# QStode Documentation

***Release 0.1.17***

**Daniel Kertesz**

December 29, 2013

# Contents

Welcome to QStode documentation. This documentation is a work in progress and is far from being complete.

You can always check out a live installation visiting QStode ;)

CHAPTER 1

# User's Guide

## 1.1 Installation

QStode is a Python web application and depends on some external libraries which can be installed via setuptools; it will also need a web server capable of running WSGI applications.

### 1.1.1 Requirements

- Python >= 2.6
- MySQL 5.x database
- python-setuptools
- python-mysqldb (also packaged as **MySQL-Python**)
- WSGI server (e.g gunicorn)
- Web server
- one writable directory

You *may* also need:

- git
- python-virtualenv
- Redis database
- c/c++ compiler

Ubuntu example:

```
$ sudo apt-get install python-setuptools python-dev python-virtualenv \
    git mysql-server gunicorn build-essential
```

### 1.1.2 Getting the sources

You can download a tarball of the latest release from the releases page on GitHub, or you can *clone* the repository with git:

```
$ cd /usr/local/src
$ git clone https://github.com/piger/qstode.git
```

### 1.1.3 Installing Python packages

virtualenv is the best way to manage your installation of QStode; if you don't know what virtualenv is be sure to check out the website and learn why you should use it with your web applications deployment.

Create a virtualenv:

```
$ virtualenv /srv/qstode/env
```

Install QStode with setuptools:

```
$ source /srv/qstode/env/bin/activate
$ cd /usr/local/src/qstode
$ python setup.py install
```

### 1.1.4 Database setup

QStode need to store data in a MySQL database with **UTF-8** character encoding; an example database can be created running the following SQL commands: :

```
mysql> create database qstode character set utf8 collate utf8_bin;
mysql> create user 'qstode'@'localhost' identified by 'somepass';
mysql> grant all privileges on qstode.* to 'qstode'@'localhost';
mysql> flush privileges;
```

### 1.1.5 Configuration file

See *Configuration* for details about the configuration file.

### 1.1.6 Initial setup

You must create a configuration file in a directory readable by the WSGI server process, for example `/etc/qstode/config.py`.

To create the database tables and the admin user (remember to activate the virtualenv first!):

```
$ source /srv/qstode/env/bin/activate
$ qstode -c /etc/qstode/config.py setup
```

You can test your installation by running a local server with the command:

```
$ qstode -c /etc/qstode/config.py server
```

A **DEBUG** mode is also available:

```
$ qstode -c /etc/qstode/config.py -D server
```

### 1.1.7 Deployment

#### Deployment with uWSGI

A configuration file to run QStode with uWSGI:

```
[uwsgi]
plugin = python
virtualenv = /srv/qstode/env
module = qstode.main
callable = run_wsgi
stats = 127.0.0.1:9191
env = APP_CONFIG=/etc/qstode/config.py
threads = 4
```

A sample configuration for nginx:

```
upstream qstode_uwsgi {
    server unix:/run/uwsgi/app/qstode/socket;
}

server {
    listen 80;

    server_name example.com;

    root /srv/qstode/htdocs;

    location /static/ {
        root /usr/local/src/qstode/qstode/;
        expires 15d;
        add_header Pragma public;
        add_header Cache-Control "public, must-revalidate, proxy-revalidate";
    }

    location / {
        try_files $uri $uri/ @proxy_to_app;
    }

    location @proxy_to_app {
        uwsgi_pass qstode_uwsgi;
        uwsgi_param APP_CONFIG /etc/qstode/config.py;
        include uwsgi_params;
    }
}
```

### 1.1.8 Migration and Backup

You can backup all your data to a *JSON* file by running the `backup` command:

```
$ qstode -c /path/to/config.py backup filename.json
```

You can also import an existing backup by running the `import` command:

```
$ qstode -c /path/to/config.py import filename.json
```

After an import you must also recreate the Whoosh index; at the moment the best way is to delete any existing Whoosh directory and then index again all your content, running the `reindex` command:

```
$ qstode -c /path/to/config.py reindex
```

## 1.2 Configuration

Configuration handling in QStode is inherited from Flask, therefore it's managed by a Python script used as a configuration file.

The following configuration values have the same meaning they have in Flask (default values between parenthesis):

**DEBUG (`False`)** Enable or disable debug mode.

**SECRET_KEY** The secret key used by cryptography functions; you can use a random string or generate one with Python:

```
$ python -c 'import os; print "%r" % os.urandom(24)'
```

**SQLALCHEMY_DATABASE_URI** The URI containing the parameters for connecting to the database, in the format `drivername://username:password@address[:port]/dbname`.

Example: `mysql://my-user:s3cRet@localhost/qstode`.

---

**Note:** To use any database other then SQLite you must install the corresponding Python driver; for example to use a MySQL database you must install the `MySQL-Python` package.

---

The following configuration values are specific to QStode:

**PUBLIC_ACCESS (`True`)** Enable anonymous access to all the public pages; if set to `False` a valid user is required to browse the application.

**USER_REGISTRATION_ENABLED (`True`)** Allow anonymous users to register themselves.

**PER_PAGE (`10`)** Specify how many bookmarks to show on each page.

**FEED_NUM_ENTRIES (`15`)** Specify how many bookmarks to list in the public RSS feed.

**TAGLIST_ITEMS (`30`)** Specify how many tags to show in the Popular Tags listing.

**ENABLE_RELATED_TAGS (`True`)** Enable functions to show related tags in the *search* views.

**Warning:** The `related tag` feature is currently half-broken when using MySQL without InnoDB.

**BABEL_DEFAULT_LOCALE (`en`)** The default locale to use if no locale selector is registered.

**BABEL_DEFAULT_TIMEZONE (`UTC`)** The timezone to use for user facing dates.

**EXTRA_TEMPLATES (`[]`)** A optional **list** of directories containing Jinja2 templates that will override the built in templates.

Example: `EXTRA_TEMPLATES = [ "/srv/www/my_templates" ]`

**WHOOSH_INDEX_PATH** The directory used to store the search engine's files; must be writable by the user running QStode.

**USE_GOOGLE_FAVICON (`True`)** Enable or disable the use of Google services to display *favicons* for bookmarked sites.

---

**Note:** This feature can be disabled by paranoid users ;)

---

**REDIS_HOST, REDIS_PORT, REDIS_DB, REDIS_PASSWORD** Redis connection parameters.

**RECAPTCHA_USE_SSL** Enable/disable recaptcha through ssl.

**RECAPTCHA_PUBLIC_KEY** Recaptcha public key.

**RECAPTCHA_PRIVATE_KEY** Recaptcha private key.

CHAPTER 2

# API Reference

If you want to interact with QStode this is the place to look.

## 2.1 API

### 2.1.1 JSON data format

A bookmark in QStode is serialized to a JSON dictionary; for example:

```
{
  "results": {
    "modified_on": "2012-06-08T11:14:18",
    "title": "A caldo: che cos\u2019\u00e8 questo golpe?\u00a0|\u00a0Giap",
    "url": "http://www.wumingfoundation.com/giap/?p=8016&cpage=1&utm_source=dlvr.it&utm_medium=twitte
    "notes": "La \u201cstrategia della tensione\u201d \u00e8 sempre una strategia di controrivoluzio
    "tags": [
          "attualita'",
          "terrorismo",
          "wu-ming"
    ],
    "id": 1,
    "private": false,
    "created_on": "2012-05-20T23:47:14"
  }
}
```

The dictionary keys are:

**id** Internal ID of the bookmark.

**title** Title given to the bookmark by the owner.

**url** URL of the bookmarked item.

**tags** The list of tags assigned to the bookmark.

**notes** The optional notes field.

**private** Privacy status of the bookmark.

**created_on** Date and time of the bookmark creation.

**modified_on** Date and time of the last modification to the bookmark data.

**Note:** All timestamps are in UTC format!

### 2.1.2 API Endpoints

**GET /api/bookmarks/**

Get all Bookmarks, with pagination.

**Example request**:

```
GET /api/bookmarks/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

**Example response**:

```
HTTP/1.1 200 OK
Content-Type: text/javascript

{
  "results": {
    "bookmarks": [
      {
        "modified_on": "2013-06-27T19:06:36",
        "title": "Occupy Gezi",
        "url": "http://occupygezi.neocities.org/",
        "notes": " Timemap of the events in Turkey between June 5th and June 17th (2013) as seen here
        "tags": [
          "gezi park",
          "mappa",
          "occupy",
          "turkey"
        ],
        "id": 712,
        "private": false,
        "created_on": "2013-06-27T19:06:36"
      },
  }
}
```

**query sort** one of `date`, `user`

**query offset** offset number, default is 0

**statuscode 200** success

**statuscode 404** error

**GET /api/bookmarks/(int:** *bookmark_id*)

Retrieve a single Bookmark by the given *bookmark_id*.

**Example request**:

```
GET /api/bookmarks/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

**Example response**:

```
HTTP/1.0 200 OK
Content-Type: text/javascript

{
  "results": {
    "modified_on": "2012-06-08T11:14:18",
    "title": "A caldo: che cos\u2019\u00e8 questo golpe?\u00a0|\u00a0Giap",
    "url": "http://www.wumingfoundation.com/giap/?p=8016&cpage=1&utm_source=dlvr.it&utm_medium=twitte
    "notes": "La \u201cstrategia della tensione\u201d \u00e8 sempre una strategia di controrivoluzion
    "tags": [
              "attualita'",
              "terrorismo",
              "wu-ming"
    ],
    "id": 1,
    "private": false,
    "created_on": "2012-05-20T23:47:14"
  }
}
```

**statuscode 200** success

**statuscode 400** error processing the request

CHAPTER 3

# Additional Notes

Design notes, legal information and changelog are here for the interested.

## 3.1 Database

### 3.1.1 Notes and Miscellaneous Informations

With alembic a migration that involve dropping a column with a constraints must be handled carefully. First you must find out the name of the constraint running a command like:

```
SHOW CREATE TABLE <tablename>;
```

For example the constraint name is *bookmarks_ibfk_3*, so you can now create the alembic operation that will first drop the contraint and then the column and the table:

```
def upgrade():
    op.drop_constraint('bookmarks_ibfk_3', 'bookmarks', 'foreignkey')
    op.drop_column("bookmarks", "category_id")
    op.drop_table("categories")
```

## 3.2 Upgrading to Newer Releases

### 3.2.1 Version 0.1.20

The database schema was mercilessly altered; I suggest to backup (somehow) to a json file before upgrading QStode.

### 3.2.2 Version 0.1.17

Apply the included alembic migration with:

```
alembic upgrade head
```

Be sure to first create or update *alembic.ini* and make sure it can connect to your database.

## 3.3 Exporting data from Scuttle

### 3.3.1 Utilities

QStode includes two utilities to export bookmark data from the database of a Scuttle installation; the code was only partially tested, so use with caution and remember to backup your data first.

#### Exporting data from Scuttle

`qstode-scuttle-export` is a Python script that exports data from the database of a Scuttle installation to either a **JSON** file or a HTML file.

Access to the database is configured in a funny way; you must write a configuration file containing just one value:

```
uri = <sqlalchemy URI>
```

For example:

```
uri = mysql://my-user:s3cRet@localhost/qstode
```

Check out the `SQLALCHEMY_DATABASE_URI` parameter in the *Configuration* page for more informations about the URI format.

To run the export utility:

```
$ qstode-scuttle-export -c config.txt
```

After the operation is completed you will find a file named `scuttle-export.json` in your current directory.

#### Importing data from a Scuttle JSON export file

To import a **JSON** backup file in QStode you must run the following command:

```
$ qstode -c /etc/qstode/config.py scuttle-import <backup-filename.json>
```

You have to specify the path to the main configuration file of your QStode installation.

Please note that if you are importing data to a fresh installation of QStode you will have to run the `setup` command before running the import commands:

```
$ qstode -c /etc/qstode/config.py setup
```

### 3.3.2 What to do if you have utf-8 data in a latin1 database

See details about character encoding of your database:

```
show create database `dbname`;
```

It should say that the default character encoding is `latin`; please note that any column could have a different character encoding associated.

Now export data ensuring that MySQL won't change the encoding:

```
mysqldump --default-character-set=utf8 --opt -u `user` -p `dbname` \
              > db-latin1.sql
```

Replace (this sound silly, I know) `latin1` with `utf8` in your SQL dump:

```
replace "CHARSET latin1" "CHARSET utf8" \
              "SET NAMES latin1" "SET NAMES utf8" \
              < db-latin1.sql > db-utf8.sql
```

The **replace** command is part of MySQL.

Now you can create a new database specifying `utf8` as the default character set:

```
mysql --default-character-set=utf8 -u root -p
> create database `mydb` character set utf8 collate utf8_bin;
```

Note that `collate utf8_bin` is optional.

It could also be necessary to change the collation of some tables; for example in scuttle we have the table `sc_tags` where a key is made by the `bookmark_id` and the `tag_id`, but with the default collation we have ‘e’ == ‘è’:

```
mysql> SELECT 'e' = 'è' COLLATE utf8_general_ci;
+-------------------------------------+
| 'e' = 'è' COLLATE utf8_general_ci   |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

To fix this you can edit the SQL dump and add `COLLATE=utf8_bin` to the `CREATE TABLE` statement of the problematic table.

## 3.4 QStode Changelog

### 3.4.1 Version 0.1.20

This version was released mostly because `0.1.19` contained some ugly bugs.

- Added `backup` and `import` command to backup and restore the contents of the database to and from a *json* file.
- Fixed an old issue related to orphan **Tag** objects in the database.
- Usual round of bug fixes.

### 3.4.2 Version 0.1.19

QStode source code is now hosted on GitHub!

- Miscellaneous bug fixes.
- Code refactoring everywhere.
- Removed some unused functionalities.
- Prototype of a Redis based indexing daemon.

### 3.4.3 Version 0.1.18

Cleanup release, released on July, 1st 2013.

- Categories are gone.

- Big localization effort, almost complete.
- Internal refactoring.
- New CSS theme :-)
- Documentation of installation process and API use.
- Changed environment variable name from *QSTODE_WEB_CONFIG* to a more generic *APP_CONFIG*.

### 3.4.4 Version 0.1.17

Some new features and a pretty new theme.

- Support for OpenID login with an existing account.
- Better support for localization.

## 3.5 License

QStode is license under a three clause BSD License.

### 3.5.1 Authors

QStode is written and maintained by Daniel Kertesz and some unnamed contributors:

- Daniel Kertesz <daniel@spatof.org>

### 3.5.2 QStode License

CHAPTER 4

# Indices and tables

- *genindex*
- *modindex*
- *search*

# HTTP Routing Table

## /api