wger Workout Manager Documentation

Release 2.3 alpha

Roland Geider

Jan 06, 2024

CONTENTS

1 Installation and administration **3**
- 1.1 Installation . . . 3

2 Development **15**
- 2.1 Tips and tricks . . . 15
- 2.2 Coding Style Guide . . . 17
- 2.3 Internationalization (i18n) . . . 17

3 Administration guide **19**
- 3.1 Commands . . . 19
- 3.2 Settings . . . 24
- 3.3 Gym administration . . . 25

4 Changelog **29**
- 4.1 Changelog . . . 29

5 Contact **43**

6 Sources **45**

7 Licence **47**

wger (v) Workout Manager is a free, open-source web application that helps you manage your personal workouts, weight, and diet plans and can also be used as a simple gym management utility. It offers a REST API as well, for easy integration with other projects and tools.

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

ONE

INSTALLATION AND ADMINISTRATION

1.1 Installation

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.

You can safely install from master, it is almost always in a usable and stable state.

The application is compatible and regularly tested with

- PostgreSQL, sqlite
- python 3.8, 3.9, 3.10 and pypy3

After installation, 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:

```
sudo apt-get install nodejs npm git python3-dev python3-venv libcairo2-dev pkg-config
sudo npm install -g yarn sass
```

If you plan to use video files, you need to install additional packages, consult the videos section for more information.

1.1.1 Development

Virtual environment

Create a new virtualenv:

```
$ python3 -m venv venv-wger
$ source venv-wger/bin/activate
```

Get the code

Download the source code:

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

Install Requirements

Install python requirements:

```
$ pip install -r requirements_dev.txt
$ pip install -e .
```

Install application

This will download the required JS and CSS libraries and create an SQLite database and populate it with data on the first run:

```
$ wger create-settings
$ wger bootstrap
$ wger load-online-fixtures
```

You can of course also use other databases such as PostgreSQL or MariaDB. Create a database and user and edit the DATABASES settings before calling bootstrap. Take a look at the *PostgreSQL* on apache on how that could look like.

Start the server

After the first run you can just use Django's development server:

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

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

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

You can reset the admin's password with `wger create-or-reset-admin`.

1.1.2 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>
    WSGIApplicationGroup %{GLOBAL}
    WSGIDaemonProcess wger python-path=/home/wger/src python-home=/home/wger/venv
    WSGIProcessGroup wger
    WSGIScriptAlias / /home/wger/src/wger/wsgi.py
    WSGIPassAuthorization On

    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 (choose the appropriate and currently supported version for your distro) and create a database and a user:

```
sudo apt-get install postgresql postgresql-server-dev-12 python3-psycopg2
sudo 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
chown :www-data -R /home/wger/db
chmod g+w /home/wger/db /home/wger/db/database.sqlite
```

Application

As the wger user, make a virtualenv for python and activate it:

```
python3 -m venv /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
pip install -e .

# If using sqlite without the --database-path
wger create-settings --database-path /home/wger/db/database.sqlite
```

Edit the settings file

- Add the correct values for the database (use `django.db.backends.postgresql` for the engine) if you are using postgres
- Set `MEDIA_ROOT` to `/home/wger/media` and `STATIC_ROOT` to `/home/wger/static`.
- Add the domains that your site will be accessed to *ALLOWED_HOSTS=['example.com', 'www.example.com']* (you might want to do this as the last step when you know everything else is working correctly)

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

```
wger bootstrap
```

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**: adminadmin

Email

The application is configured to use Django's console email backend by default, which causes messages intended to be sent via email to be written to `stdout`.

In order to use a real email server, another backend listed in Django's documentation can be configured instead. Parameters for the backend are set as variables in `settings.py`. For example, the following allows an SMTP server at `smtp.example.com` to be used:

```
Email_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
ENABLE_EMAIL = True
EMAIL_HOST = 'smtp.example.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'wger@example.com'
EMAIL_HOST_PASSWORD = 'example_password'
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
DEFAULT_FROM_EMAIL = 'wger Workout Manager <wger@example.com>'
```

Django provides a `sendtestemail` command via `manage.py` to test email settings:

```
python manage.py sendtestemail user@example.com
```

Site Settings

Some wger features make use of Django's site name and domain settings in the `contrib.sites` framework. These should be set through the Python shell:

```
python manage.py shell
>>> from django.contrib.sites.models import Site
>>> site = Site.objects.get(pk=1)
>>> site.domain = 'wger.example.com'
```

(continues on next page)

(continued from previous page)

```
>>> site.name = 'example.com wger Workout Manager'
>>> site.save()
```

where `wger.example.com` is the domain of the wger instance. This assumes that wger is using the default site ID of 1. If a different site ID is being used, it must be specified in `settings.py`:

```
SITE_ID = 2
```

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

1.1.3 Celery queue

wger can use a celery queue for some background tasks. At the moment this is used for things like fetching the ingredient images or periodically synchronizing the exercise database.

The celery queue is optional and is not required.

You will need to configure a cache backend, redis is a good and easy solution and you might already have it running for the regular application cache:

```
CELERY_BROKER_URL = "redis://localhost:6379/2"
CELERY_RESULT_BACKEND = "redis://localhost:6379/2"
```

Finally, set the option in settings.py to use it:

```
WGER_SETTINGS['USE_CELERY'] = True
```

For alternatives, consult celery's documentation: https://docs.celeryq.dev/en/stable/

Note: The docker compose file has all services already configured, you don't need to do anything

Celery worker

For development, to start the celery worker just run in your virtual env:

```
celery -A wger worker -l INFO
```

In a production setting you will need to properly daemonize the service. One option is a systemd service file. If you are using the structure and names used in the production chapter of this documentation, you can proceed as follows.

Create a folder in where we will put the configuration:

```
mkdir /home/wger/celery/
```

Create a configuration file called `conf` and paste this:

```
# Name of nodes to start
CELERYD_NODES="w1"

# Absolute path to the 'celery' command within the venv
CELERY_BIN="/home/wger/venv/bin/celery"

# App instance to use
CELERY_APP="wger"

# How to call manage.py
CELERYD_MULTI="multi"

# Extra command-line arguments to the worker
CELERYD_OPTS="--time-limit=300 --concurrency=8"

# - %n will be replaced with the first part of the nodename.
# - %I will be replaced with the current child process index
#   and is important when using the prefork pool to avoid race conditions.
CELERYD_PID_FILE="/var/run/celery/%n.pid"
CELERYD_LOG_FILE="/var/log/celery/%n%I.log"
CELERYD_LOG_LEVEL="INFO"

CELERYBEAT_PID_FILE="/var/run/celery/beat.pid"
CELERYBEAT_LOG_FILE="/var/log/celery/beat.log"
```

Since we're running this as the wger user, we need to make sure the process can write the logs and the pid files. For this edit a new file `/etc/tmpfiles.d/celery.conf`, past the following content and reboot the system:

```
d /run/celery 0755 wger wger -
d /var/log/celery 0755 wger wger -
```

Add a new file */etc/systemd/system/celery.service* with the following contents (if you don't have a redis service or is called differently, adjust as needed):

```
[Unit]
Description=Celery Service
After=network.target
After=redis.service
Requires=redis.service

[Service]
Type=forking
User=wger
Group=wger
EnvironmentFile=/home/wger/celery/conf
WorkingDirectory=/home/wger/src
ExecStart=/bin/sh -c '${CELERY_BIN} -A $CELERY_APP multi start $CELERYD_NODES \
    --pidfile=${CELERYD_PID_FILE} \
    --logfile=${CELERYD_LOG_FILE} \
    --loglevel="${CELERYD_LOG_LEVEL}" $CELERYD_OPTS'
ExecStop=/bin/sh -c '${CELERY_BIN} multi stopwait $CELERYD_NODES \
    --pidfile=${CELERYD_PID_FILE} \
    --logfile=${CELERYD_LOG_FILE} \
```

(continues on next page)

(continued from previous page)

```
    --loglevel="${CELERYD_LOG_LEVEL}"'
ExecReload=/bin/sh -c '${CELERY_BIN} -A $CELERY_APP multi restart $CELERYD_NODES \
    --pidfile=${CELERYD_PID_FILE} \
    --logfile=${CELERYD_LOG_FILE} \
    --loglevel="${CELERYD_LOG_LEVEL}" $CELERYD_OPTS'
Restart=always

[Install]
WantedBy=multi-user.target
```

Read the file with `systemctl daemon-reload` and start it with `systemctl start celery`. If there are no errors and `systemctl status celery` shows that the service is active, everything went well. With ``systemctl enable celery.service``the service will be automatically restarted after a reboot.

For more up to date information on how this could look like: https://docs.celeryq.dev/en/stable/userguide/daemonizing.html

Celery beat

Celery beat is used to perform periodic tasks. This is used at the moment to regularly sync the exercises from the configured wger instance. A random time and day of the week is selected in which the individual task will be run. Each task can be toggled on or off with a setting in the `WGER_SETTING` dictionary:

- **SYNC_EXERCISES_CELERY** to synchronize the exercises themselves
- **SYNC_EXERCISE_IMAGES_CELERY** to synchronize exercise images
- **SYNC_EXERCISE_VIDEOS_CELERY** to synchronize exercise videos

To start it just run in your virtual env:

```
celery -A wger beat -l INFO
```

To daemonize this you just need to add a new service, e.g. `/etc/systemd/system/celery-beat.service`:

```
[Unit]
Description=Celery Beat Service
After=network.target
After=celery.service
Requires=celery.service

[Service]
Type=forking
User=wger
Group=wger
EnvironmentFile=/home/wger/celery-conf/celery
WorkingDirectory=/home/wger/src
ExecStart=/bin/sh -c '${CELERY_BIN} -A ${CELERY_APP} beat \
    --pidfile=${CELERYBEAT_PID_FILE} \
    --logfile=${CELERYBEAT_LOG_FILE} \
    --loglevel=${CELERYD_LOG_LEVEL}'
Restart=always

[Install]
WantedBy=multi-user.target
```

Then as above, reload the server and start the service:

```
systemctl daemon-reload
systemctl start celery-beat
```

Celery flower

Celery flower is a web app that allows you to take a look at the performed tasks

To start it just run in your virtual env:

```
celery -A wger --broker="${CELERY_BROKER}" flower
```

1.1.4 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.

Note that you need to build from the project's source folder, e.g:

```
docker build -f extras/docker/development/Dockerfile -t wger/server .
docker build -f extras/docker/demo/Dockerfile --tag wger/demo .
```

Production

There is a configured docker compose file with all necessary services

https://github.com/wger-project/docker

Demo

Self contained demo image

Get the image:

```
docker pull wger/demo
```

Run a container and start the application:

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

Then just open http://localhost:8000 and log in as: **admin**, password **adminadmin**

Please note that the database will not be persisted when you update the image

Development

This image installs the application using virtualenv, uses an SQLite database and serves it with Django's development server.

Get the image:

```
docker pull wger/server
```

Run a container and start the application:

```
docker run -ti --name wger.dev --publish 8000:8000 wger/server
(in docker) python manage.py runserver 0.0.0.0:8000
```

Then just open http://localhost:8000 and log in as: **admin**, password **adminadmin**

As an alternative, 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 do this:

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

It will mount the local path *on top* of the folder in the container. For this to work you obviously need to manually checkout the code to `/path/to/local/wger/` and create a settings file as well.

1.1.5 Updating the application

To keep the application updated you need to regularly perform the following steps well you pull from upstream.

Python dependencies

To install updated python dependencies:

```
source /path/to/venv/bin/activate
python manage.py migrate
```

Upgrade the database

There are regular changes and upgrades to the database schema (these may also come from new versions of Django or the installed dependencies). If you start your server and see a message that there are unapplied migrations, just do:

```
python manage.py migrate --all
```

You might want to save a dump of the database before applying the migrations in case something happens.

JS and CSS dependencies

We use yarn to download the JS and CSS libraries. To update them just run the following command on the source folder:

```
yarn install
yarn build:css:sass
python manage.py collectstatic # only needed in production
```

Pull new data

You can pull new exercise data and ingredients but remember that new ingredients can overwrite the ones you added manually:

```
python3 manage.py sync-exercises
python3 manage.py download-exercise-images
wger load-online-fixtures
```

CHAPTER

TWO

DEVELOPMENT

2.1 Tips and tricks

2.1.1 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` or just set the timeout to something like one second (in settings.py:

```
CACHES = {
'default': {
    'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
    'LOCATION': 'wger-cache',
    'TIMEOUT': 1
    }
}
```

2.1.2 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'
```

2.1.3 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 are dummy data generator scripts for the different entry types:

```
dummy-generator-users
dummy-generator-gyms
dummy-generator-body-weight
dummy-generator-nutrition
dummy-generator-measurement-categories
dummy-generator-measurements
dummy-generator-workout-plans
dummy-generator-workout-diary
```

Alternatively, you can just call *dummy-generator* which will in turn call the other scripts with their default values

For help, just do:

```
python3 manage.py dummy-generator-<name> --help
```

Note: All generated users have their username as a 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.

2.1.4 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
```

2.1.5 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 work. It's recommended that you work on a feature branch when working on something, especially 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 appropriate tests for the new code.
- **Code according to the coding style**: *Coding Style Guide*

2.2 Coding Style Guide

2.2.1 Python

Code according to PEP8, but with a maximum line length of 100.

2.2.2 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`

2.2.3 Automatic coding style checks

The coding style is automatically checked by GitHub actions after sending a pull request.

2.3 Internationalization (i18n)

2.3.1 Django

Updating the translation files

wger uses Django's translation infrastructure, but there are a couple of things that need to be considered. First, you need to extract some translatable strings from the database such as exercise categories and muscle names:

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

Then, update your po files with the usual Django command (run this in the wger sub folder, not the root one):

```
django-admin makemessages --all --extension py,html,tpl
```

and finally, you can compile them with:

```
django-admin compilemessages
```

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.

2.3.2 React

The frontend that is built with react uses its own translation files. See the README.md file in the locale folder in the react repo for more information.

CHAPTER

THREE

ADMINISTRATION GUIDE

3.1 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.).

3.1.1 Administration Commands

The application provides several administration and bootstrapping 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
load-online-fixtures    Downloads fixtures from server and installs them (at the moment␣
↪only ingredients)
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
```

(continues on next page)

(continued from previous page)

```
  -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 fixtures

Command: **load-fixtures**

Loads all fixture files 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.

Note that ingredients are not included and need to be installed separately with download-online-fixtures.

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
```

Load online fixtures

Command: **load-online-fixtures**

Downloads ingredient and weight units fixtures and installs them. They are not included in the repository due to size.

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-online-fixtures [--options] [other tasks here ...]

Docstring:
  Downloads fixtures from server and installs them (at the moment only ingredients)

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

3.1.2 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>`:

sync-exercises
: synchronizes the exercise database from the default wger instance to the local installation. This will also update categories, equipment, languages, muscles and will delete entries that were removed on the remote server (this basically only applies to exercises that were submitted several times). Exercises that you added manually to the database are not touched.

download-exercise-images
: synchronizes the exercise images from the default wger instance to the local installation

download-exercise-videos
: synchronizes the exercise videos from the default wger instance to the local installation

extract-i18n
: Used for development only. Extracts strings from the database that need to be translated

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.

exercises-health-check.py
: Performs a series of basic health checks. Basically sees if there are exercises that don't have a default English translation or worse, don't have any translation at all

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 a user that need to create a new workout.

email-weight-reminders
sends out email reminders for a 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.

3.2 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_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.

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

DOWNLOAD_INGREDIENTS_FROM: Default `WGER`
Where to download ingredient images from. Possible values are 'WGER' and 'OFF'. It is recommended to leave the default of WGER as to now hit the Open Food Facts with too many requests

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

EXERCISE_CACHE_TTL: Default `3600`
Sets how long the overview responses for exercise, exerciseinfo and exercisebaseinfo are cached for. The value is in seconds, so 3600 is one hour.

MIN_ACCOUNT_AGE_TO_TRUST: Default `21`
Users won't be able to contribute to exercises if their account age is lower than this amount in days.

SYNC_EXERCISES_CELERY: Default `False`
Whether to periodically synchronize the exercise database from the default wger instance. Needs celery to be configured.

SYNC_EXERCISE_IMAGES_CELERY: Default `False`
Whether to periodically synchronize the exercise images from the default wger instance. Needs celery to be configured.

SYNC_EXERCISE_VIDEOS_CELERY: Default `False`
Whether to periodically synchronize the exercise videos from the default wger instance. Needs celery to be configured.

USE_CELERY: Default `False`.
Whether celery is configured and can be used

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

WGER_INSTANCE: Default `https://wger.de`.
The wger instance from which commands like exercise synchronization will use to fetch data from.

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.

Set the URL for your site in the table `django_site`. This is only used e.g. in the password reset emails.

3.3 Gym administration

wger provides support for managing both gyms and members. For example, trainers/coaches can follow their students progress, and gym managers are able to keep track of their members contracts.

If the installation is being used for a single gym, 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 members but is not allowed to see the members' workout data.
- **Gym Manager:** Can manage the users for a single gym (editing, deactivating, and 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.

3.3.1 Member Management

To add members to a gym:

1. Click the `Add Member` button at the top of the member overview.
2. Fill in the form to generate a password for the user.
3. Save this password and give it to the user and it cannot be retrieved later.

   OR

1. Click the `Add Member` button at the top of the member overview.
2. Instruct new members to use the reset password links when logging in for the first time.

To export all gym members:

1. Navigate to the `Actions` button on the gym detail page.
2. Here, you are provided with 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. Additionally, 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.

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.

3.3.2 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.

3.3.3 E-Mails

You can send a batch e-mail to all members of a gym. Currently, there is no support for filtering members based on specific criterion.

How to send e-mails:

1. Navigate to the gym's overview and click "Add" on the Email actions button.
2. Fill in the subject and the body.
3. Review, and accept the e-mail's preview.
4. After submitting, emails will be delivered in batch format based on your cron jobs configuration.

3.3.4 Configuration

Inactive members

Inactive members are members that have not logged in for X weeks. For example, a trainer can check to see which users have not visited the gym in X weeks.

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 for any other reason.

Gym name in the 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

FOUR

CHANGELOG

4.1 Changelog

4.1.1 2.3 - IN DEVELOPMENT

Upgrade steps from 2.2

- Update python libraries `pip3 install -r requirements.txt`
- Migrate database `python manage.py migrate`
- Update CSS and JS libraries `yarn install`
- Compile the CSS `yarn build:css:sass`
- Update static files (only production): `python3 manage.py collectstatic`

Features

-

Maintenance

- Added UUID fields to exercise aliases and comments for better synchronization

Bug Fixes

-

4.1.2 2.2

2023-11-06

https://github.com/wger-project/wger/releases/tag/2.2

Upgrade steps from 2.1

- Update python libraries `pip3 install -r requirements.txt`
- Migrate database `python manage.py migrate`
- Update CSS and JS libraries `yarn install`
- Compile the CSS `yarn build:css:sass`
- Update static files (only production): `python3 manage.py collectstatic`
- Load new permissions `python3 manage.py loaddata groups.json categories.json`
- Read the section on celery in this documentation on how to set it up. While at the moment this is not needed and only provides quality of life features, in the future this might change

Features

- Improvements to the nutritional plan handling. Users don't have to setup a detailed plan with meals anymore, instead they can just log their meals #817
- Allow users to set goals for their nutritional plans. This basically works like the sum of the individual meals, but is simpler and easier to setup #1003
- Implemented nutrition page with react
- Added general measurements tracking to the web application #875
- Added JWT authentication to the REST API (thanks @RohanKaran!) #1047
- Added images to ingredients. This can now be shown in the nutritional plan, the autocompleter, etc. #653
- Add a celery queue for longer running or periodic tasks. At the moment this is only used to keep the exercises in sync and download the ingredient images, but other features are planned #1174
- When scanning a product, fetch the data from the live OFF server if it is not found locally #1012, #1348
- Added brute protection against brute force login attacks (thanks @RohanKaran!) #1096
- Reworked the landing page (thanks @12people!) #1112
- Allow to set the minimum account age for users to contribute exercises (thanks @mohammadrafigh!) #1187
- Document the API with openAPI, redoc and all the goodies that come from it (better online docs, being able to generate clients, etc.) #1127
- Allow searching exercises and ingredient in English in addition to the user's currently selected language #1238
- More flexible user (sub)locale switching. This specially affected English users that would be shown dates in US format #1245
- Add a deletion log for exercises. This allows exercises to be marked as deleted by the system. Alternatively a replacement can be set so that when other instances sync the exercises logs and routines are correctly updated #1237
- Show all the authors of an exercise and any of its child items (translations, images, videos, etc.) #1137
- Allow users to give meals a description (thanks @mohammadrafigh!) #822
- Added style field (Foto, 3D, etc.) to exercise image (thanks @LucasSD!) #822
- Added exercise edit history (thanks @ImTheTom!) #1082
- Added JWT authentication to rest API (thanks @RohanKaran!) #1134

- Add support for sub-locales in the application such as en-gb #1275
- Moved some parts of routine management to react #1328

Maintenance

- Improvements to the Open Food Facts product importer. The setup has been simplified with a docker compose, making the process much more streamlined. #1505
- #1137 (thanks @AdamPetik!)
- Show last modified datetime of exercises in the API #1387
- Better handling of exercises without translations #1319
- Split the dummy generator into individual files #919
- Update bootstrap to current version #1109
- Update django to current version #1110
- Bettler handling of exercises UUIDs (thanks @Gr8ayu) #675
- Add foreign key to meals on log (thanks @Alig1493) #842
- Make URL for media, static and login redirect configurable (thanks @novalis111) #1020
- Configure django axes (thanks @RohanKaran) #1143
- Add tzdate package to docker base image (thanks @bbkz) #1408

Bug Fixes

- Fix issue with django axes and mobile app #1163
- Correctly format decimal places in numbers according to the user's locale #1402
- Fix issue when a user tried to register with an existing email via the app (thanks @JayanthBontha!) #1459
- Fix bug in the demo entries generator (thanks @JayanthBontha!) #1278
- Fix issue with password reset links and expired tokens (thanks @RohanKaran!) #1154
- Fix issue with password reset links and expired tokens (thanks @RohanKaran!) #1287
- Fix issue that prevented users from resetting their password (thanks @RohanKaran!) #1154
- Fix import error (thanks @sophiamartelli!) #986
- Use either TLS or SSL for emails (thanks @bbkz!) #1514
- Fix bug in the link used in the password reset link #1320
- Fix bug in the weight log chart #1308

4.1.3 2.1

2022-10-11

Upgrade steps from 2.0:

- Install ffmpeg if you want to upload videos (consult documentation).
- Update python libraries `pip3 install -r requirements.txt`
- To sync the new exercises:
  - Run migrations `python3 manage.py migrate`
  - delete all exercises not in use `python manage.py delete-unused-exercises` (this will delete all exercises that are currently in the database but are not part of any workout, log, etc. You will be promted before the script does anything)
  - get the new exercises `python manage.py sync-exercises` (Also note that if you don't perform these steps and directly run a regular sync the worst that can happen is that you might have some duplicate exercises in your installation)
  - get the new images `python manage.py download-exercise-images`
  - get the new videos `python manage.py download-exercise-videos` (please note that this needs more space)
- Update CSS and JS libraries `yarn install`
- Compile the CSS `yarn build:css:sass`
- Update static files (only production): `python3 manage.py collectstatic`
- Load new permissions `python3 manage.py loaddata groups.json categories.json`

Features:

- The exercise database has undergone a huge cleanup, combining duplicates and translations, deleting stubs, etc. Refreshed the UI for the exercise overview, detail view and contribution page. It is now easier (or at all possible) to submit, correct and translate the exercises. #1120
- New gallery where users can upload pictures to track their progress #572
- Exercises can now have videos. Also many thanks to Goulart for providing 150 videos #970 and releasing them under the CC-BY-SA license.
- Add templates / centrally managed workouts (thanks @qwert45hi) #639
- Add comment filed to set for user notes #702
- Custom measurements such as biceps size or body fat #133
- Add picture type to exercise images (thanks @LucasSD) #589
- Add optional relation from nutritional diary to meal (thanks @Alig1493) #819
- Muscles now have a "common" name, besides their names in Latin (thanks @ImTheTom) #1041
- Allow to add nutritional plan diary entries for other dates (thanks @ImTheTom) #520

Bug Fixes:

- Adding a new workout day no longer needs to be saved twice (thanks @ImTheTom) #974

Maintenance:

- Exercise API response is now cached (thanks @ImTheTom) #1033

- Changes to the REST API:
  - /exercisebaseinfo/ - New endpoint to get exercise information grouped by the base exercise
  - /language/ - Also expose the language ID
  - /exerciseimage/ - `exercise` was renamed to `exercise_base` (was pointing there anyway) - New field `style`
  - /workout/ - `comment` was renamed to name - field `description` was added, for longer descriptions
  - /set/ - field `comment` added, for user notes
  - /nutritiondiary/ - field `meal` added, optional reference to meal
  - /min-app-version/ - New endpoint indicating minimum required version for flutter app
- #666, #667, #656 (thanks @jackmulligan-ire), #716

4.1.4 2.0

2021-05-01

Upgrade steps from 1.9:

- Update python libraries `pip3 install -r requirements.txt`
- Install `yarn` and `sass` (e.g. `sudo npm install -g yarn sass`)
- Update CSS and JS libraries `yarn install`
- Compile the CSS `yarn build:css:sass`
- Run migrations `python3 manage.py migrate`
- Update data `python3 manage.py loaddata licenses.json languages.json language_config.json`
- Load new ingredients (note that this will overwrite any ingredients that you might have added) `wger load-online-fixtures`
- Update static files (only production): `python3 manage.py collectstatic`
- Subcommands for `wger` now use dashes in their names (i.e. create-settings instead of create_settings)

Features:

- Add nutrition diary to log the daily calories actually taken #284, #501 and #506 (thanks @WalkingPizza and @oconnelc)
- Support for reps-in-reserve (RiR) in workout plans and logs #479 (thanks @SkyNetIndustry)
- Improved user experience, on desktop and mobile #337
- Around 70000 new ingredients with Open Food Facts import with more to come #422 (thanks @harlenesamra, @nikithamurikinati and @jcho1)
- Group common exercise information such as muscles, etc. for more easy translations, data management, etc. #448 (thanks @nikithamurikinati, @harlenesamra, @jcho17, @vaheeshta and @jeevikaghosh)
- Group similar exercises such as wide grip, reverse, etc. #555 (thanks @ryowright)
- Improved info endpoints for exercises and ingredients, this saves additional API calls #411
- Show BMI on weight graph #462 (thanks @Svn-Sp)
- Allow user to edit and delete body weight entries #478 (thanks @beingbiplov)

- Show kJoules as well as kcal in nutritional plan #568 (thanks @nopinter and @derekli17)
- Check name similarity when adding exercises to avoid duplicates #551 (thanks @lydiaxing, @eq8913, @Hita-K)
- Return the muscle background images in the REST API #547 (thanks @gengkev)

Bug Fixes:

- #368, #379, #426 (thanks @austin-leung), #499, #505, #504, #511, #516, #522, #554 and #560 (thanks @sandil-sranasinghe), #564, #565, #615, #560 (thanks @bradsk88), #617 (thanks @Sidrah-Madiha), #636, #640, #642, #648, #650

Maintenance:

- Moved translations to weblate #266
- Improved docker and docker-compose images #340
- Updated many libraries to the last version (bootstrap, font awesome, etc.)
- Use yarn to download CSS/JS libraries
- Improvements to documentation (e.g. #494)
- Improved cache handling #246 (thanks @louiCoder)
- Others: #450 (thanks @Rkamath2), #631 (thanks @harlenesamra), #664 (thanks @calvinrw),

4.1.5 1.9

2020-06-29

Upgrade steps from 1.8:

- Django update to 3.x: `pip install -r requirements.txt`
- Database upgrade: `python manage.py migrate`
- Update static files (only production): `python manage.py collectstatic`

New features:

- Allow users to enter their birthdate instead of just the age (thanks @dtopal) #332
- Work to ensure that mobile templates are used when appropriate
- Added optional S3 static asset hosting.
- Drop Python 2 support.
- Replaced django-mobile with django-user_agent (and some custom code) This isn't as slick as django-mobile was, but it unblocks possible Django 2.x support.
- Update many dependencies to current versions.

Improvements:

- Improve the 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

- Work to slim docker image * Download images at startup - If *DOWNLOAD_IMGS* environmental variable is set to *TRUE* * Uninstall dev packages
- Update Ubuntu version used in docker container.
- Fixed a handful of hard coded static path references to use *static* taglib
- Updated tinymce theme for v5.

Other improvements and bugfixes: #336, #359,`#386`_, #443

4.1.6 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 the 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

4.1.7 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 nutritional 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

4.1.8 1.6.1

2015-07-25

Bugfix release

4.1.9 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 the 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

4.1.10 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 the 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

4.1.11 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.)

4.1.12 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

4.1.13 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

4.1.14 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

4.1.15 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

4.1.16 1.0.3

2012-11-19

New features and bugfixes:

- Add option to copy (duplicate) workouts and nutritional plans
- Login without an account with Mozilla's Persona (BrowserID)
- Better AJAX handling of the modal dialogs, fewer page reloads and redirects
- Expand the list of ingredients in German
- Add pagination to the 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

4.1.17 1.0.2

2012-11-02

Bugfix release

4.1.18 1.0.1

2012-11-02

New features and bugfixes:

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

4.1.19 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

FIVE

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).

- **Discord:** https://discord.gg/rPWFv6W
- **Issue tracker:** https://github.com/wger-project/wger/issues
- **Mastodon:** https://fosstodon.org/@wger

CHAPTER

SIX

SOURCES

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

CHAPTER

SEVEN

LICENCE

The application is licenced under the Affero GNU General Public License 3 or at your choice any later version (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 were taken from Wikipedia, see the SOURCES file in their respective folders for more details.