Kool Documentation

Release 0.0.1

Antony Orenge

Sep 11, 2017

Contents

Kool is an open source platform for online classroom management.

CHAPTER 1

User's Guide

1.1 Introduction

Currently, teachers who manage online classes use several tools with different purposes. For example, they can use one application for distributing content (mailing lists, custom websites), one application for quizzes and homework submission, one application to manage gradebooks, another for classroom discussion (forums, webconferencing or chat), and several others. The aim for this project is to consolidate the necessary functionality into one standalone solution while keeping it lightweight, easy to deploy & use, and also make it easy to add new functionality.

1.2 Getting Started

1.2.1 Prerequisites

1. Python3
2. Virtualenv
3. Pip

1.2.2 Installing Kool

To install Kool from PyPI, run:

```
$ python install kool
```

If you prefer to install from source, download the latest version from Releases. Thereafter, extract and run:

```
$ python setup.py install
```

1.2.3 Basic Usage

The current version support creation of users, courses and quizzes. It also includes a flatfile database implementation of csv storage. The database can be configured as below:

```
$ vim kool/db/flatfile/database.py
```

Under *class FlatFileDB(object):*, set default values as below:

```
DEFAULT_DB = '<enter database name>'
DEFAULT_TABLE = '<enter default table name>'
DEFAULT_STORAGE = <set default storage>
```

Once you are done with setting default database configs. You can proceed to load classes from the library. For example to initialize class User with some new user, run:

```
>>> from kool.contrib.auth import User, Group, Permission
>>> user = User(first_name='John', last_name='Doe', email='john@doe.com', password=
↪'secretpwd')
```

To make the record persistent, call save() method for it to be written to a csv file.

```
>>> user.save()
1
>>>
```

Once you have a table object, you can insert records by:

```
>>> user.insert({'first_name': 'Mary', 'last_name': 'Doe', 'email': 'mary@doe.com',
↪'password': 'secretpwd'})
2
>>>
```

To get all records stored in a table, run:

```
>>> user.all()
[{'_id': '1', 'email': 'john@doe.com', 'groups': '[]', 'password': 'bcrypt_sha256$$2b
↪$12$8XUv6hUS/Zs7Hjw8U/ArqOHdj/WeutsReeTWgchVpET7HczuMVpIi', 'is_active': 'True',
↪'last_modified': '2017-09-09 23:18:23.917851', 'first_name': 'John', 'date_created
↪': '2017-09-09 23:18:23.918017', 'last_name': 'Doe', 'permissions': '[]'}, {'_id':
↪'2', 'email': 'mary@doe.com', 'groups': '', 'password': 'secretpwd', 'is_active': '
↪', 'last_modified': '', 'first_name': 'Mary', 'date_created': '', 'last_name': 'Doe
↪', 'permissions': ''}]
>>>
```

Passwords are securely stored as a bcrypt sha 256 hash code.

To filter a specific record from a table, run:

```
>>> from kool.db.models import where
>>> user.filter(where('first_name')=='John')
[{'_id': '1', 'email': 'john@doe.com', 'groups': '[]', 'password': 'bcrypt_sha256$$2b
↪$12$8XUv6hUS/Zs7Hjw8U/ArqOHdj/WeutsReeTWgchVpET7HczuMVpIi', 'is_active': 'True',
↪'last_modified': '2017-09-09 23:18:23.917851', 'first_name': 'John', 'date_created
↪': '2017-09-09 23:18:23.918017', 'last_name': 'Doe', 'permissions': '[]'}]
>>>
```

Call the update() method to modify a record:

```
>>> user.email = 'john@gmail.com'
>>> user.update()
2
>>> user.filter(where('first_name')=='John')
[{'_id': '2', 'email': 'john@gmail.com', 'groups': '[]', 'password': 'bcrypt_sha256$
↪$2b$12$8XUv6hUS/Zs7Hjw8U/ArqOHdj/WeutsReeTWgchVpET7HczuMVpIi', 'is_active': 'True',
↪'last_modified': '2017-09-09 23:38:12.813258', 'first_name': 'John', 'date_created
↪': '2017-09-09 23:18:23.918017', 'last_name': 'Doe', 'permissions': '[]'}]
>>>
```

To delete an existing record:

```
>>> user.delete()
>>> user.filter(where('first_name')=='John')
[]
>>>
```

1.3 Advanced Usage

Kool has a *models.py* file that allows database operations to be applied on a class that inherits from it. Look at the *models* API for allowed operations.

1.3.1 Table

Table is composed of records which are made up of multiple fields. Each *Record* is identified by *rid* which is a proxy to the *_id* field in the table files. In this flat file database implementation, every table represents a file. Multiple files makeup a database. The *Table* class provides methods to perform CRUD and Query operations on its data.

To obtain a *Table* object without instantiating the class you can use the class method *table()* which takes a class as an argument and returns an equivalent *Table* object of the class:

```
>>> from kool.db.models import table
>>> from kool.contrib.auth import User
>>> user_table = table(User)
```

1.3.2 Queries

Queries allows data to be requested from a table or combination of tables. The result is generated as list of flatfile records. To query a table, you need an object of type *Table*.

Query operations include providing a condition as an argument to query methods *any()*, *filter()*, *matches()*, and *all()*. Conditions are made up by comparing values using binary operators (==, <, >, >=, <=, ~). Two ways of performing queries are:

1. Classical style:

```
>>> from kool.db.models import where
>>> user_table.filter(where('value') == True)
```

2. ORM-like style:

```
>>> from kool.db.models import Query
>>> User = Query()
>>> user_table.filter(User.first_name == 'John')
>>> user_table.filter(User['last_name'] == 'Doe')
>>> user_table.filter(User.age >= 21)
```

1.3.3 Advanced queries

Additionally, you can perform complex queries by using logical expressions to modify or combine queries as show below:

```
>>> # Logical AND:
>>> user_table.filter((User.name == 'John') & (User.age <= 26))
```

```
>>> # Logical OR:
>>> user_table.filter((User.name == 'John') | (User.name == 'Mary'))
```

Other operations that can be performed include:

Check the existence of a field:

```
>>> user_table.filter(User.first_name.exists())
```

Perform a Regular expression. The field has to match the regex:

```
>>> user_table.filter(User.first_name.matches('[aZ]*'))
```

Perform a Custom test:

```
>>> test_func = lambda s: s == 'John'
>>> user_table.filter(User.first_name.test(test_func))
```

Custom test with parameters:

```
>>> def test_func(val, m, n):
>>>     return m <= val <= n
>>> user_table.filter(User.age.test(test_func, 18, 21))
>>> user_table.filter(User.age.test(test_func, 21, 30))
```

Note: When using `&` or `|`, make sure you wrap the conditions on both sides with parentheses or Python will mess up the comparison.

CHAPTER 2

Extending Kool

2.1 How to Extend Kool

Kool library allows extending of its contrib and storage modules and modifying its behavior.

2.1.1 Contrib

The contrib module is directory structure is as shown below:

```
.
├── auth
│   ├── group.py
│   ├── hasher.py
│   ├── permission.py
│   ├── user.py
├── courses
│   ├── course.py
└── quizzes
    ├── question.py
    └── quiz.py
```

Below is an example of extending class *User* to *Student*:

```
>>> from kool.contrib.auth import User
>>> class Student(User):
...   pass
...
>>> student = Student(first_name='John', last_name='Doe', email='john@doe.com',␣
↪password='secretpwd')
>>> student.save()
1
>>>
```

2.1.2 Storage

Storage provides a way of making data persistent in the Kool library. By default, Kool supports CSV file storage. If you wish to continue storing data with the flatfile database implementation, you can extend the base Storage class and write your preferred implementation.

For example, to create a JSON file storage:

```
class JSONStorage(Storage):
    """
    Store the data in a JSON file.
    """

    def __init__(self, path, create_dirs=False, **kwargs):
        """
        Create a new instance.
        Also creates the storage file, if it doesn't exist.
        :param path: Where to store the JSON data.
        :type path: str
        """

        super(JSONStorage, self).__init__()
        touch(path, create_dirs=create_dirs)  # Create file if not exists
        self.kwargs = kwargs
        self._handle = open(path, 'r+')

    def close(self):
        self._handle.close()

    def read(self):
        # Get the file size
        self._handle.seek(0, os.SEEK_END)
        size = self._handle.tell()

        if not size:
            # File is empty
            return None
        else:
            self._handle.seek(0)
            return json.load(self._handle)

    def write(self, data):
        self._handle.seek(0)
        serialized = json.dumps(data, **self.kwargs)
        self._handle.write(serialized)
        self._handle.flush()
        self._handle.truncate()
```

Much of the storage implementation was borrowed from TinyDB. So, have a look at it for more examples.

CHAPTER 3

API Reference

3.1 API documentaiton

3.1.1 kool.contrib.auth.user

3.1.2 kool.contrib.auth.group

3.1.3 kool.contrib.auth.permission

3.1.4 kool.contrib.courses.course

3.1.5 kool.contrib.quizzes.question

3.1.6 kool.contrib.quizzes.quiz

3.1.7 kool.db.flatfile.database

3.1.8 kool.db.flatfile.queries

3.1.9 kool.db.flatfile.storages

CHAPTER 4

Additional Notes

4.1 Contributor Covenant Code of Conduct

4.1.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

4.1.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.1.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

4.1.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

4.1.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team contact at *orenge at ut dot ee*. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

4.1.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

4.2 Changelog

4.2.1 Version Numbering

Kool subscribes to Semantic Versioning guidelines as stipulated at semver.org

4.2.2 v0.0.1 (2017-09-10)

- Initial release of Kool.

4.3 Upgrading to Newer Releases

No upgrade to initial release for now. Please check later.