wger Workout Manager Documentation

Release 1.9 alpha1

Roland Geider

May 05, 2017

Contents

wger is a free, open source web application that manages your exercises and personal workouts, weight and nutrition. It can also be used as a simple gym management utility, providing different administrative roles (trainer, manager, etc.). It offers a REST API as well, for easy integration with other projects and tools. It is written with python/django and uses jQuery and some D3js for charts.

For more details and a live system, refer to the project's site: https://github.com/wger-project/wger

This documentation is intended for developers and administrators of the software.

CHAPTER 1

Installation and development

Getting started

You can get a local instance of wger installed in a couple of minutes.

It is recommended to install a development instance or start a docker image if you just want to try the application on your server or PC. All the following steps are performed on a debian based linux distribution. If your setup differs (e.g. in Red Hat based distros the package names are slightly different) you will need to change the steps as appropriate.

The application is compatible and regularly tested with

- sqlite, postgres
- python 2.7, 3.4, 3.5 and 3.6

You might also want to take a look at the *Other changes* section for other changes you might want to do to your local instance such as Terms of Service or contact page.

Note: Please note that all the steps related to upgrading the database or downloading external JS dependencies mentioned in the *Tips and tricks* section in the development page apply to all the installation options.

These are the necessary packages for both development and production (node and npm are only used to download JS and CSS libraries):

```
sudo apt-get install nodejs nodejs-legacy npm git \
                     python-virtualenv python3-dev \
                     libjpeg8-dev zlib1g-dev libwebp-dev
```

On fedora 23:

```
sudo dnf install nodejs npm git \
                 python-virtualenv python3-devel \
                 libjpeg-turbo-devel zlib-devel
```

Note: The application is developed with python 3, which these installation instructions also use. If you want to use python 2.7, make sure you install the appropriate packages (e.g. python-dev instead of python3-dev, etc.)!

Development

Assumptions

For clarity purposes regarding these instructions we are assuming the following

- You will be installing the program in `/home/wger/wger`

Requirements

Get the code

The code is available on Github:

```
$ git clone https://github.com/wger-project/wger.git
```

Create a virtual environment

It's a best practice to create a Python virtual environment:

```
$ virtualenv --python python3 venv-wger
$ source venv-wger/bin/activate
$ cd wger
```

Install Requirements

To install the Python requirements:

```
$ pip install -r requirements_devel.txt
$ python setup.py develop
```

Install application

To install the development server, init the database and create a settings file:

```
$ wger create_settings \
        --settings-path /home/wger/wger/settings.py \
        --database-path /home/wger/wger/database.sqlite
$ wger bootstrap \
        --settings-path /home/wger/wger/settings.py \
        --no-start-server
```

Start the server

To start the server:

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

That's it. You can log in with the default administrator user:

- **username**: admin
- **password**: admin

You can start the application again with the django server with `python manage.py runserver`.

Docker images

There are docker files available to quickly get a version of wger up and running. They are all located under `extras/docker` if you want to build them yourself.

Development

This image installs the application using virtualenv, uses a sqlite database and serves it with django's development server.

First build the image:

```
docker build -t wger/devel .
```

Run a container and start the application:

```
docker run -ti --name wger.devel --publish 8000:8000 wger/devel
(in docker) source ~/venv/bin/activate
(in docker) python manage.py runserver 0.0.0.0:8000
```

Now you can access the application on port 8000 of your host (probably just http://localhost:8000).

Depending on what you intend to do with it, you might want to map a local folder to the container. This is interesting if e.g. you want to keep the wger source code on your host machine and use docker only to serve it. Then you can do this:

```
docker run -ti \
    --name wger.test1 \
    --publish 8005:8000 \
    --volume /path/to/local/wger/:/home/wger/src \
     wger/devel
```

It will mount the local path *on top* of the folder in the container, basically exchanging one for the other. Please note that for this to work you need to manually checkout the code to `/path/to/local/wger/` and create a settings file as well.

Apache

This image runs the application using WSGI and apache.

First build the image:

```
docker build --tag wger/apache .
```

Run a container and start the application:

```
docker run -ti --name wger.apache --publish 8000:80 wger/apache
```

Now you can access the application on port 8000 of your host (probably just http://localhost:8000).

Production

Wger user

It is recommended to add a dedicated user for the application:

```
sudo adduser wger --disabled-password --gecos ""
```

The following steps assume you did, but it is not necessary (nor is it necessary to call it 'wger'). In that case, change the paths as needed.

Apache

Install apache and the WSGI module:

```
sudo apt-get install apache2 libapache2-mod-wsgi-py3
sudo vim /etc/apache2/sites-available/wger.conf
```

Configure apache to serve the application:

```
<Directory /home/wger/src>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>


<VirtualHost *:80>
    WSGIDaemonProcess wger python-path=/home/wger/src python-home=/home/wger/venv
    WSGIProcessGroup wger
    WSGIScriptAlias / /home/wger/src/wger/wsgi.py

    Alias /static/ /home/wger/static/
    <Directory /home/wger/static>
        Require all granted
    </Directory>

    Alias /media/ /home/wger/media/
    <Directory /home/wger//media>
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/wger-error.log
    CustomLog ${APACHE_LOG_DIR}/wger-access.log combined
</VirtualHost>
```

Apache has a problem when uploading files that have non-ASCII characters, e.g. for exercise images. To avoid this, add to /etc/apache2/envvars (if there is already an `export LANG`, replace it) or set your system's locale:

```
export LANG='en_US.UTF-8'
export LC_ALL='en_US.UTF-8'
```

Activate the settings and disable apache's default:

```
sudo a2dissite 000-default.conf
sudo a2ensite wger
sudo service apache2 reload
```

Database

postgreSQL

Install the postgres server and create a database and a user:

```
sudo apt-get install postgresql postgresql-server-dev-9.3 # or appropriate version
su - postgres
createdb wger
psql wger -c "CREATE USER wger WITH PASSWORD 'wger'";
psql wger -c "GRANT ALL PRIVILEGES ON DATABASE wger to wger";
```

You might want or need to edit your `pg_hba.conf` file to allow local socket connections or similar.

sqlite

If using sqlite, create a folder for it (must be writable by the apache user):

```
mkdir db
touch db/database.sqlite
chmod -R o+w db
```

Application

Make a virtualenv for python and activate it:

```
virtualenv --python python3 /home/wger/venv
source /home/wger/venv/bin/activate
```

Create folders to collect all static resources and save uploaded files. The `static` folder will only contain CSS and JS files, so it must be readable by the apache process while `media` will contain the uploaded files and must be writeable as well:

```
mkdir static

mkdir media
chmod o+w media
```

Get the application:

```
git clone https://github.com/wger-project/wger.git /home/wger/src
cd /home/wger/src
pip install -r requirements.txt
python setup.py develop
pip install psycopg2 # Only if using postgres
wger create_settings \
      --settings-path /home/wger/src/settings.py \
      --database-path /home/wger/db/database.sqlite
```

If you are using postgres, you need to edit the settings file and set the correct values for the database (use `django.db.backends.postgresql_psycopg2` for the engine). Also set `MEDIA_ROOT` to `/home/wger/media` and `STATIC_ROOT` to `/home/wger/static`.

Run the installation script, this will download some CSS and JS libraries and load all initial data:

```
wger bootstrap --settings-path /home/wger/src/settings.py --no-start-server
```

Collect all static resources:

```
python manage.py collectstatic
```

The bootstrap command will also create a default administrator user (you probably want to change the password as soon as you log in):

- **username**: admin
- **password**: admin

Other changes

If you want to use the application as a public instance, you will probably want to change the following templates:

- **tos.html**, for your own Terms Of Service here
- **about.html**, for your contact address or other such legal requirements

Settings

You can configure some of the application behaviour with the `WGER_SETTINGS` dictionary in your settings file. Currently the following options are supported:

ALLOW_REGISTRATION: Default `True`. Controls whether users can register on their own or if a gym administrator has to create the user accounts.

ALLOW_GUEST_USERS: Default `True`. Controls whether users can use the site as a guest user or if an administrator has to create the user accounts, as with the option above.

USE_RECAPTCHA: Default `False`. Controls whether a captcha challenge will be presented when new users register.

REMOVE_WHITESPACE: Default `False`. Removes whitespaces around HTML tags to reduce the size of the resulting HTML. If you are not serving the site using TLS you probably want to use the GZip middleware instead. Read the django documentation on the security implications (BREACH attack).

EMAIL_FROM: Default *wger Workout Manager <wger@example.com>* The sender address used for sent emails by the system such as weight reminders

Note: If you want to override a default setting, don't overwrite all the dictionary but only the keys you need, e.g. `WGER_SETTINGS['foo'] = 'bar'`. This avoids problems when new keys are added in the global settings.

Commands

Please note that the administration commands are intended e.g. to bootstrap/install an application to a new system, while the management ones are made to administer a running application (to e.g. delete guest users, send emails, etc.).

Administration Commands

The application provides several administration and bootstraping commands that can be passed to the `wger` command:

```
wger <command>
```

You can get a list of all available commands by calling `wger` without any arguments:

```
Available tasks:

bootstrap               Performs all steps necessary to bootstrap the application
config_location         Returns the default location for the settings file and the␣
↪data folder
create_or_reset_admin   Creates an admin user or resets the password for an existing␣
↪one
create_settings         Creates a local settings file
load_fixtures           Loads all fixtures
migrate_db              Run all database migrations
start                   Start the application using django's built in webserver
```

You can also get help on a specific command with `wger --help <command>`.

Note: Most commands support a `--settings-path` command line option that sets the settings file to use for the operation. If you use it, it is recommended to use absolute paths, for example:

```
wger bootstrap --settings-path /path/to/development/wger/settings-test.py
```

Bootstrap

Command: **bootstrap**

This command bootstraps the application: it creates a settings file, initialises a sqlite database, loads all necessary fixtures for the application to work and creates a default administrator user. While it can also work with e.g. a postgreSQL database, you will need to create it yourself:

```
wger bootstrap
```

The most usual use-case is creating the settings file and the sqlite database to their default locations, but you can set your own paths if you want e.g. start developing on a branch that is going to change the database schema.

Usage:

```
Usage: inv[oke] [--core-opts] bootstrap [--options] [other tasks here ...]

Docstring:
  Performs all steps necessary to bootstrap the application

Options:
  -a STRING, --address=STRING         Address to use. Default: localhost
  -b, --browser                       Whether to open the application in a browser␣
↪window. Default: false
  -d STRING, --database-path=STRING   Path to sqlite database (absolute path␣
↪recommended). Leave empty for default
  -p, --port                          Port to use. Default: 8000
  -s STRING, --settings-path=STRING   Path to settings file (absolute path␣
↪recommended). Leave empty for default
```

Start wger

Command: **start**

Starts an already installed application:

```
wger start
```

Please note that this is simply a comfort function and does not use any *magic*, it simply calls django's development server and (optionally) opens a browser window. If you are developing, using the usual `python manage.py runserver` is probably better.

Usage:

```
Usage: inv[oke] [--core-opts] start [--options] [other tasks here ...]

Docstring:
  Start the application using django's built in webserver

Options:
  -a STRING, --address=STRING         Address to bind to. Default: localhost
  -b, --browser                       Whether to open the application in a browser␣
↪window. Default: false
  -e STRING, --extra-args=STRING      Additional arguments to pass to the builtin␣
↪server. Pass as string: "--arg1 --arg2=value". Default: none
  -p, --port                          Port to use. Default: 8000
  -s STRING, --settings-path=STRING   Path to settings file. Leave empty for default
  -t, --[no-]start-server             Whether to start the development server.␣
↪Default: true
```

Default locations

Command: **config_location**

Information command that simply outputs the default locations for the settings file as well as the data folder used for the sqlite database and the uploaded files.

Create settings

Command: **create_settings**

Creates a new settings file based. If you call it without further arguments it will create the settings in the default locations:

```
wger create settings
```

If you pass custom paths, it's recommended to use absolute paths:

```
wger create_settings --settings-path /path/to/development/wger/settings-test.py --
↪database-path /path/to/development/wger/database-test.sqlite
```

Usage:

```
Usage: inv[oke] [--core-opts] create_settings [--options] [other tasks here ...]

Docstring:
  Creates a local settings file

Options:
  -a STRING, --database-type=STRING   Database type to use. Supported: sqlite3,␣
↪postgresql. Default: sqlite3
  -d STRING, --database-path=STRING   Path to sqlite database (absolute path␣
↪recommended). Leave empty for default
  -k, --key-length                    Lenght of the generated secret key. Default: 50
  -s STRING, --settings-path=STRING   Path to settings file (absolute path␣
↪recommended). Leave empty for default
  -u STRING, --url=STRING
```

Create or reset admin

Command: **create_or_reset_admin**

Makes sure that the default administrator user exists. If you change the password it is reset.

Usage:

```
Usage: inv[oke] [--core-opts] create_or_reset_admin [--options] [other tasks here ...]

Docstring:
  Creates an admin user or resets the password for an existing one

Options:
  -s STRING, --settings-path=STRING   Path to settings file (absolute path␣
↪recommended). Leave empty for default
```

Migrate database

Command: **migrate_db**

Migrates the database schema. This command is called internally when installing the application. The only need to call this explicitly is after installing a new version of the application.

Calling this command is a safe operation, if your database is current, nothing will happen.

Usage:

```
Usage: inv[oke] [--core-opts] migrate_db [--options] [other tasks here ...]

Docstring:
  Run all database migrations

Options:
  -s STRING, --settings-path=STRING   Path to settings file (absolute path␣
↪recommended). Leave empty for default
```

Load all fixtures

Command: **load_fixtures**

Loads all fixture file with the default data. This data includes all data necessary for the application to work such as:

- exercises, muscles, equipment
- ingredients, units
- languages
- permission groups
- etc.

This command is called internally when installing the application but you can use it to reset the data to the original state. Note: new entries or user entries such as workouts are *not* reset with this, only the application data.

Usage:

```
Usage: inv[oke] [--core-opts] load_fixtures [--options] [other tasks here ...]

Docstring:
  Loads all fixtures

Options:
  -s STRING, --settings-path=STRING   Path to settings file (absolute path␣
↪recommended). Leave empty for default
```

Management commands

wger also implements a series of django commands that perform different management functions that are sometimes needed. Call them with `python manage.py <command_name>`:

download-exercise-images synchronizes the exercise images from wger.de to the local installation. Read its help text as it could save the wrong image to the wrong exercise should different IDs match.

redo-capitalize-names re-calculates the capitalized exercise names. This command can be called if the current "smart" capitalization algorithm is changed. This is a safe operation, since the original names (as entered by the user) are still available.

submitted-exercises simply prints a list of user submitted exercises

extract-i18n extract strings from the database that have to be inserted manually in the PO file when translating. These include e.g. exercise categories.

clear-cache clears different application caches. Might be needed after some updates or just useful while testing. Please note that you must select what caches to clear.

update-user-cache update the user cache-table. This command is only needed when the python code used to calculate any of the cached entries is changed and the ones in the database need to be updated to reflect the new logic.

Cron

The following commands are built to be called regularly, via a cronjob or similar

delete-temp-users deletes all guest users older than 1 week. At the moment this value can't be configured

email-reminders sends out email reminders for user that need to create a new workout.

email-weight-reminders sends out email reminders for user that need to enter a new (body) weight entry.

inactive-members Sends email for gym members that have not been to the gym for a specified amount of weeks.

Tips and tricks

Updating the code

When pulling updates from upstream there are a couple of things to consider. These steps apply to all installation methods above.

Upgrading the database

There are regularly changes and upgrades to the database schema (these may also come from new versions of django or the installed dependencies). If you start the development server and see a message that there are unapplied migrations, just do `python manage.py migrate --all`.

Downloading dependencies with Bower

Bower is used to download different JS and CSS libraries. If you update master it is recommended that you first delete the existing libraries (`rm wger/core/static/bower_components`) and then download the new versions with:

```
$ python manage.py bower install
```

Some info about bower, during the bootstrap process bower is installed locally to src/wger. If this didn't work and you get an error saying that bower is not installed, you can manually install it by going to the project's root directory and performing the step manually:

```
$ cd src/wger
$ npm install bower
```

Alternatively, you can manually set the path to the bower binary by editing `BOWER_PATH` (see `wger/settings_global.py`).

Clearing the cache

Sometimes there are changes to the internal changes of the cached structures. It is recommended that you just clear all the existing caches `python manage.py clear-cache --clear-all`

Miscellaneous settings

The following settings can be very useful during development (add to your settings.py):

Setting the email backend Use the console backend, all sent emails will be printed to it:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

Dummy data generator

To properly test the different parts of the application for usability or performance, it is often very useful to have some data to work with. For this reason, there is a dummy data generator script in extras/dummy_generator/generator.py. It allows you to generate entries for users, gyms, workouts and logs. For detailed usage options do:

```
python generator.py --help
```

Or for options for, e.g. user generation:

```
python generator.py users --help
```

To get you started, you might want to invoke the script in the following way. This will create 10 gyms and 300 users, randomly assigning them to a different gym. Each user will have 20 workouts and each exercise in each workout 30 log entries:

```
python generator.py gyms 10
python generator.py users 300
python generator.py workouts 20
python generator.py logs 30
python generator.py sessions random
python generator.py weight 100
python generator.py nutrition 20
```

Note: All generated users have their username as password.

Note: While it is possible to generate hundreds of users, gyms are more restricted and you will probably get duplicate names if you generate more than a dozen.

Selectively running tests

If you do a `python manage.py test` you will run the complete testsuite, and this can take a while. You can control which tests will be executed like this.

Test only the tests in the 'core' app:

```
python manage.py test wger.core
```

Test only the tests in the 'test_user.py' file in the core app:

```
python manage.py test wger.core.tests.test_user
```

Test only the tests in 'StatusUserTestCase' in the file 'test_user.py' file in the core app:

```
python manage.py test wger.core.tests.test_user.StatusUserTestCase
```

Using runserver_plus

During development you can use `runserver_plus` instead of the default django server as you can use an interactive debugger directly from the browser if an exception occurs. It also accepts the same command line options. For this just install the following packages:

```
pip install django_extensions werkzeug
python manage.py runserver_plus [options]
```

Contributing

- **Send pull requests**: for new code you want to share, please send pull requests in github. Sending patches by email or attaching them to an issue means a lot more of work. It's recommended that you work on a feature branch when working on something, specially when it's something bigger. While many people insist on rebasing before sending a pull request, it's not necessary.
- **Run the tests**: wger is proud to have a test coverage of over 90%. When you implement something new, don't forget to run the testsuite and write approriate tests for the new code. If you use github, configure the awesome Travis CI, there is already a .travis file in the sources.
- **Code according to the coding style**: *Coding Style Guide*

Coding Style Guide

Python

- Code according to PEP8

  Check that the code is structured as per pep8 but with a maximum line length of 100.

- Code for Python 3

  While the application should remain compatible with python2, use django's suggestion to mantain sanity: code for py3 and treat py2 as a backwards compatibility requirement. If you need, you can use six.

Javascript

- Follow AirBnB ES5 style guide, with the following changes:
  - Disallow named function expressions, except in recursive functions, where a name is needed.
  - Console logging is allowed
- Functions called from Django templates need to start with `wger`

Automatic coding style checks

The coding style is automatically check by Travis-CI. To manually check your files you can run the following commands:

- Python: `pep8 wger`
- Javascript: `./node_modules/.bin/gulp lint-js` (or just `gulp lint-js` if you installed the node libraries globally on your system)

Internationalization (i18n)

Updating the translation files

wger uses django's translation infrastructure, but there are a couple of things that need to be considered. First, update your po files with the usual django command (run this in the wger subfolder, not the root one):

```
django-admin makemessages --locale en --extension py,html,tpl
```

Extract some translatable strings from the database such as exercise categories and muscle names:

```
python ../manage.py extract-i18n
```

and add them to the end of `locale/en/LC_MESSAGES/django.po`. Once you have translated the file, compile it with:

```
django-admin compilemessages --all
```

Adding new languages

Besides adding the new translations to the locale folder, they have to be activated in the django settings file and in the application itself.

Note: At the moment composed language codes such as pt-BR (Brazilian Portuguese) are **not** supported, the issue for this problem is #130

- **django:** add an entry to `LANGUAGES` in `wger/settings_global.py`
- **wger:** add the new language in the language admin page and set the visibility of exercises and ingredients. For the short name, use the language code such as 'fr', for the long name the native name, in this example 'français'.
- **compile**: to use the new language files, the translation files have to be compiled. Do this by changing to the wger folder (so you see a `locale` folder there) and invoking `django-admin compilemessages`. You will also need to restart the webserver.
- **flag icon:** add an appropriate flag icon in SVG format in `images/icons/flag-CODE.svg` in the static folder of the core application.
- **fixtures:** after having added the language in the admin module, the data has to be exported so the current language configuration can be reproduced. This is done with the `filter-fixtures.py` script:
  - while in `extras/scripts`, export the whole database to a json file with:

```
python ../../manage.py dumpdata --indent 4 --natural-foreign > data.json
```

- filter the database dump, this will generate a json file for each "important" module:

```
python filter-fixtures.py
```

- copy the generated files `languages.json` and `language_config.json` to the fixtures folder in core and config (you'll probably want to delete the remaining json files):

```
cp languages.json ../../wger/core/fixtures/
cp language_config.json ../../wger/config/fixtures/
rm *.json
```

Getting new languages

If you have a local installation and new languages arrive from upstream, you need to load the necessary data to the language tables in the database (note that you'll need to reload/restart the webserver so the new po files are picked up):

```
python manage.py loaddata languages
python manage.py loaddata language_config
```

Please note that this will overwrite any changes you might have done from the language administration module.

CHAPTER 2

Administration guide

Gym admnistration

wger has support for gym and member management, allowing e.g. trainers to follow the progress of their athletes and for the gym managers to keep track of the member's contracts.

It is possible to manage a single or different ones with one instance of wger. If the installation is used for a single gym only, you can set the default gym in the global configuration options in the gym list. This will update all existing users as well as newly registered ones so they belong to that gym.

There are 3 groups used for the different administrative roles:

- **general manager:** Can manage (add, edit, delete) the different gyms for the installation as well as add gym managers, trainers and member, but is not allowed to see the members workout data.
- **gym manager:** Can manage users for a single gym (editing, deactivating, adding contracts, etc.).
- **trainer:** Can manage the workouts and other data for the members of a single gym.

These roles are not mutually exclusive, if your workflow demands it, you can combine all three roles into one account.

Except for general managers, administrative users belong to a single gym (the one they were created in) and can access only those members. This setting cannot be changed later. The user's gym appears in the top right menu.

Member management

You can new members to a gym by clicking the *Add member* button at the top of the member overview. After filling in the form, a password will be generated for the user. You should save this and give it to the user, as it is not possible to retrieve it later. Alternatively you can just instruct the new members to use the reset password links when logging in for the first time.

An export of all gym members is available as well from the "actions" button on the gym detail table. This provides a CSV file that can be imported into a spreadsheet program for further processing.

Trainers can click on a user and access an overview of the user's workouts, body weights, nutrition plans, etc. When clicking on the "log in as this user", the trainer can assume the identity of the user to create new workouts for example.

Individual members can be deactivated by clicking on the "actions" button on the top of the member's detail table. Deactivated users can't log in, but are not deleted from the system and can be reactivated at any time in the future. If you wish to completely delete a user from the system, use the "delete" option but keep in mind that this action cannot be undone.

Notes and documents

Trainers can add notes and upload documents related to individual members. A note is a free text, while a document can be any file. This information can be used to save information on specific injuries or other important notes related to the member. Note that these entries are not accessible by the members themselves, but only by the trainers.

Contracts

It is also possible to manage the members' contracts with the application. A contract is composed of a base form and optional *type* and *options*. The type is a single attribute, such as "Student contract" or "Special offer 2016". The options are basically the same but more than one can be selected at once and can be used for items that can e.g. be booked in addition to the default contract such as "Sauna" or "Protein drink flatrate".

The types and the options are added gym-wide in the member overview by the managers. Once these are saved, they can be used when adding or editing a contract to a specific user.

Mass emails

It is possible to send an email to all members of a gym. At the moment it is not possible to filter this list or send it only to members that fulfill a specific criterion.

To send this, go to the gym's overview and click "Add" on the Email actions button. After filling in the subject and the body you need to accept the preview, in order to make sure that the text you wrote is indeed the one you intend to send. After submitting the form, the emails will be sent in batches (how exactly depends on how you configured the different cron jobs needed to run the application).

Configuration

Inactive members

The idea behind the inactive members is that trainers want to know which users have not visited the gym for longer than X weeks and e.g. contact them. 'Inactive' means here that they don't have any logs in the time period.

This can be configured in the following ways:

number of weeks The value in weeks after which users are considered inactive (default is 8). This applies to the whole gym and can be deactivated by entering a 0.

trainer configuration Each trainer can opt-out of receiving such emails

user configuration Individual users can be opt-out of being included in the reminder emails if they don't want to use the log or any other reason.

Gym name in header

A checkbox to control whether the gym's name will appear in the header instead of the application's name for all logged in users of this gym. This applies to members, trainers and managers.

CHAPTER 3

Changelog

Changelog

1.9 - IN DEVELOPMENT

2017-xx-xx

Upgrade steps from 1.8:

- Django update to 1.10: `pip install -r requirements.txt`
- Database upgrade: `python manage.py migrate`
- Update static files (only production): `python manage.py collectstatic`

New features:

- ...

Improvements:

- Improve look of weight graph (thanks @alokhan) #381
- Added password validation rules for more security
- Exercise image downloader checks only accepted exercises (thanks @gmmoraes) #363
- Use a native data type for the exercises' UUID (thanks @gmmoraes) #364
- Increase speed of testsuite by performing the tests in parallel (thanks @Mbarak-Mbigo) wger_vulcan/#6
- Update screen when adding an exercise to the workout while using set slider (thanks @gmmoraes) #374

Other improvements and bugfixes: #336, #359,'#386'_

1.8

2017-04-05

Warning: There have been some changes to the installation procedure. Calling 'invoke' on its own has been deprecated, you should use the 'wger' command (which accepts the same options). Also some of these commands have been renamed:

- `start_wger` to `wger`
- `bootstrap_wger` to `bootstrap`

Upgrade steps from 1.7:

- Django update to 1.9: `pip install -r requirements.txt`
- Database upgrade: `python manage.py migrate`
- Reset cache: `python manage.py clear-cache --clear-all`
- Due to changes in the JS package management, you have to delete wger/core/static/bower_components and do a `python manage.py bower install`
- Update static files (only production): `python manage.py collectstatic`
- Load new the languages fixtures as well as their configuration `python manage.py loaddata languages` and `python manage.py loaddata language_config`
- New config option in settings.py: `WGER_SETTINGS['TWITTER']`. Set this if your instance has its own twitter account.

New languages:

- Norwegian (many thanks to Kjetil Elde @w00p #304)
- French (many thanks to all translators)

New features:

- Big ingredient list in Dutch, many thanks to alphafitness.club!
- Add repetition (minutes, kilometer, etc.) and weight options (kg, lb, plates, until failure) to sets #216 and #217
- Allow administrators to deactivate the guest user account #330
- Add option to show the gym name in the header instead of the application name, part of #214
- Exercise names are now capitalized, making them more consistent #232
- Much improved landing page (thanks @DeveloperMal) #307
- Add extended PDF options to schedules as well (thanks @alelevinas) #272
- Show trained secondary muscles in workout view (thanks @alokhan) #282
- Use the metricsgraphics library to more easily draw charts #188
- Removed persona (browserID) as a login option, the service is being discontinued #331

Improvements:

- Check and enforce style guide for JS files #317 (@petervanderdoes)
- BMI calculator now works with pounds as well (thanks @petervanderdoes) #318
- Give feedback when autocompleter didn't find any results #293
- Make exercise names links to their detail page in training log pages #350
- Better GUI consistency in modal dialogs (thanks @jstoebel) #274
- Cache is cleared when editing muscles (thanks @RyanSept @pythonGeek) #260

- Fields in workout log form are no longer required, making it possible to only log weight for certain exercises #334
- New, more verbose, API endpoint for exercises, (thanks @andela-bmwenda)
- The dashboard page was improved and made more user friendly #201 (partly)
- Replace jquery UI's autocompleter and sortable this reduces size of JS and CSS #78 and #79
- Update to D3js v4 #314, #302
- Remove hard-coded CC licence from documentation and website #247

Other improvements and bugfixes: #25, #243, #279, #275, #270, #258, #257, #263, #269, #296, #297, #303, #311, #312, #313, #322, #324, #325

1.7

2016-02-28

New translations:

- Czech (many thanks to Tomáš Z.!)
- Swedish (many thanks to ywecur!)

New features:

- Workout PDF can now print the exercises' images and comments #261
- Allow login with username or email (thanks @warchildmd) #164'_
- Correctly use user weight when calculating nutrional plans' calories (thanks @r-hughes) #210
- Fix problem with datepicker #192
- Order of exercises in supersets is not reverted anymore #229
- Improvements to the gym management:
  - Allow to add contracts to members
  - Visual consistency for lists and actions
  - Vastly reduce the number of database queries in gym member list #144
  - Global list of users for installation #212
  - Allow administrators to restrict user registration #220
  - Refactored and improved code, among others #208
  - Allow gym managers to reset a member's password #186
- Better rendering of some form elements #244
- Improved GUI consistency #149
- Docker images for easier installation #181
- Use hostname for submitted exercises (thanks @jamessimas) #159
- Download js libraries with bowerjs (thanks @tranbenny) #126
- Improved and more flexible management commands #184
- Fixed error when importin weight entries from CSV (thanks @r-hughes) #204
- Fixed problems when building and installing the application on Windows (thanks @romansp) #197

- Fixed potential Denial Of Service attack (thanks @r-hughes) #238
- Dummy data generator can not create nutrition plans (thanks @cthare) #241

Other improvements and bugfixes: #279, #275, #270, #258, #257

1.6.1

2015-07-25

Bugfix release

1.6

2015-07-25

New translations:

- Greek (many thanks to Mark Nicolaou!)

New features:

- Save planed weight along with the repetitions #119
- Improvements to the workout calendar #98
- Allow external access to workouts and other pages to allow for sharing #102, #124
- Email reminder to regularly enter (body) weight entries #115
- Allow users to submit corrections to exercises
- Add day detail view in workout calendar #103
- Fix bug where the exercises added to a superset did not remain sorted #89
- Reduce size of generated html code #125
- Allow users to copy shared workouts from others #127
- Added breadbrumbs, to make navigation easier #101
- Add option to delete workout sessions and their logs #156
- Improve installation, development and maintenance documentation #114

Other improvements and bugfixes: #99, #100, #106, #108, #110, #117, #118, #128, #131, #135, #145, #155

1.5

2014-12-16

New Translations:

- Dutch (many thanks to David Machiels!)
- Portuguese (many thanks to Jefferson Campos!) #97

New features:

- Add support for gym management #85
  - Gym managers can create and manage gyms

    - Trainers can see the gym's users and their routines
- Reduce amount of CSS and JS libraries by using bootstrap as much as possible #73
- Improvements to the REST API #75
    - Add read-write access
    - Add live browsing of the API with django rest framework
    - Improve documentation
    - /api/v1 is marked deprecated
- Show exercise pictures in workout as well
- Detailed view of exercises and workouts in schedule #86
- Support for both metric (kg) and imperial (lb) weight units #105
- Allow the user to delete his account and data #84
- Add contact field to feedback form
- Cleanup translation strings #94
- Python 3 compatibility! #68

Other improvements and bugfixes: #51, #76, #80, #81, #82, #91, #92, #95, #96

1.4

2014-03-08

New features and bugfixes:

- Calendar view to more easily check workout logs
- Add "gym mode" with timer to log the workout while at the gym
- Add automatic email reminders for new workouts
- New iCal export to add workouts and schedules e.g. to google calendar
- New exercise overview, grouped by equipment
- Add possibility to write comments and rate the workout
- Simplify form for new exercises
- Alternative PDF export of workout without table for entering logs
- Unified way of specifying license of submitted content (exercises, etc.)

1.3

2013-11-27

New translations:

- Bulgarian (many thanks to Lyuboslav Petrov!)
- Russian (many thanks to Inna!)
- Spanish

New features and bugfixes:

- Mobile version of website
- Add images to the exercises
- Exercises now can list needed equipment (barbell, etc.)
- BMI calculator
- Daily calories calculator
- New management utility for languages
- Improved performance
- RESTful API

1.2

2013-05-19

New features and bugfixes:

- Added scheduling option for workouts.
- Open all parts of website to all users, this is done by a custom middleware
- Regular users can submit exercises and ingredients to be included in the general list
- Add more 'human' units to ingredients like '1 cup' or '1 slice'
- Add nutritional values calculator on the ingredient detail page
- Several bugfixes
- Usability improvements

1.1.1

2013-03-06

New features and bugfixes:

- Pin version of app django_browserid due to API changes in 0.8
- Fix issue with tabs on exercise overview due to API changes in JQuery

1.1

2013-02-23

New features and bugfixes:

- Better navigation bar
- Added descriptions for the exercises (German)
- New workout logbook, to keep track of your improvements
- Import your weight logs from a spreadsheet (CSV-Import)
- Better filtering for weight chart
- Muscle overview with corresponding exercises

- Add guest accounts by generating a temporary user
- Description pages about the software
- Easier installation process

1.0.3

2012-11-19

New features and bugfixes:

- Add option to copy (duplicate) workouts and nutritional plans
- Login without an account with [[https://login.persona.org/|mozilla's Persona]] (BrowserID)
- Better AJAX handling of the modal dialogs, less page reloads and redirects
- Expand the list of ingredients in German
- Add a pagination to ingredient list
- Improvements to user page:
  - Add a "reset password" link to the login page
  - Email is now user editable
- More natural lines in weight chart with cubic interpolation

1.0.2

2012-11-02

Bugfix release

1.0.1

2012-11-02

New features and bugfixes:

- Fix issue with password change
- Small improvements to UI
- Categories editable/deletable from exercise overview page
- Exercise AJAX search groups by category
- More tests!
- Use generic views for editing, creating and deleting objects

1.0

2012-10-16

Initial release.

New features and bugfixes:

- Workout manager
- PDF output for logging progress
- Initial data with the most popular exercises
- Simple weight chart
- Nutrition plan manager
- Simple PDF output
- Initial data with nutritional values from the USDA

CHAPTER 4

Contact

Feel free to contact us if you found this useful or if there was something that didn't behave as you expected (in this case you can also open a ticket on the issue tracker).

- **twitter:** https://twitter.com/wger_de
- **mailing list:** https://groups.google.com/group/wger / wger@googlegroups.com, no registration needed
- **IRC:** channel #wger on freenode.net, webchat: http://webchat.freenode.net/?channels=wger
- **issue tracker:** https://github.com/wger-project/wger/issues

CHAPTER 5

Sources

All the code and the content is freely available and is hosted on github: https://github.com/wger-project/wger

CHAPTER 6

Licence

The application is licenced under the Affero GNU General Public License 3 or later (AGPL 3+).

The initial exercise and ingredient data is licensed additionally under a Creative Commons Attribution Share-Alike 3.0 (CC-BY-SA 3.0)

The documentation is released under a CC-BY-SA either version 4 of the License, or (at your option) any later version.

Some images where taken from Wikipedia, see the SOURCES file in their respective folders for more details.

CHAPTER 7

Authors

Developers

- Roland Geider – https://github.com/rolandgeider
- Helen Sherwood-Taylor - https://github.com/helenst
- Tushar Gupta - https://github.com/Tushar-Gupta
- Matheus Zorzete Marchiore - https://github.com/marchiore
- Yu Yu Aung - https://github.com/max111
- Laszlo Ratsko - https://github.com/rlaszlo
- James Simas - https://github.com/jamessimas
- Benny Tran - https://github.com/tranbenny
- Roman Pavlov - https://github.com/romansp
- Mihail Burduja - https://github.com/warchildmd
- Colin Hare - https://github.com/cthare
- Ryan Hughes - https://github.com/purplebird
- Alexandre Murphy-Gonthier - https://github.com/murphyalexandre
- Verdi R-D: https://github.com/azend
- Alois: https://github.com/alokhan
- Alelevinas: https://github.com/alelevinas
- Jacob Stoebel: https://github.com/jstoebel
- Peter van der Does: https://github.com/petervanderdoes
- Malcolm Jones: https://github.com/DevloperMal
- Boniface Mwenda: https://github.com/andela-bmwenda

Translators

- Bulgarian: Lyuboslav Petrov
- Czech: Tomáš Z.
- Dutch: David Machiels
- Greek: Mark Nicolaou
- Norwegian: Kjetil Elde
- Portuguese: Jefferson Campos – http://jeffersoncampos.eti.br
- Russian: Inna
- Spanish: acv2facundo
- Swedish: ywecur

Exercises

And of course many thanks as well to everyone that submited exercises!

CHAPTER 8

Indices and tables

- genindex
- search
- modindex