User) – User instance

Returns Iterator over batches

Return type Generator[*Batch*, None, None]

coins

Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

classmethod count_of_tasks_done_by_user(*user:* vulyk.models.user.User) → int

Number of tasks finished by current user

Parameters **user** (User) – User instance

Returns Count of tasks done

Return type int

classmethod from_event(*event:* vulyk.blueprints.gamification.core.events.Event)

Event to DB-specific model converter.

Parameters **event** (Event) – Source event instance

Returns New full-bodied model instance

Return type *EventModel*

classmethod get_all_events(*user:* vulyk.models.user.User) → Iterator

Returns aggregated and sorted generator of events (achievements & level-ups) user'd been given.

Parameters **user** (User) – The user to extract events for

Returns A generator of events in ascending chronological order.

Return type Generator[*Event*, None, None]

classmethod get_unread_events(*user:* vulyk.models.user.User) → Generator[vulyk.blueprints.gamification.core.events.Event, None, None]

Returns aggregated and sorted list of generator (achievements & level-ups) user'd been given but hasn't checked yet.

Parameters **user** (User) – The user to extract events for

Returns A generator of events in ascending chronological order.

Return type Generator[*Event*, None, None]

id
A field wrapper around MongoDB's ObjectIds.

level_given
32-bit integer field.

classmethod mark_events_as_read(*user:* vulyk.models.user.User) → None
Mark all user events as viewed

Parameters **user** (User) – The user to mark unseed events as viewed

Returns Nothing. None. Empty. Long Gone

Return type None

objects = []

points_given
Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

timestamp
ComplexDateTimeField handles microseconds exactly instead of rounding like DateTimeField does.

Derives from a StringField so you can do *gte* and *lte* filtering by using lexicographical comparison when filtering / sorting strings.

The stored string has the following format:

YYYY,MM,DD,HH,MM,SS,NNNNNN

Where NNNNNN is the number of microseconds of the represented *datetime*. The , as the separator can be easily modified by passing the *separator* keyword when initializing the field.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

New in version 0.5.

to_event() → *vulyk.blueprints.gamification.core.events.Event*
DB-specific model to Event converter.

Returns New Event instance

Return type *Event*

user
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a Document which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the LazyReferenceField.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).

- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

viewed
Boolean field type.

New in version 0.1.2.

vulyk.blueprints.gamification.models.foundations module

Contains all DB models related to foundations we donate or we rely on

class vulyk.blueprints.gamification.models.foundations.**FundFilterBy**
Bases: enum.Enum

The intent of this enum is to represent filtering policies.

DONATABLE = 1

NON_DONATABLE = 2

NO_FILTER = 0

class vulyk.blueprints.gamification.models.foundations.**FundModel**(**args*, ***values*)
Bases: flask_mongoengine.Document

Database-specific foundation representation

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

description
A unicode string field.

donatable
Boolean field type.

New in version 0.1.2.

email
A field that validates input as an email address.

New in version 0.4.

classmethod find_by_id(*fund_id: str*) → Optional[*vulyk.blueprints.gamification.core.foundations.Fund*]
Convenience method that returns an optional fund by its ID.

Parameters **fund_id** (*str*) – Fund's ID

Returns Found instance or none.

Return type Optional[*Fund*]

classmethod from_fund(*fund:* vulyk.blueprints.gamification.core.foundations.Fund)
Fund to DB-specific model converter.

Parameters **fund** (Fund) – Current Fund instance

Returns New FundModel instance

Return type *FundModel*

classmethod get_funds(*filter_by: vulyk.blueprints.gamification.models.foundations.FundFilterBy = <FundFilterBy.NO_FILTER: 0>*) → Iterator[*vulyk.blueprints.gamification.core.foundations.Fund*]
Returns an enumeration of funds available using the filtering policy passed.

Parameters **filter_by** (FundFilterBy) – Filtering policy

Returns Lazy enumeration of funds available.

Return type Iterator[*Fund*]

id
A unicode string field.

logo
A Image File storage field.

Parameters

- **size** – max size to store images, provided as (width, height, force) if larger, it will be automatically resized (ex: size=(800, 600, True))
- **thumbnail_size** – size to generate a thumbnail, provided as (width, height, force)

New in version 0.6.

name
A unicode string field.

objects = []

site
A unicode string field.

to_fund() → *vulyk.blueprints.gamification.core.foundations.Fund*
DB-specific model to Fund converter.

Returns New Fund instance

Return type *Fund*

vulyk.blueprints.gamification.models.rules module

class vulyk.blueprints.gamification.models.rules.**RuleModel**(**args*, ***values*)
Bases: flask_mongoengine.Document

Database-specific rule representation

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

badge
A unicode string field.

bonus
32-bit integer field.

days_number
32-bit integer field.

description
A unicode string field.

classmethod from_rule(*rule:* vulyk.blueprints.gamification.core.rules.Rule)
Rule to DB-specific model converter.

Parameters **rule** (Rule) – Current Rule instance

Returns New RuleModel instance

Return type *RuleModel*

classmethod get_actual_rules(*skip_ids: List[str]*, *rule_filter: vulyk.blueprints.gamification.models.rules.RuleFilter*, *is_weekend: bool*) → Iterator[*vulyk.blueprints.gamification.core.rules.Rule*]
Prepare the list of rules to apply. Cutting off is possible by using different tiny yet smart heuristics.

Parameters

- **skip_ids** (*List[str]*) – A list of rules to be skipped for any reason
- **rule_filter** (*RuleFilter*) – Prepared query container.
- **is_weekend** (*bool*) – If today is a working day, there is no reason to check for rules that are weekend-backed.

Returns An array of rules to be checked and assigned.

Return type Iterator[*Rule*]

id
A unicode string field.

is_adjacent
Boolean field type.

New in version 0.1.2.

is_weekend
Boolean field type.

New in version 0.1.2.

name
A unicode string field.

objects = []

task_type_name
A unicode string field.

tasks_number
32-bit integer field.

to_rule() → *vulyk.blueprints.gamification.core.rules.Rule*
DB-specific model to Rule converter.

Returns New Rule instance

Return type *Rule*

class vulyk.blueprints.gamification.models.rules.**AllRules**
Bases: vulyk.blueprints.gamification.models.rules.RuleFilter

Should return all rules.

to_query() → mongoengine.queryset.visitor.Q
Prepares a filtering query.

Returns Prepared query instance.

Return type Q

class vulyk.blueprints.gamification.models.rules.**ProjectAndFreeRules**(*task_type_name: str*)
Bases: vulyk.blueprints.gamification.models.rules.RuleFilter

Should return all rules related to a certain project along with project-agnostic ones.

to_query() → mongoengine.queryset.visitor.Q
Prepares a filtering query.

Returns Prepared query instance.

Return type Q

class vulyk.blueprints.gamification.models.rules.**StrictProjectRules**(*task_type_name: str*)
Bases: vulyk.blueprints.gamification.models.rules.RuleFilter

Should return all rules related only to a certain project.

to_query() → mongoengine.queryset.visitor.Q
Prepares a filtering query.

Returns Prepared query instance.

Return type Q

vulyk.blueprints.gamification.models.state module

Contains all DB models related to aggregated user state within the game

class vulyk.blueprints.gamification.models.state.**StateSortingKeys**
Bases: enum.Enum

The intent of this enum is to keep different sorting options shortcuts.

ACTUAL_COINS = 1

LEVEL = 3

POINTS = 0

POTENTIAL_COINS = 2

class vulyk.blueprints.gamification.models.state.**UserStateModel**(**args*, ***values*)
Bases: flask_mongoengine.Document

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

achievements
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

actual_coins
Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

classmethod from_state(*state:* vulyk.blueprints.gamification.core.state.UserState)
UserState to DB-specific model converter.

Parameters **state** (UserState) – Source user state

Returns New model instance

Return type *UserStateModel*

classmethod get_or_create_by_user(*user:* vulyk.models.user.User) → *vulyk.blueprints.gamification.core.state.UserState*
Returns an existing UserStateModel instance bound to the user, or a new one if didn't exist before.

Parameters **user** (User) – Current User model instance

Returns UserState instance

Return type *UserState*

classmethod get_top_users(*limit: int*, *sort_by:* vulyk.blueprints.gamification.models.state.StateSortingKeys) → Iterator[*vulyk.blueprints.gamification.core.state.UserState*]

Enumerates over top users basing on passed sorting criteria and limiting number. The yield sorting is descending.

Parameters

- **limit** (*int*) – Top limit
- **sort_by** (StateSortingKeys) – Criteria enumeration item

Returns An iterator

Return type Iterator[*UserState*]

id

A field wrapper around MongoDB's ObjectIds.

last_changed

ComplexDateTimeField handles microseconds exactly instead of rounding like DateTimeField does.

Derives from a StringField so you can do *gte* and *lte* filtering by using lexicographical comparison when filtering / sorting strings.

The stored string has the following format:

YYYY,MM,DD,HH,MM,SS,NNNNNN

Where NNNNNN is the number of microseconds of the represented *datetime*. The , as the separator can be easily modified by passing the *separator* keyword when initializing the field.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

New in version 0.5.

level

32-bit integer field.

objects = []

points

Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

potential_coins

Fixed-point decimal number field. Stores the value as a float by default unless *force_string* is used. If using floats, beware of Decimal to float conversion (potential precision loss)

Changed in version 0.8.

New in version 0.3.

to_state() → *vulyk.blueprints.gamification.core.state.UserState*

DB-specific model to UserState converter.

It isn't supposed to dig out what's been buried once, yet this method is really useful for tests.

Returns New UserState instance

Return type *UserState*

classmethod transfer_coins_to_actual(*uid: bson.objectid.ObjectId*, *amount: decimal.Decimal*) → bool

Parameters

- **uid** (*ObjectId*) – Current user ID
- **amount** (*Decimal*) – Money amount

Returns True if success

Return type bool

classmethod update_state(*diff:* vulyk.blueprints.gamification.core.state.UserState) → None
Prepares and conducts an atomic update query from passed diff.

Parameters **diff** (UserState) – State object contains values that are to be changed only.

Return type None

user
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a Document which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the LazyReferenceField.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a ListField of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

classmethod withdraw(*user:* vulyk.models.user.User, *amount: decimal.Decimal*) → bool
Reflects money withdrawal from current account.

Parameters

- **user** (User) – User to perform the action on.
- **amount** (*Decimal*) – Money amount (ONLY positive)

Raise

- RuntimeError – if amount is negative

Returns True if needed amount was successfully withdrawn

Return type bool

vulyk.blueprints.gamification.models.task_types module

class vulyk.blueprints.gamification.models.task_types.**AbstractGamifiedTaskType**(*settings: Dict[str, Any]*)

Bases: *vulyk.models.task_types.AbstractTaskType*

to_dict() → Dict[str, Union[str, Dict, None]]

Prepare simplified dict that contains basic info about the task type + information on next open batch

Returns distilled dict with basic info

Return type Dict[str, Union[str, Optional[Dict]]]

Module contents

vulyk.blueprints.gamification.listeners module

vulyk.blueprints.gamification.listeners.**track_events**(*sender: object*, *answer:* vulyk.models.tasks.AbstractAnswer) → None

The most important gear of the gamification module.

Parameters

- **sender** (*object*) – Sender
- **answer** (AbstractAnswer) – Current finished task's answer instance

Return type None

vulyk.blueprints.gamification.listeners.**get_actual_rules**(*state:* vulyk.blueprints.gamification.core.state.UserState, *task_type_name: str*, *now: datetime.datetime*) → Iterator[*vulyk.blueprints.gamification.core.rules.Rule*]

Returns a list of eligible rules.

Parameters

- **state** (UserState) – Current state
- **task_type_name** (*str*) – The task's project
- **now** (*datetime*) – Timestamp to check for weekends

Returns Iterator of Rule instances

Return type Iterator[*Rule*]

vulyk.blueprints.gamification.services module

Services module

class vulyk.blueprints.gamification.services.**DonationResult**
Bases: enum.Enum

An enumeration to represent different results of donation process.

BEGGAR = 2

ERROR = 666

LIAR = 3

STINGY = 1

SUCCESS = 0

class vulyk.blueprints.gamification.services.**DonationsService**(*user:* vulyk.models.user.User, *fund_id:* *str*, *amount:* *decimal.Decimal*)
Bases: object

Class hides mildly complex donation logic from cruel outer world.

donate() → *vulyk.blueprints.gamification.services.DonationResult*

Perform a donation process:

- check if there is enough active money to spare
- check if fund exists
- try to decrease amount of coins on current account
- create and save an event

Returns One of *DonationResult* enum values: SUCCESS - everything went okay; STINGY - you tried to donate nothing; BEGGAR - you have less money than tried to spare; LIAR - you passed non-existent fund; ERROR - sh*t happened :(.

Return type *DonationResult*

class vulyk.blueprints.gamification.services.**StatsService**
Bases: object

Facade, the root stats collector to provide aggregated data from different repositories.

classmethod projects_count(*user:* vulyk.models.user.User) → int
Aggregate the number of batches in which user has done at least single tiny task.

Parameters **user** (User) – Current user

Returns Number of batches

Return type int

classmethod state_of_user(*user:* vulyk.models.user.User) → Optional[*vulyk.blueprints.gamification.core.state.UserState*]
Return current state of given user

Returns Object which holds aggregated values

on user current state for the registered user and None otherwise :rtype: Optional[UserState]

classmethod tasks_done_by_user(*user:* vulyk.models.user.User) → int
Returns optional of the total number of tasks were finished by user.

Parameters **user** (User) – Current user

Returns Number of tasks done or None

Return type int

classmethod total_money_donated() → float
Count and return total amount of money donated by all users to all foundations

Returns Total amount in UAH

Return type float

classmethod total_money_donated_by_user(*user:* vulyk.models.user.User) → float
Count and return total amount of money donated by current user

Returns Total amount in UAH

Return type float

classmethod total_money_earned() → float
Count and return total amount of money earned by all users on all tasks

Returns Total amount in UAH

Return type float

classmethod total_number_of_open_tasks() → int
Count and return number of open tasks in all projects

Returns Number of open tasks

Return type int

classmethod total_number_of_users() → int
Count and return number of users registered in the system

Returns Number of active users

Return type int

classmethod total_time_for_user(*user:* vulyk.models.user.User) → int
Count and return number of hours, spent on the site doing tasks.

Parameters **user** (User) – Current user

Returns Full hours

Return type int

Module contents

The core of gamification sub-project.

vulyk.bootstrap package

Submodules

vulyk.bootstrap._assets module

vulyk.bootstrap._assets.**init**(*app*) → None
Bundle projects assets.

Parameters **app** (*flask.Flask*) – Main application instance

vulyk.bootstrap._social_login module

Module contains stuff related to interoperability with PSA.

vulyk.bootstrap._social_login.**init_social_login**(*app*, *db*) → None
Login manager initialisation.

Parameters

- **app** (*flask.Flask*) – Main application instance
- **db** (*flask_mongoengine.MongoEngine*) – MongoDB wrapper instance

vulyk.bootstrap._tasks module

Module contains code that performs plugins initialisation.

vulyk.bootstrap._tasks.**init_plugins**(*app*) → Dict[str, *vulyk.models.task_types.AbstractTaskType*]
Extracts modules (task types) from global configuration.

Parameters **app** (*flask.Flask*) – Current Flask application instance

Returns Dictionary with instantiated TaskType objects

Return type dict[str, *AbstractTaskType*]

Module contents

Project bootstrapper.

Contains code not to be used directly after the initialization.

vulyk.bootstrap.**init_app**(*name*)

Parameters **name** (*str*) – application alias

Returns Bootstrapped cached application instance

Return type flask.Flask

vulyk.bootstrap.**init_plugins**(*app*) → Dict[str, *vulyk.models.task_types.AbstractTaskType*]
Extracts modules (task types) from global configuration.

Parameters **app** (*flask.Flask*) – Current Flask application instance

Returns Dictionary with instantiated TaskType objects

Return type dict[str, *AbstractTaskType*]

vulyk.cli package

Submodules

vulyk.cli.admin module

Package contains CLI tools implementation related to admins

vulyk.cli.admin.**list_admin**() → None
Outputs a list of emails of administrators.

vulyk.cli.admin.**toggle_admin**(*email*, *state*) → None
Toggles admin state of given user.

Parameters

- **email** (*str*) – email address.
- **state** (*bool*) – State we want to set.

vulyk.cli.batches module

Package contains CLI tools related to managing batches of tasks.

vulyk.cli.batches.**add_batch**(*batch_id: str*, *count: int*, *task_type: vulyk.models.task_types.AbstractTaskType*, *default_batch: str*, *batch_meta: Optional[Dict] = None*) → None
Updates or creates new batch after loading new dataset. Only default batch may be extended.

Parameters

- **batch_id** (*str*) – Name of the batch
- **count** (*int*) – Number of tasks to load
- **task_type** (AbstractTaskType) – Type of tasks loaded into the batch
- **default_batch** (*str*) – Name of the default batch
- **batch_meta** (*Optional[Dict]*) – User params to override default batch meta

Raise click.BadParameter

vulyk.cli.batches.**batches_list**() → List[str]

Returns List of batches IDs to validate CLI input

Return type List[str]

vulyk.cli.batches.**validate_batch**(*ctx*, *param: str*, *value: str*, *default_batch: str*) → str
Refuses your attempts to add tasks to existing batch (except 'default')

Parameters

- **ctx** – Click context
- **param** (*str*) – Name of parameter (*batch*)
- **value** (*str*) – Value of *batch* parameter
- **default_batch** (*str*) – Name of the default batch

Returns the value if is valid

Return type str

Raise click.BadParameter

vulyk.cli.db module

vulyk.cli.db.**export_reports**(*task_id:* vulyk.models.task_types.AbstractTaskType, *path: str*, *batch: str*, *closed: bool*) → None

vulyk.cli.db.**load_tasks**(*task_type:* vulyk.models.task_types.AbstractTaskType, *path: str*, *batch: str*) → int

Parameters **batch** (*str*) – Batch ID tasks should be loaded into

Returns Number of tasks loaded

Return type int

vulyk.cli.db.**open_anything**(*filename: str*)

vulyk.cli.groups module

vulyk.cli.groups.**add_task_type**(*gid: str*, *task_type: str*) → None
Appends task type to the list of allowed ones of certain group

Parameters

- **gid** (*str*) – Group's symbolic code
- **task_type** (*str*) – Task type symbolic code

Raises **click.BadParameter** – if wrong *gid* has been passed

vulyk.cli.groups.**assign_to**(*username: str*, *gid: str*) → None
Assigns a group to user

Parameters

- **gid** (*str*) – Group's symbolic code
- **username** (*str*) – Username of member

Raises **click.BadParameter** – if wrong *gid* or ` username` has been passed

vulyk.cli.groups.**get_groups_ids**() → List[str]
Returns list of groups codes

:rtype : list[str]

vulyk.cli.groups.**list_groups**() → Generator[str, None, None]
Generates list of group representation strings

:rtype : _generator[str]

vulyk.cli.groups.**new_group**(*gid: str*, *description: str*) → None
Creates new group

Parameters

- **gid** (*str*) – Group's symbolic code
- **description** (*str*) – Short description (optional)

Raises **click.BadParameter** – if wrong *id* has been passed

vulyk.cli.groups.**remove_group**(*gid: str*) → None
Delete existing group

Parameters **gid** (*str*) – Group's symbolic code

Raises **click.BadParameter** – if wrong *id* has been passed

vulyk.cli.groups.**remove_task_type**(*gid: str*, *task_type: str*) → None
Removes task type from the list of allowed types of specified group

Parameters

- **gid** (*str*) – Group's symbolic code
- **task_type** (*str*) – Task type symbolic code

Raises **click.BadParameter** – if wrong *gid* has been passed

vulyk.cli.groups.**resign**(*username: str*, *gid: str*) → None
Excludes user from specified group

Parameters

- **gid** (*str*) – Group's symbolic code
- **username** (*str*) – Username of member

Raises **click.BadParameter** – if wrong *gid* or ` username` has been passed

vulyk.cli.groups.**validate_id**(*ctx*, *param: str*, *value: str*) → str
Allows group code to consist only of letters/numbers/underscore

Parameters

- **ctx** – Click context
- **param** (*str*) – Name of parameter (*id*)
- **value** (*str*) – Value of *id* parameter

Returns true if value passes

Return type str

Raises **click.BadParameter** –

vulyk.cli.stats module

vulyk.cli.stats.**batch_completeness**(*batch_name: str*, *task_type: str*) → collections.OrderedDict

Gathers completeness stats on every batch using 2 metrics: - actually completed tasks - actual reports to total planned amount ratio (tasks * redundancy)

Parameters

- **batch_name** (*str*) – Optional name of batch to filter by
- **task_type** (*str*) – Optional name of task type to filter by

Return type OrderedDict

Module contents

The package consists of modules hat provide support for different aspects of project's CLI.

vulyk.cli.**is_initialized**(*default_key='default'*) → bool

The method checks whether the default group has been created already or has not.

Parameters **default_key** (*str*) – Default group ID

Returns A boolean flag

Return type bool

vulyk.cli.**project_init**(*allowed_types*) → None

The method reassures that a default group is already available, otherwise it will be created and passed task types are to be made accessible to the group.

Parameters **allowed_types** (*list[str]*) – Task type names to be allowed to default group.

vulyk.ext package

Submodules

vulyk.ext.leaderboard module

class vulyk.ext.leaderboard.**LeaderBoardManager**(*task_type_name: str*, *answer_model: vulyk.models.tasks.AbstractAnswer*, *user_model: type*)

Bases: object

get_leaderboard(*limit: int*) → List[Dict[str, Union[*vulyk.models.user.User*, int]]]

Find users who contributed the most

Parameters **limit** (*int*) – number of top users to return

Returns List of dicts {user: user_obj, freq: count}

Return type List[Dict[str, Union[*User*, int]]]

get_leaders() → List[Tuple[bson.objectid.ObjectId, int]]

Return sorted list of tuples (user_id, tasks_done)

Returns list of tuples (user_id, tasks_done)

Return type List[Tuple[ObjectId, int]]

vulyk.ext.storage module

Copy files from all static folders to root folder.

class vulyk.ext.storage.**Storage**(*collect*, *verbose: bool = False*)
Bases: flask_collect.storage.file.Storage

Storage that copies static files.

vulyk.ext.worksession module

class vulyk.ext.worksession.**WorkSessionManager**(*work_session_model: Type[U]*)
Bases: object

This class is responsible for accounting of work-sessions. Every time we give a task to user, a new session record is being created. If user skips the task, we mark it as skipped and delete the session. When user finishes the task, we close the session having added the timestamp of the event. Thus we're able to perform any kind of data mining and stats counting using the data later.

Could be overridden in plugins.

U = ~U

delete_work_session(*task:* vulyk.models.tasks.AbstractTask, *user_id:* *bson.objectid.ObjectId*) → None
Deletes current WorkSession if skipped.

Parameters

- **task** (AbstractTask) – Given task
- **user_id** (*ObjectId*) – ID of user, who skips a task

Raises WorkSessionLookUpError – session is not found; WorkSessionUpdateError – can not delete the session

end_work_session(*task:* vulyk.models.tasks.AbstractTask, *user_id:* *bson.objectid.ObjectId*, *answer:* vulyk.models.tasks.AbstractAnswer) → None
Ends given WorkSession for given user. This is the route for correctly finished tasks: given session to be marked as closed and a timestamp of the event to be saved.

Parameters

- **task** (AbstractTask) – Given task
- **user_id** (*ObjectId*) – ID of user, who finishes a task
- **answer** (AbstractAnswer) – Given answer

Raises WorkSessionLookUpError – session is not found; WorkSessionUpdateError – can not close the session

record_activity(*task:* vulyk.models.tasks.AbstractTask, *user_id: bson.objectid.ObjectId*, *seconds: int*) → None
Update an activity counter. The intention is to find out how much time was actually spent working on the task, excluding sexting, brewing coffee and jogging.

Parameters

- **task** (AbstractTask) – The task the session belongs to.
- **user_id** (*ObjectId*) – ID of current user
- **seconds** (*int*) – User was active for

Raises WorkSessionLookUpError – session is not found; WorkSessionUpdateError – can not update the session

start_work_session(*task:* vulyk.models.tasks.AbstractTask, *user_id:* *bson.objectid.ObjectId*) → None

Starts new WorkSession for given user. By default we use *datetime.now* in the underlying model to save in *start_time* field.

A user should finish a certain task only once, that's why we perform an upsert below.

Parameters

- **task** (*AbstractTask*) – Given task
- **user_id** (*ObjectId*) – ID of user, who gets new task

Raises WorkSessionUpdateError – can not start a session

Module contents

vulyk.models package

Submodules

vulyk.models.exc module

Module contains all exception classes could be raised during work with the DB.

exception vulyk.models.exc.**TaskImportError**
Bases: Exception

exception vulyk.models.exc.**TaskNotFoundError**
Bases: Exception

exception vulyk.models.exc.**TaskSaveError**
Bases: Exception

exception vulyk.models.exc.**TaskPermissionError**
Bases: Exception

exception vulyk.models.exc.**TaskSkipError**
Bases: Exception

exception vulyk.models.exc.**TaskValidationError**
Bases: Exception

exception vulyk.models.exc.**WorkSessionLookUpError**
Bases: Exception

exception vulyk.models.exc.**WorkSessionUpdateError**
Bases: Exception

vulyk.models.stats module

Module contains all models used to keep some metadata we could use to perform any kind of analysis.

class vulyk.models.stats.**WorkSession**(**args*, ***values*)
Bases: flask_mongoengine.Document

Class which represents a timespan during which user was working on the task Also it stores links to every entity involved.

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

activity
64-bit integer field. (Equivalent to IntField since the support to Python2 was dropped)

answer
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a Document which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the LazyReferenceField.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a ListField of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

end_time
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

classmethod get_total_user_time_approximate(*user_id: bson.objectid.ObjectId*) → int
Aggregated time spent doing tasks on all projects by certain user. As the source we use approximate values of start time and end time. Might be useful if no proper time accounting is done on frontend.

Parameters **user_id** (*ObjectId*) – User ID

Returns Total time (in seconds)

Return type int

classmethod get_total_user_time_precise(*user_id: bson.objectid.ObjectId*) → int
Aggregated time spent doing tasks on all projects by certain user. As the source we use more precise value of activity field.

Parameters **user_id** (*ObjectId*) – User ID

Returns Total time (in seconds)

Return type int

id
A field wrapper around MongoDB's ObjectIds.

objects = []

start_time
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

task
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a `Document` which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the `LazyReferenceField`.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

task_type
A unicode string field.

user
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a `Document` which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the `LazyReferenceField`.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

vulyk.models.task_types module

Module contains all models related to task type (plugin root) entity.

class vulyk.models.task_types.**AbstractTaskType**(*settings: Dict[str, Any]*)
Bases: object

The main entity in the application. Contains all the logic we need to handle task emission/accounting.

Could be overridden in plugins to fit your needs.

The simplest and most common scenario of being overridden is to have own task type name and description to separate your tasks from any other.

CSS_ASSETS = []

JS_ASSETS = []

answer_model = None

property description
Explicit description of the plugin.

Returns Plugin description.

Return type str

export_reports(*batch: str*, *closed: bool = True*, *qs=None*) → Generator[Dict[str, Any], None, None]
Exports results. IO is left out of scope here as well

Parameters

- **batch** (*str*) – Certain batch to extract
- **closed** (*bool*) – Specify if we need to export only closed tasks reports
- **qs** (*QuerySet*) – Queryset, an optional argument. Default value is QS that exports all tasks with amount of answers > redundancy

Returns Generator of lists of dicts with results

Return type Generator[Dict[str, Any], None, None]

get_leaderboard(*limit: int = 10*) → List[Dict]
Find users who contributed the most

Parameters **limit** (*int*) – number of top users to return

Returns List of dicts {user: user_obj, freq: count}

Return type List[Dict]

get_leaders() → List[Tuple[bson.objectid.ObjectId, int]]
Return sorted list of tuples (user_id, tasks_done)

Returns list of tuples (user_id, tasks_done)

Return type List[Tuple[ObjectId, int]]

get_next(*user:* vulyk.models.user.User) → Dict
Finds given user a new task and starts new WorkSession

Parameters **user** (*User*) – an instance of User model

Returns Prepared dictionary of model, or empty dictionary

Return type Dict

helptext_template = ''

import_tasks(*tasks: List[Dict]*, *batch: Optional[AnyStr]*) → None
Imports tasks from an iterable over dicts io is left out of scope here.

Parameters

- **tasks** (*List[Dict]*) – An iterable over dicts
- **batch** (*Optional[AnyStr]*) – Batch ID (optional)

Raise TaskImportError

property name
Human-readable name of the plugin.

Returns Name of the task type.

Return type str

on_task_done(*user:* vulyk.models.user.User, *task_id: AnyStr*, *result: Dict[str, Any]*) → None
Saves user's answers for a given task. Assumes that user is eligible for this kind of tasks.

Parameters

- **task_id** (*AnyStr*) – Given task ID
- **user** (*User*) – an instance of User model who provided an answer
- **result** (*Dict[str, Any]*) – Task solving result

Raises TaskSaveError - in case of general problems

Raises TaskValidationError - in case of validation problems

record_activity(*user_id: Union[AnyStr, bson.objectid.ObjectId]*, *task_id: AnyStr*, *seconds: int*) → None
Increases the counter of activity for current user in given task.

Parameters

- **user_id** (*Union[AnyStr, ObjectId]*) – ID of user, who gets new task
- **task_id** (*AnyStr*) – Current task
- **seconds** (*int*) – User was active for

Raises TaskSkipError, TaskNotFoundError

redundancy = 3

skip_task(*task_id: AnyStr*, *user:* vulyk.models.user.User)
Marks given task as a skipped by a given user Assumes that user is eligible for this kind of tasks

Parameters

- **task_id** (*AnyStr*) – Given task ID
- **user** (*User*) – an instance of User model who provided an answer

Raises TaskSkipError, TaskNotFoundError

task_model = None

property task_type_meta
Dict with task type metadata (freeform dict)

Returns project specific metadata

Return type Dict[str, Any]

template = ''

to_dict() → Dict[str, Any]
Prepare simplified dict that contains basic info about the task type.

Returns distilled dict with basic info

Return type Dict[str, Any]

type_name = ''

property work_session_manager
Returns current instance of WorkSessionManager used in the task type.

Returns Active WorkSessionManager instance.

Return type *WorkSessionManager*

vulyk.models.tasks module

Module contains all models directly related to the main entity - tasks.

class vulyk.models.tasks.**AbstractAnswer**(**args*, ***values*)
Bases: flask_mongoengine.Document

This is AbstractTask model. You need to inherit it in your model

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

classmethod answers_numbers_by_tasks(*task_ids: List[str]*) → Dict[bson.objectid.ObjectId, int]
Groups answers, filtered by tasks they belong to, by user and count number of answers for every user.

Parameters **task_ids** (*List[str]*) – List of tasks IDs

Returns Map having user IDs as keys and answers numbers as values

Return type Dict[ObjectId, int]

as_dict() → Dict[str, Dict]
Converts the model-instance into a safe that will include also task and user.

Return type Dict[str, Dict]

property corrections
Returns whole amount of actions/corrections given by user in this particular answer.

Returns Count of corrections in this answer

Return type int

created_at
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

created_by
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a `Document` which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the `LazyReferenceField`.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

id
A field wrapper around MongoDB's ObjectIds.

objects = []

result
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

New in version 0.3.

Changed in version 0.5: - Can now handle complex / varying types of data

task
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a `Document` which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if

you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the `LazyReferenceField`.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

task_type
A unicode string field.

class vulyk.models.tasks.**AbstractTask**(**args*, ***values*)
Bases: `flask_mongoengine.Document`

This is AbstractTask model. You need to inherit it in your model

exception DoesNotExist
Bases: `mongoengine.errors.DoesNotExist`

exception MultipleObjectsReturned
Bases: `mongoengine.errors.MultipleObjectsReturned`

as_dict() → Dict[str, Any]
Converts the model-instance into a safe and lightweight dictionary.

Return type Dict[str, Any]

batch
A reference to a document that will be automatically dereferenced on access (lazily).

Note this means you will get a database I/O access everytime you access this field. This is necessary because the field returns a `Document` which precise type can depend of the value of the *_cls* field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it *pk* field which is already known before dereference). To solve this you should consider using the `LazyReferenceField`.

Use the *reverse_delete_rule* to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an *InvalidDocumentError* will be raised if trying to set on one of these Document / Field types.

The options are:

- DO_NOTHING (0) - don't do anything (default).
- NULLIFY (1) - Updates the reference to null.
- CASCADE (2) - Deletes the documents associated with the reference.
- DENY (3) - Prevent the deletion of the reference object.
- PULL (4) - Pull the reference from a `ListField` of references

Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

```
class Org(Document):
    owner = ReferenceField('User')

class User(Document):
    org = ReferenceField('Org', reverse_delete_rule=CASCADE)

User.register_delete_rule(Org, 'owner', DENY)
```

Changed in version 0.5: added *reverse_delete_rule*

closed
Boolean field type.

New in version 0.1.2.

id
A unicode string field.

classmethod ids_in_batch(*batch:* vulyk.models.tasks.Batch) → List[str]
Collects IDs of all tasks that belong to certain batch.

Parameters **batch** (`Batch`) – Batch instance

Returns List of IDs

Return type List[str]

objects = []

task_data
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

New in version 0.3.

Changed in version 0.5: - Can now handle complex / varying types of data

task_type
A unicode string field.

users_count
32-bit integer field.

users_processed
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

users_skipped
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

class vulyk.models.tasks.**Batch**(**args*, ***values*)
Bases: flask_mongoengine.Document

Helper category to group tasks.

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

batch_meta
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

New in version 0.3.

Changed in version 0.5: - Can now handle complex / varying types of data

closed
Boolean field type.

New in version 0.1.2.

id
A unicode string field.

objects = []

classmethod task_done_in(*batch_id: str*) → *vulyk.models.tasks.BatchUpdateResult*
Increment needed values upon a task from the batch is done. In case if all tasks are finished – close the batch.

Parameters **batch_id** (*str*) – Batch ID

Returns Aggregate which represents complex effect of the method

Return type *BatchUpdateResult*

task_type
A unicode string field.

tasks_count
32-bit integer field.

tasks_processed
32-bit integer field.

class vulyk.models.tasks.**BatchUpdateResult**(*success*, *closed*)
Bases: tuple

property closed
Alias for field number 1

property success
Alias for field number 0

vulyk.models.user module

Module contains all models related to member entity.

class vulyk.models.user.**Anonymous**
Bases: flask_login.mixins.AnonymousUserMixin

name = 'Anonymous'

class vulyk.models.user.**Group**(**args*, ***values*)
Bases: flask_mongoengine.Document

Class was introduced to serve the permissions purpose

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

allowed_types
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

description
A unicode string field.

id
A unicode string field.

objects = []

class vulyk.models.user.**User**(**args*, ***values*)
Bases: flask_mongoengine.Document, flask_login.mixins.UserMixin

Main model for member entity.

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

active
Boolean field type.

New in version 0.1.2.

admin
Boolean field type.

New in version 0.1.2.

as_dict() → Dict[str, str]
Converts the model-instance into a safe dict that will include some basic info about member.

Returns Reduced set of information about member.

Return type Dict[str, str]

email
A unicode string field.

classmethod get_by_id(*user_id: str*) → Optional[flask_mongoengine.Document]

Parameters **user_id** (*str*) – Needed user ID

Returns The user

Return type Optional[*User*]

get_stats(*task_type*) → Dict[str, int]
Returns member's stats containing the number of tasks finished and the position in the global rank.

Parameters **task_type** (vulyk.models.task_types.AbstractTaskType) – Task type instance.

Returns Dictionary that contains total finished tasks count and the position in the global rank.

Return type Dict[str, int]

groups
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

---

Note: Required means it cannot be empty - as the default for ListFields is []

---

id
A field wrapper around MongoDB's ObjectIds.

is_active() → bool

is_admin() → bool

is_eligible_for(*task_type: str*) → bool
Check that user is authorized to work with this tasks type

Parameters **task_type** (*str*) – Tasks type name

Returns True if user is eligible

Return type bool

Raises AssertionError - if no *task_type* specified

last_login
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

name
A unicode string field.

objects = []

password
A unicode string field.

classmethod pre_save(*sender: Type*, *document: flask_mongoengine.Document*, ***kwargs: Dict*) → flask_mongoengine.Document
A signal handler which will put a new member into a default group if any hasn't been assigned yet.

Parameters

- **sender** (*Type*) – Type of signal emitter.
- **document** (User) – New instance of User model.
- **kwargs** (*Dict*) – Additional parameters

Returns Modified User instance.

Return type *User*

processed
32-bit integer field.

username
A unicode string field.

Module contents

5.1.2 Submodules

5.1.3 vulyk.app module

Die Hauptstadt of our little project. Just a usual Flask application.

5.1.4 vulyk.control module

vulyk.control.**abort_if_false**(*ctx*, *param*, *value*) → None

5.1.5 vulyk.settings module

5.1.6 vulyk.signals module

5.1.7 vulyk.utils module

Every project must have a package called *utils*.

vulyk.utils.**chunked**(*iterable: Iterator*, *n: int*) → Generator[Tuple, None, None]
Returns chunks of n length of iterable

If len(iterable) % n != 0, then the last chunk will have length less than n.

Example:

```
>>> chunked([1, 2, 3, 4, 5], 2)
[(1, 2), (3, 4), (5,)]
```

Parameters

- **iterable** (*Iterator*) – Source we need to chop up.
- **n** (*int*) – Slice length

Returns Sequence of tuples of given size.

Return type Generator[Tuple, None, None]

vulyk.utils.**get_tb**() → Dict
Returns traceback of the latest exception caught in 'except' block

Returns traceback of the most recent exception

vulyk.utils.**get_template_path**(*app: flask.app.Flask*, *name: str*) → str
Finds the path to the template.

Parameters

- **app** (*flask.Flask*) – Flask application instance.
- **name** (*str*) – Name of the template.

Returns Full path to the template.

Return type str

vulyk.utils.**json_response**(*result: Dict*, *errors: Optional[Iterator] = None*, *status: int = <HTTPStatus.OK: 200>*) → flask.wrappers.Response
Handy helper to prepare unified responses.

Parameters

- **result** (*Dict*) – Data to be sent
- **errors** (*Optional[Iterator]*) – List (set, tuple, dict) of errors
- **status** (*int*) – Response http-status

Returns Jsonified response

Return type flask.Response

vulyk.utils.**resolve_task_type**(*type_id: str*, *tasks: Dict*, *user:* vulyk.models.user.User)
Looks for *type_id* in TASK_TYPES map.

Parameters

- **type_id** (*str*) – ID of the TaskType in the map.
- **tasks** (*Dict[str, vulyk.models.task_types.AbstractTaskType]*) – map of *task type id -> task type instance*
- **user** (User) – Current user.

Returns Correct TaskType instance or throws an exception.

Return type *vulyk.models.task_types.AbstractTaskType*

5.1.8 Module contents

CHAPTER

SIX

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/mrgambal/vulyk/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

6.1.4 Write Documentation

Vulyk could always use more documentation, whether as part of the official Vulyk docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mrgambal/vulyk/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *vulyk* for local development.

1. Fork the *vulyk* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/vulyk.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv vulyk
$ cd vulyk/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 vulyk tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.4 and 3.5. Check https://travis-ci.org/mrgambal/vulyk/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_vulyk
```

CHAPTER

SEVEN

CREDITS

7.1 Development Lead

- Dmytro Hambal <mr_hambal@outlook.com>
- Dmitry Chaplinsky <chaplinsky.dmitry@gmail.com>
- Volodymyr Hotsyk <gotsyk@gmail.com>
- Alexander Sapronov <https://github.com/WarmongeR1>

7.2 Contributors

You can be here, see *Contributing*

CHAPTER

EIGHT

HISTORY

CHAPTER

NINE

0.1.0 (2015-01-12)

- First release on PyPI.

CHAPTER

TEN

0.2.0 (2015-03-31)

- minimal viable product

CHAPTER

ELEVEN

0.3.0 (2015-05-30)

- added batches (#32, #45, #47)
- added basic stats (#27, #49)
- fixed static timestamp (#42)
- added top users scoreboard (#39)
- added possibility to serve static files from wherever you want (#10)
- a lots of other little fixes and improvements. . .

CHAPTER

TWELVE

0.3.1 (2016-04-27)

- lightweight tasks randomization
- filter stats by batch name and/or task type
- now vulyk is recording all media

CHAPTER

THIRTEEN

0.4.0 (2016-05-18)

- python 3 migration
- fixes for manage script
- Move base templates to base-folder
- Added key LANGUAGE and an example translation
- Added keys SITE_LOGO and SITE_TITLE
- Added key THANKS_TASK_MESSAGE
- Fixed templates for navigate.

CHAPTER

FOURTEEN

0.5.0 (2019-03-20)

- dropped python 3.4 support
- enabled python 3.7 support
- dropped support for MongoDB prior to 3.4

CHAPTER

FIFTEEN

0.5.1 (2019-04-10)

- fixes for the documentation build process
- updated metadata

CHAPTER

SIXTEEN

0.5.2 (2020-06-08)

- update dependencies
- move the build process to GitHub Actions

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

V

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W