wger Workout Manager Documentation

Release 2.6 alpha

Roland Geider

Jun 02, 2026

CONTENTS

wger (v) 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

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

User manual
: User-facing manual: how to use routines and progression rules.

Installation
: Setting up wger in production. Docker is the recommended path.

Administration
: Day-to-day operations: updates, backups, settings, commands.

Development
: For developers who want to work on wger itself (backend, frontend, mobile app).

API
: REST API reference for client integrations.

Contributing
: How to contribute code, exercises, ingredients or translations.

Changelog
: Release notes and version history.

CHAPTER

ONE

USER MANUAL

User manual for the wger app. Currently only a chapter for the routines is available.

1.1 Using the routines

A routine is a planned sequence of workouts. It automatically tells you which workout you have to do next, so for example, if you did an "arms" workout today, a routine can remind you that "chest and shoulders" is next.

> **Note**
>
> The routines are made around the assumption that you will do the same workout for some weeks. If you prefer to do ad-hoc workouts with whatever exercises you feel, you will not be happy with this app!

1.1.1 Workout days

The basic building block of a routine is a **day**. A "day" can either be a regular workout day with a list of exercises, or it can be a designated **rest day**.

You have two main options for setting up the timing and sequence of the workout (in either case, it's probably better to make sure the first day of the routine is a Monday, but it's not necessary):

Fit the routine in a week

If you want a consistent weekly schedule, like working out every Monday, Wednesday, and Friday or just "three times a week" without any specific weekdays, simply add your workout days (e.g., three different workout days) and enable the `fit in week` toggle. Your routine will then restart every week on the same day.

If using this mode, it's probably a good idea to activate the "needs logs to advance" toggle (see next section).

Create a custom cycle

This option gives you maximum flexibility to create routines with unique lengths and periodizations. You can explicitly add all your workout and rest days in the exact order you want them to repeat, even if they don't fit into one week.

For example a cycle of `Day A` → `Day B` → `Rest` → `Rest` → `Day C` → `Rest` will repeat in that exact sequence, repeating every 6 days.

1.1.2 Advancing through the routine

By default, the routine advances to the next day automatically. However, you can control this behavior.

If you want to ensure you complete a workout before the routine moves on, you can enable the `needs logs to advance` toggle for any workout day. When this is active, that day remains the active one until you log at least one set for any of its exercises. This is useful for making sure you don't miss any sessions.

Note that this toggle does not need to be applied to all days. For example: you have a routine with a "Monday group class" workout day, but for the others you go alone. If you can't attend one week, leaving the toggle off for that day allows the routine to advance automatically. This keeps your schedule on track without requiring you to log a session you didn't do.

The currently active day in the routine is then marked with a calendar icon on the dashboard.

1.1.3 Sets and exercises

A workout day is a collection of exercises you plan to do in a single session. You can add as many sets as you need. By default, the interface is kept simple, if you need more control, you can use the `simple mode` toggle. This will reveal more options, allowing you to:

- Set other units (e.g., minutes, distance).
- Specify an explicit rest time between sets.
- Specify a min and max value for the settings

Supersets

If you add more than one exercise to a set, it automatically becomes a superset. When you run the workout in gym mode, exercises are presented in an alternating sequence. Also note that not all exercises need to have the same number of sets, e.g.:

- Exercise 1, 4 sets
- Exercise 2, 2 sets
- Exercise 3, 3 sets

Would result in:

- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 1
- Exercise 3
- Exercise 1

(with their respective values for weight, reps, etc.)

1.1.4 Progression rules

> **Note**
>
> This feature is currently only available on the **web version**. However, the resulting routines can be used and viewed on the mobile app as well.

For more advanced control over your progress, you can set rules for automatic changes in weight, repetitions, number of sets or RiR. You have the option to

- add or substract a fixed value
- add or substract a percentage (in this case it's probably a good idea to also set a rounding value)
- reset the value to a specified number

in any future week.

You can also set the `repeat rule` toggle so that the rule will be repeated until a new rule is encountered.

In case you don't want these values to change automatically, you can also configure their `requirements`, which are rules that make it so that a planned progression only takes effect if certain conditions are met.

For example you can create a rule to increase the weight of an exercise by 1.25kg every week. You can then set a requirement that this increase only happens if you successfully completed all the planned reps and weight from a previous session. If you didn't meet the requirement, the weight will stay the same until you do.

Using the routines
How to use the routines and the progression rules.

CHAPTER

TWO

INSTALLATION

It is *very* recommended to use the Docker image for production: everything is already set up and configured, and it's what we use for the official wger.de server. The other installation paths exist for cases where Docker isn't an option, but they're not as extensively tested or are submitted by the community.

2.1 Docker compose

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

The prod docker compose file starts up gunicorn as the application server, postgres as the database, redis for caching, and nginx as a reverse proxy.

The database, static files and uploaded images are mounted as volumes so the data is persisted. The only thing you need to do is update the docker images. Consult the docker volume command for details on how to access or backup this data.

It is recommended to regularly pull the latest version of the compose file, since sometimes new configurations or environmental variables are added.

If after installing not everything works, consult the *Common errors and pitfalls* section for common errors and how to fix them.

2.1.1 Quickstart

To start all services:

```
docker compose up -d
```

Then open http://localhost (or your server's IP) and log in as: **admin**, password **adminadmin**.

> **Warning**
>
> If your instance is reachable over the internet, change the default password after logging in for the first time.

2.1.2 Configuration

Instead of editing the compose file or the env file directly, it is recommended to extend it. That way you can more easily pull changes from this repository.

For example, you might not want to run the application on port 80 because some other service in your network is already using it. For this, simply create a new file called `docker-compose.override.yml` with the following content

```
services:
  nginx:
    ports:
      - "8080:80"
```

Now the port setting will be overwritten from the configured nginx service when you do a `docker compose up`. However, note that compose will concatenate both sets of values so in this case the application will be binded to 8080 (from the override) *and* 80 (from the regular compose file).

In Docker Compose 2.24.4 and later, you can fully override values using the !override yaml directive. i.e.:

```
services:
  nginx:
    ports: !override
      - "8080:80"
```

The same applies to the env variables, just create a new file called e.g. `my.env` and add it after the provided `prod.env` for the web service (again, this is `docker-compose.override.yml`). There you add the settings that you changed, and only those, which makes it easier to troubleshoot, etc.

```
web:
  env_file:
    - ./config/prod.env
    - ./config/my.env
```

To add a web interface for the celery queue, add a new service to the override file

```
celery_flower:
  image: wger/server:latest
  container_name: wger_celery_flower
  command: /start-flower
  env_file:
    - ./config/prod.env
  ports:
    - "5555:5555"
  healthcheck:
    test: wget --no-verbose --tries=1 http://localhost:5555/healthcheck
    interval: 10s
    timeout: 5s
    retries: 5
  depends_on:
    celery_worker:
      condition: service_healthy
```

For more information and possibilities consult https://docs.docker.com/compose/extends/ and https://docs.docker.com/reference/compose-file/merge/

2.1.3 Deployment

The easiest way to deploy this application to prod is to use a reverse proxy like nginx or traefik. You can change the port this application exposes and reverse proxy your domain to it. For this just edit the "nginx" service in docker-compose.yml and set the port to some value, e.g. `"8080:80"` then configure your proxy to forward requests to it, e.g. for nginx (no other ports need to be changed, they are used only within the application's docker network).

The docker repo ships a Caddyfile.example for users who'd rather use Caddy than nginx in front of the application. Caddy automatically obtains and renews SSL certificates from Let's Encrypt.

Also notice that the application currently needs to run on its own (sub)domain and not in a subdirectory, so `<domain>/wger` will probably only mostly work.

2.1.4 Next steps

Once your installation is running, see the *Administration* section for ongoing operations.

2.2 Kubernetes

Helm charts for deploying wger to a Kubernetes cluster are available in the wger-project/helm-charts repository.

The charts are maintained by a community contributor and kept current with new wger releases. For installation and configuration details, see the repository's README.

For issues with the charts themselves (templating, values, install errors), open an issue against wger-project/helm-charts. For issues with wger itself, open them against the main wger repository instead.

2.3 Installation from source

Install `npm` / `nodejs` >= 22, and `sass`

If you plan to use video files, it's recommended you install ffmpeg as well (this is not strictly necessary, it's just used for better upload validation as well as the extraction of some information from the video files such as resolution, codec, etc):

```
apt-get install ffmpeg
pip install ffmpeg-python
```

2.3.1 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.

2.3.2 Database

PostgreSQL

Install the Postgres server, then create a database and a user:

```
sudo apt-get install postgresql
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";
```

The `postgresql` meta-package pulls in whatever currently-supported version your distro packages, any Postgres 15 should work.

You might need to edit your `pg_hba.conf` to allow local socket connections.

SQLite

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

```
mkdir /home/wger/db
touch /home/wger/db/database.sqlite
chown :www-data -R /home/wger/db
chmod g+w /home/wger/db /home/wger/db/database.sqlite
```

2.3.3 Application

Install uv if you don't have it already, then clone the repository and install wger with the `docker` dependency group. This pulls in gunicorn alongside the base dependencies (for the systemd unit we'll create in the Webserver section below):

```
git clone https://github.com/wger-project/wger.git /home/wger/src
cd /home/wger/src
uv sync --group docker --no-managed-python
```

The `--no-managed-python` flag tells uv to use your system's Python (installed via apt or similar) instead of downloading its own. Drop the flag if you'd rather have uv manage it.

uv creates the virtualenv at `/home/wger/src/.venv` automatically. Activate it for the bootstrap commands below:

```
source /home/wger/src/.venv/bin/activate
```

Create folders for static resources and uploaded files. The `static` folder contains generated CSS and JS files (read by Caddy), the `media` folder receives uploads (read by Caddy, written by gunicorn/django):

```
mkdir /home/wger/{static,media}
chmod o+w /home/wger/media
```

Configuration

wger reads its configuration from environment variables. Place them in `/home/wger/wger.env` so the systemd unit we'll define in the Webserver section can pick them up (via its `EnvironmentFile=` directive), and so you can source them into a shell when running one-off commands.

For a complete commented list of available settings, use the Docker setup's `prod.env` as a reference: https://github.com/wger-project/docker/blob/master/config/prod.env

A minimal example:

```
# /home/wger/wger.env

# Change these
ALLOWED_HOSTS=example.com,www.example.com
DJANGO_SECRET_KEY=your-very-long-and-random-secret-key
TIME_ZONE=Europe/Berlin

# Application
MEDIA_ROOT=/home/wger/media
STATIC_ROOT=/home/wger/static

# Django Setup
DJANGO_SETTINGS_MODULE=settings.main
```

(continues on next page)

(continued from previous page)

```
PYTHONPATH=/home/wger/src

# Postgres
DJANGO_DB_ENGINE=django.db.backends.postgresql
DJANGO_DB_NAME=wger
DJANGO_DB_USER=wger
DJANGO_DB_PASSWORD=wger
DJANGO_DB_HOST=localhost
DJANGO_DB_PORT=5432
```

For SQLite, swap the database block for:

```
DJANGO_DB_ENGINE=django.db.backends.sqlite3
DJANGO_DB_NAME=/home/wger/db/database.sqlite
```

The file contains secrets, so restrict permissions:

```
sudo chown wger:wger /home/wger/wger.env
sudo chmod 600 /home/wger/wger.env
```

Note

The file uses systemd's `EnvironmentFile` syntax: `KEY=VALUE` per line, no `export` prefix and no spaces around `=`. Quotes around values are usually unnecessary; if you do need them, use single quotes.

Bootstrap

For one-off commands (bootstrap, migrate, collectstatic, ...) the env file needs to be sourced into the current shell. As the wger user with the virtualenv active:

```
set -a; . /home/wger/wger.env; set +a
```

Run the installation script, this downloads JS/CSS libraries and loads initial data:

```
wger bootstrap
```

Collect all static resources:

```
python manage.py collectstatic
```

Compile the translation (.po) files:

```
cd wger
django-admin compilemessages
```

The bootstrap command also creates a default administrator user. **Change the password immediately after first login:**

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

2.3.4 Webserver

The recommended setup runs **gunicorn** as the WSGI application server, with **Caddy** as a reverse proxy in front of it that also serves the static and media files. This mirrors how the Docker image is configured internally.

If you prefer nginx, Apache or another webserver, the building blocks are the same, just set up a reverse proxy that forwards to gunicorn and serves `/static/` and `/media/` directly. See Django's deployment guide: https://docs.djangoproject.com/en/dev/howto/deployment/

gunicorn

Create a systemd service file at `/etc/systemd/system/wger.service`:

```
[Unit]
Description=wger gunicorn daemon
After=network.target

[Service]
User=wger
Group=wger
WorkingDirectory=/home/wger/src
EnvironmentFile=/home/wger/wger.env
ExecStart=/home/wger/src/.venv/bin/gunicorn wger.wsgi:application \
    --preload \
    --bind 127.0.0.1:8000 \
    --workers 3 \
    --threads 2 \
    --worker-class gthread \
    --timeout 240 \
    --access-logfile -

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

The worker count of `(2 × $num_cores) + 1` is a common starting point. The unit reads your env vars from `/home/wger/wger.env` via `EnvironmentFile`.

Reload systemd and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable --now wger
```

Check that it's running with `systemctl status wger` and `journalctl -u wger -f`.

Caddy

Install Caddy following the official instructions: https://caddyserver.com/docs/install

Create `/etc/caddy/Caddyfile`:

```
your-domain.example.com {
    encode

    reverse_proxy 127.0.0.1:8000 {
        header_up Host {host}
        header_up X-Real-IP {remote_host}
        header_up X-Forwarded-For {http.X-Forwarded-For} {remote_host}
        header_up X-Forwarded-Proto {scheme}
```

(continues on next page)

(continued from previous page)

```
        header_up X-Http-Version {http.request.proto}
    }

    handle /static/* {
        root * /home/wger
        header Cache-Control "public, max-age=31536000, immutable"
        file_server
    }

    handle /media/* {
        root * /home/wger
        file_server
    }
}
```

Caddy automatically obtains and renews a Let's Encrypt certificate as long as the domain's DNS points to your server. Reload Caddy to pick up the config:

```
sudo systemctl reload caddy
```

The `X-Forwarded-*` headers are important, without them, generated URLs (e.g. pagination links in the API) and CSRF protection can break. If you see CSRF errors after setup, check *Common errors and pitfalls*.

Note

If `/static/` or `/media/` return 403 or 404, check whether the webserver's systemd unit has `ProtectHome=true`, as it blocks reads from `/home/wger/`. Override the unit:

```
sudo systemctl edit caddy
```

and add:

```
[Service]
ProtectHome=off
```

Then reload: `sudo systemctl daemon-reload && sudo systemctl restart caddy`.

2.3.5 Next steps

Once your installation is running, see the *Administration* section for ongoing operations.

2.4 TrueNAS SCALE

> **Note**
>
> this guide was provided by @Phyxsius7 and was copied over from issue 130

This guide shows how to install wger on TrueNAS SCALE using your own datasets (no ixVolume) and no shell. It covers only dataset settings and wger app settings.

2.4.1 Dataset layout

Create these four child datasets under your preferred parent (example shown):

```
/mnt/tank/configs/wger/
├─ static
├─ media
├─ beat
└─ pgdata
```

Mount only the four children in the app. The parent wger is for organization.

2.4.2 Step 1 – Create datasets (UI)

1. Datasets -> select your parent (e.g., `tank/configs`).
2. Add Dataset -> Name: wger -> Save.
3. Select `.../wger` -> Add Dataset four times for: `static`, `media`, `beat`, `pgdata`. Keep *Share Type* = *Generic* and defaults.

2.4.3 Step 2 – Set dataset permissions (UI)

Do the following *for each child*: `static`, `media`, `beat`, `pgdata`.

1. Datasets -> select the dataset -> … -> Edit Permissions.
2. If you see "Set ACL", click it and "Strip ACL" (switch to Unix / POSIX mode).
3. In "Unix Permissions Editor": * Leave "User" and "Group" as they are (don't change owners). * Under "Access", check `Read + Write + Execute` for `User / Group / Other` -> this is 0777. * Tick "Apply permissions recursively".
4. Save.

Rationale: wger/NGINX run as a non-root user (UID 1000) and Postgres as UID 999. `0777` ensures all containers can write without managing UIDs.

Optional: (You can harden later by using ACL entries for UID `1000` on `static/media/beat` and UID `999` on `pgdata`.)

2.4.4 Step 3 – Wger app settings (UI)

Open Apps -> Available Applications -> Wger -> Install and set:

A) Wger Configuration

- **Timezone**: `Europe/Amsterdam` (or your TZ)
- **Database Password / Redis Password / Secret Key / Signing Key**: enter values (50 chars for keys)
- **Trusted Origins**: must include at least one full URL you'll use, e.g. `http://<truenas-ip>:30370` (Include scheme and port. Add your hostname here too if you'll use one.)

B) Network Configuration

- **WebUI Port** -> Port Bind Mode: Publish
- **Port Number**: choose a **free** port, e.g. `30370`.

C) Storage Configuration

For each storage below:

- **Type**: Host Path
- **Enable ACL**: OFF
- **Host Path**: point to the matching dataset

Storage field	Host Path	Extra toggle
Wger Static Storage	/mnt/tank/configs/wger/static	—
Wger Media Storage	/mnt/tank/configs/wger/media	—
Wger Beat Storage	/mnt/tank/configs/wger/beat	—
Postgres Data Storage	/mnt/tank/configs/wger/pgdata	Automatic Permissions = ON

Automatic Permissions = ON (only for `pgdata`) lets SCALE set the correct internal ownership for Postgres on first start.

Click Install

2.4.5 Quick checklist (if install fails)

- Each child dataset has POSIX 0777 (User/Group/Other all RWX) and was applied recursively.
- Trusted Origins is not empty and includes a full URL with scheme + port.
- The chosen WebUI Port is not used by another app.

2.5 Proxmox

There is a community-maintained installation script for Proxmox VE:

https://community-scripts.org/scripts/wger

The script lives in the community-scripts/ProxmoxVE repository (`ct/wger.sh` and `install/wger-install.sh`).

This script is **not maintained by the wger project**. We can't guarantee that it stays up to date with current wger releases, and we can't provide support for issues caused by the script itself. For problems with the script, open an issue against community-scripts/ProxmoxVE.

For a setup that we test and support, see *Docker compose*.

Docker compose
: Recommended production setup, uses Docker and Docker Compose.

Kubernetes
: Helm charts for deploying to a Kubernetes cluster.

Installation from source
: Installation from source, when Docker isn't an option.

Community-contributed installation guides:

TrueNAS SCALE
: Installation guide for TrueNAS SCALE.

Proxmox
: Proxmox installation script.

CHAPTER

THREE

ADMINISTRATION

This section describes configurations, commands and operations relevant for administrators running a wger instance.

> **Note**
>
> Most examples on the following pages assume a Docker compose setup, since that is the recommended deployment. If you installed wger differently, just adapt the commands to your environment (e.g., `docker compose exec web ./manage.py ...` might become `./manage.py ...`).

3.1 Lifecycle

Day-to-day commands for stopping, starting and inspecting your wger installation.

3.1.1 Basic commands

To stop all services while preserving containers and volumes:

```
docker compose stop
```

To start everything up again:

```
docker compose start
```

To remove all containers (volumes are kept):

```
docker compose down
```

To view the logs:

```
docker compose logs -f
```

You might also occasionally need to run other commands inside the containers, e.g.:

```
docker compose exec web python3 manage.py migrate
docker compose exec --user root web /bin/bash
docker compose exec db psql wger -U wger
docker compose exec cache redis-cli FLUSHALL
```

3.1.2 Auto-start with systemd

Once your installation is running, you'll typically want it to auto-restart when the server reboots. With systemd, create `/etc/systemd/system/wger.service` (verify the absolute path of the docker binary with `which docker`):

```
[Unit]
Description=wger docker compose service
PartOf=docker.service
After=docker.service

[Service]
Type=oneshot
RemainAfterExit=true
WorkingDirectory=/path/to/the/docker/compose/
ExecStart=/usr/bin/docker compose up -d --remove-orphans
ExecStop=/usr/bin/docker compose down

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

Reload systemd and start the service:

```
systemctl daemon-reload
systemctl start wger
```

If `systemctl status wger` shows the service as active (this can take a moment), everything's good. Enable auto-start on reboot with:

```
systemctl enable wger
```

3.2 Updating wger

Pull new releases regularly to get bug fixes, new features and security updates.

Docker

Remove the containers and pull the newest images:

```
docker compose down
docker compose pull
docker compose up -d
```

That's it, database migrations and static files are applied automatically on container start.

From source

Pull new changes and apply them in the source tree:

```
git pull
pip install .
python manage.py migrate --all
npm install
npm run build:css:sass
python manage.py collectstatic
```

If you've configured Celery for periodic data sync, the exercise and ingredient datasets stay current automatically. Otherwise, see *Syncing exercises and ingredients* for the manual sync commands.

3.3 Backup

The most important thing to back up is the **Postgres database**, that's where all user data lives. The media volume can usually be re-fetched from upstream, and static files are regenerated on every container start.

You should perform a test restore at least once, so you know the procedure works when you actually need it.

3.3.1 Database

Make a dump of the database:

```
# Stop the other services so the database isn't changed mid-export
docker compose stop web nginx cache celery_worker celery_beat
docker compose exec db pg_dumpall --clean --username wger > backup.sql
docker compose start
```

To restore from a dump:

```
docker compose stop
docker volume remove docker_postgres-data
docker compose up db
cat backup.sql | docker compose exec -T db psql --username wger --dbname wger
docker compose up
```

3.3.2 Media

If you haven't uploaded your own exercise images, exercise videos or gallery images, you don't need to back up the media volume, the contents can be re-downloaded from the upstream wger instance. Truncate the relevant tables and run the sync commands again:

```
docker compose exec db psql -U wger -c "TRUNCATE TABLE exercises_exerciseimage,␣
↪exercises_exercisevideo;"
docker compose exec db psql -U wger -c "TRUNCATE TABLE nutrition_image;"

docker compose exec web python3 manage.py download-exercise-images
docker compose exec web python3 manage.py download-exercise-videos
```

If you do have uploaded media that needs to be preserved, such as gallery entries, consult these options for backing up Docker volumes:

- https://www.docker.com/blog/back-up-and-share-docker-volumes-with-this-extension/
- https://github.com/BretFisher/docker-vackup

3.3.3 Static files

The contents of the static volume are 100% generated and recreated on startup, no need to back up anything.

3.4 Syncing exercises and ingredients

The Docker image ships with a default set of exercises but no ingredients (the dataset is large). You can manually sync both from wger.de or any other instance using these commands.

The application can also be configured to perform these steps automatically in the background via Celery; see the `SYNC_*` options in *Settings*.

> **Note**
>
> Sync commands never overwrite exercises or ingredients you've added yourself to your instance.

3.4.1 Exercises

```
docker compose exec web python3 manage.py sync-exercises
docker compose exec web python3 manage.py download-exercise-images
docker compose exec web python3 manage.py download-exercise-videos
```

3.4.2 Ingredients

The full ingredient dataset takes around 1 GB of database space and needs a while to download and process. For a quick start, load a smaller base set:

```
docker compose exec web wger load-online-fixtures
```

For the full dataset:

```
# On a fresh install:
docker compose exec web ./manage.py sync-ingredients-bulk --set-mode insert

# On an existing install (preserves your additions):
docker compose exec web ./manage.py sync-ingredients-bulk --set-mode update
```

3.5 Monitoring

The docker-compose repository ships with a pre-configured Grafana and Prometheus setup that can be used to monitor the wger application, as well as the logs via Loki and Alloy:

1. Set `EXPOSE_PROMETHEUS_METRICS=True` in your env file and restart the application.
2. Change into the `grafana` folder of the docker-compose repository and start its compose file.
3. Open http://localhost:3000 and log in with username `admin` and password `adminadmin`.

To change the default password, edit `grafana/web.yml`.

3.6 Settings

wger is configured via environment variables. In a Docker setup, these go in your env file (typically extending `prod.env`; see *Docker compose* for the override mechanism). For from-source installations, they go in `/home/wger/wger.env` (see *Installation from source*).

The reference `prod.env` in the docker repository contains commented examples for every option and is the canonical source for what's available: https://github.com/wger-project/docker/blob/master/config/prod.env

This page is the structured reference. For a few specialized topics like S3 storage, SSO via reverse proxy, monitoring, the relevant settings live on their dedicated pages, linked below.

3.6.1 Required

These should be set explicitly for any non-trivial deployment. Leaving the defaults means new keys are generated on every restart, which invalidates sessions and tokens.

SECRET_KEY
Django's secret key, used for cryptographic signing (sessions, CSRF tokens, password reset links, etc.). Generate a random 50-character string:

```
python -c "import secrets; print(secrets.token_urlsafe(50))"
```

SIGNING_KEY
JWT signing key for the API. Use a *different* value than `SECRET_KEY`. Same generation method.

TIME_ZONE / TZ
Server timezone, e.g. `Europe/Berlin`. See https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

ALLOWED_HOSTS
Comma-separated list of hostnames the application accepts requests for, e.g. `example.com,www.example.com`.

SITE_URL
Default `http://localhost`. Public base URL of your instance, no trailing slash. Used to build absolute links in outgoing emails.

3.6.2 CSRF and reverse proxy

If you run wger behind a reverse proxy (recommended), Django needs to know about your domain to accept form submissions. See *Common errors and pitfalls* if you run into CSRF problems.

CSRF_TRUSTED_ORIGINS
Comma-separated list of full URLs (with scheme, and port if non-default), e.g. `https://wger.example.com, https://192.168.1.10:8080`.

X_FORWARDED_PROTO_HEADER_SET
Default `False`. If your proxy sets the `X-Forwarded-Proto` header, set this to `True`. Read the security implications in Django's docs.

NUMBER_OF_PROXIES
Default 1. Number of proxies in front of the application. Used by Django REST Framework's request throttling to determine the real client IP.

3.6.3 URLs and media

MEDIA_URL / STATIC_URL
Override the default `/media/` and `/static/` paths if you serve files from a different host (CDN, S3 see *Storage*).

WGER_PORT
Default `8000`. Port gunicorn binds to inside the container. Rarely changed.

3.6.4 Application

WGER_INSTANCE
Default `https://wger.de`. Upstream wger instance to sync exercises and ingredients from.

ALLOW_REGISTRATION
Default `True`. Whether users can register accounts on their own.

ALLOW_GUEST_USERS
Default `True`. Whether the application offers guest user functionality.

ALLOW_UPLOAD_VIDEOS
Default `True`. Whether users can upload exercise videos.

MIN_ACCOUNT_AGE_TO_TRUST
Default `21` (days). Users below this account age cannot contribute to exercises.

3.6.5 Database

See *Storage* for switching between Postgres and SQLite.

DJANGO_DB_ENGINE
Default `django.db.backends.postgresql`. Use `django.db.backends.sqlite3` for SQLite.

DJANGO_DB_DATABASE
Database name (Postgres) or full path to the SQLite file.

DJANGO_DB_USER, DJANGO_DB_PASSWORD, DJANGO_DB_HOST, DJANGO_DB_PORT
Postgres connection details. Ignored for SQLite.

DJANGO_PERFORM_MIGRATIONS
Default `True`. Apply pending database migrations on container startup.

3.6.6 Cache (Redis)

DJANGO_CACHE_BACKEND
Default `django_redis.cache.RedisCache`.

DJANGO_CACHE_LOCATION
Default `redis://cache:6379/1`. Redis connection URL for the Django cache.

DJANGO_CACHE_TIMEOUT
Default `1296000` seconds (15 days).

DJANGO_CACHE_CLIENT_CLASS
Default `django_redis.client.DefaultClient`.

DJANGO_CACHE_CLIENT_PASSWORD
Optional. Redis password if your Redis is configured with one.

DJANGO_CACHE_CLIENT_SSL_KEYFILE, DJANGO_CACHE_CLIENT_SSL_CERTFILE, DJANGO_CACHE_CLIENT_SSL_CERT_REQS, DJANGO_CACHE_CLIENT_SSL_CHECK_HOSTNAME
Optional TLS settings for Redis connections.

3.6.7 Celery

Background task processing. See *Celery* for the worker setup.

USE_CELERY
Default `True` in the Docker setup (`False` otherwise). Master switch, most `SYNC_*_CELERY` and `CACHE_API_*_CELERY` options require this.

CELERY_BROKER, CELERY_BACKEND
Default `redis://cache:6379/2`. Redis URL for Celery's broker and result backend.

CELERY_WORKER_CONCURRENCY
Default `4`. Set to `1` if using SQLite (which doesn't tolerate concurrent writes well).

CELERY_FLOWER_PASSWORD
Default `adminadmin`. Password for the optional Celery Flower admin UI.

3.6.8 Data sync

See *Syncing exercises and ingredients* for the operational context.

On startup (synchronous, no Celery needed)

`SYNC_EXERCISES_ON_STARTUP`
: Default `False`. Sync the exercise database every time the application starts. Slows startup; runs synchronously.

`DOWNLOAD_EXERCISE_IMAGES_ON_STARTUP`, `DOWNLOAD_EXERCISE_VIDEOS_ON_STARTUP`
: Default `False`. Download exercise images / videos on startup.

`LOAD_ONLINE_FIXTURES_ON_STARTUP`
: Default `False`. Load a small base set of ingredients on startup.

Periodic via Celery

These tasks require `USE_CELERY=True`. Each picks a random hour and minute on Celery worker startup and keeps that schedule until the worker restarts, so two instances won't hammer the upstream wger or Open Food Facts servers at the same time.

`SYNC_EXERCISES_CELERY`, `SYNC_EXERCISE_IMAGES_CELERY`, `SYNC_EXERCISE_VIDEOS_CELERY`
: Default `True` in the Docker setup (`False` otherwise). Run **weekly** on a random day of the week.

`CACHE_API_EXERCISES_CELERY`
: Default `True` in the Docker setup (`False` otherwise). Periodically warms up the exercise API cache. Runs **weekly** on a random day.

`CACHE_API_EXERCISES_CELERY_FORCE_UPDATE`
: Default `True`. Always rebuild the cache on each scheduled run, even if it looks current.

`SYNC_INGREDIENTS_CELERY`
: Default `True` in the Docker setup (`False` otherwise). Downloads the full ingredient dataset as a bulk dump. Runs **monthly** on a random day of the month (1–28).

`SYNC_INGREDIENTS_DUMP_URL`
: Default points to the wger.de dump.

`EXPORT_INGREDIENTS_BULK_CELERY`
: Default `False`. Generates the ingredient bulk dump that can be served to other instances. Runs on the **1st and 15th of every month**. Mainly used by the wger.de instance itself; you don't normally need to enable this.

`SYNC_OFF_DAILY_DELTA_CELERY`
: Default `False`. Imports Open Food Facts' daily delta product updates. Runs **once a day** at a random time. Mainly used by the wger.de instance itself; you don't normally need to enable this.

`DOWNLOAD_INGREDIENTS_FROM`
: Default `WGER` (or `None` to disable). Where to fetch ingredient data from when scanning a barcode for an unknown product. Requires Celery.

Always-on

When `USE_CELERY=True`, expired JWT refresh tokens are flushed from the database **once a day** at a random time. There is no opt-out, without it the token table grows monotonically.

3.6.9 JWT authentication

`ACCESS_TOKEN_LIFETIME`
: Default `10` (minutes). Short-lived access token.

`REFRESH_TOKEN_LIFETIME`
: Default `2880` (hours, 4 months). Long-lived refresh token.

For SSO via reverse proxy (`AUTH_PROXY_*`), see *Authentication Proxy (SSO)*.

3.6.10 Brute-force protection

Powered by django-axes.

AXES_ENABLED
Default `True`.

AXES_FAILURE_LIMIT
Default `10`. Failed login attempts before lockout.

AXES_COOLOFF_TIME
Default `30` (minutes). Lockout duration.

AXES_HANDLER
Default `axes.handlers.cache.AxesCacheHandler`.

AXES_LOCKOUT_PARAMETERS
Default `ip_address`. What to lock on (IP, username, etc.).

AXES_IPWARE_PROXY_COUNT, AXES_IPWARE_META_PRECEDENCE_ORDER
Advanced settings for IP detection behind proxies. See django-axes docs.

3.6.11 reCAPTCHA

USE_RECAPTCHA
Default `False`. Show a captcha challenge during user registration.

RECAPTCHA_PUBLIC_KEY, RECAPTCHA_PRIVATE_KEY
Your keys from https://www.google.com/recaptcha/

RECAPTCHA_REQUIRED_SCORE
Default `0.75`.

3.6.12 Email

By default wger uses Django's console email backend (writes outgoing emails to stdout). To use a real SMTP server:

ENABLE_EMAIL
Default `False`. Set to `True` to enable real email delivery.

EMAIL_HOST, EMAIL_PORT
SMTP server hostname and port (commonly `587`).

EMAIL_HOST_USER, EMAIL_HOST_PASSWORD
SMTP credentials.

EMAIL_USE_TLS, EMAIL_USE_SSL
Default `True` and `False`. Pick whichever your provider uses (typically TLS for port 587, SSL for port 465).

FROM_EMAIL
Default `wger Workout Manager <wger@example.com>`. The "From:" address on outgoing emails.

DJANGO_ADMINS
Optional. `Name,email@example.com` to receive notifications about internal server errors. Requires a working email configuration.

To verify your setup, run:

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

(In Docker: `docker compose exec web python3 manage.py sendtestemail ...`)

3.6.13 Logging

LOG_LEVEL_PYTHON
Default `INFO`. Possible values: `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`.

DJANGO_DEBUG
Default `False`. **Never enable in production**, Django leaks internal details (stack traces, settings) on error pages.

3.6.14 Static files

For S3-backed static and media files, see *Storage*.

DJANGO_COLLECTSTATIC_ON_STARTUP
Default `True`. Run `collectstatic` on every container start. Set to `False` for faster startup if you call collectstatic manually.

DJANGO_CLEAR_STATIC_FIRST
Default `False`. Delete the static directory before re-collecting. Useful if stale files are causing problems after upgrades.

COMPRESS_ENABLED
Optional. Enables django-compressor to bundle CSS/JS into single files.

3.6.15 Gunicorn

WGER_USE_GUNICORN
Default `True` in production, `False` in dev (uses Django's development server).

GUNICORN_CMD_ARGS
Default `--workers 3 --threads 2 --worker-class gthread --timeout 240`. A common starting point for worker count is `(2 × $num_cores) + 1`. See https://docs.gunicorn.org/en/stable/settings.html

3.6.16 Monitoring

EXPOSE_PROMETHEUS_METRICS
Default `False`. Expose Prometheus metrics endpoints. See *Monitoring*.

3.6.17 Python-only settings

Docker users can skip this section; everything common is exposed as an env var above. If you maintain a custom settings module (see *Backend*), there are two cache settings that can only be set in Python via the `WGER_SETTINGS` dictionary:

```
WGER_SETTINGS['INGREDIENT_CACHE_TTL'] = 86400
WGER_SETTINGS['ROUTINE_CACHE_TTL'] = 86400
```

INGREDIENT_CACHE_TTL
Default `604800` (one week). How long ingredient list responses are cached.

ROUTINE_CACHE_TTL
Default `2419200` (four weeks). How long routine list responses are cached.

3.6.18 Site settings (Django sites framework)

Some wger features use Django's `contrib.sites` framework for the site domain and name. The Docker entrypoint sets these from `SITE_URL` automatically via the `set-site-url` management command. For from-source installations or to override manually, run:

```
python manage.py set-site-url https://wger.example.com
```

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

This assumes the default site ID of 1. If you use a different one, set it in your settings module:

```
SITE_ID = 2
```

3.6.19 Template customization

If you run a public instance, you might want or need to customize a couple of templates to match info and legal requirements:

- `wger/software/templates/tos.html` for your Terms of Service
- `wger/core/templates/misc/about.html` for your contact address or other legal information

With Docker, the simplest option is to mount your replacements over the files inside the container via your `docker-compose.override.yml`.

```
services:
  web:
    volumes:
      - ./my-tos.html:/home/wger/src/wger/software/templates/tos.html:ro
      - ./my-about.html:/home/wger/src/wger/core/templates/misc/about.html:ro
```

For other setups, these are standard Django templates. See Django's docs on overriding templates for the general approach.

3.7 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.).

3.7.1 Administration Commands

Use the `wger` command to perform different administration and bootstrapping tasks such as initialising the database. You can get a list of all available commands by calling `wger` without any arguments as well as help on a specific command with `wger --help <command>`.

Here are some of the most important ones:

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:

create-or-reset-admin
: Makes sure that the default administrator user exists. If you change the password, it is reset.

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.

load-online-fixtures
: Downloads a subset of ingredients and the weight units fixtures, then installs them. To download all ingredients, use the manage.py command with the `sync-ingredients-bulk` option (see below).

3.7.2 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>`.

To retrieve a full list of available commands, call `python manage.py` without any arguments and look under the app names (weight, nutrition, manager, core, exercises). To get help on a specific command, call `python manage.py <command_name> --help`.

Here are some of the most important ones:

sync-ingredients-bulk
: Options: *–set-mode=update|replace*

  Synchronizes the ingredient database from the default wger instance to the local installation. Ingredients that you added manually to the database are not touched. This command downloads a dump of the ingredient database from the default wger instance, imports the data and then deletes it. This is the recommended way to synchronize the ingredients. Setting the `SYNC_INGREDIENTS_CELERY` option uses Celery to perform the synchronization in the background in regular intervals.

sync-ingredients[-async]
: Same as `sync-ingredients-bulk`, but instead of downloading a dump, it uses the API to retrieve the ingredients. This way is much slower, but is kept for backwards compatibility. The `-async` version will use Celery to perform the synchronization in the background.

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, does not overwrite existing images.

download-exercise-videos
: synchronizes the exercise videos from the default wger instance to the local installation, does not overwrite existing videos.

import-off-products
: Imports and updates products from the Open Food Facts database. You can select whether to use a local file with the full database dump, the daily delta updates or use a mongo database, see the help for more information. Note that this command is not intended to be used on a regular basis in local installations, in order to

not put too much load on the Open Food Facts servers. To keep your database up-to-date, you can use the `sync-ingredients-bulk` management command.

`extract-i18n`
Used for development only. Extracts strings from the database that need to be translated. See the *Internationalization (i18n)* section for more information.

`dummy-generator-*`
Use to generate dummy data for the different entry types. For more information see the *Dummy data* section.

3.7.3 Celery

To list the currently scheduled tasks:

```
celery -A wger inspect active
```

To to clear the currently waiting tasks:

```
from wger.celery_configuration import app
app.control.purge()
```

For other possible commands, consult the celery documentation.

3.8 Common errors and pitfalls

3.8.1 Missing static files

If you start the application and don't see any CSS styles, images, etc., there's a problem with the static files. This happens often.

The reason for this is that for performance reasons, django itself does not serve any of the static files in production. Instead, it relies on a separate dedicated process server (in our case, the nginx service, but it could be an external CDN or similar) to do it. The process works in two steps:

- **Collect:** the django service runs a command to gather all static files into a single directory. This step happens automatically when you start the `web` service but can be manually triggered with `docker compose exec web python3 manage.py collectstatic`.
- **Serve:** the nginx service reads files from that exact same directory and serves them to the user.

These two steps need access to shared docker volume. If you change the volume configuration, django might not be able to write the file or the web server might no longer find them. This can happen very easily if you use mounted folders and the permissions aren't correctly set (make sure they are `chown`-ed to the UID and GID 1000, even if this user doesn't exist on your system, and are readable by everyone).

The solution is always to ensure that the volume or folder which holds your static files is correctly mounted by both the django and the web server services. If you want to use your own, exising, web server, you need to make sure that the files are read and served under the right URLS. Take a look at our nginx.conf to see how this can look like.

For more information, consult django's documentation.

3.8.2 Email verification links

When self-hosting, verification emails may otherwise contain links like `http://localhost/...`. Set `SITE_URL` in your environment to your public base URL (no trailing slash) so links use your domain instead:

```
SITE_URL=https://your.public.domain
```

This value is used by the application to build absolute links in outgoing emails and other places where a full URL is required.

3.8.3 CSRF errors

You will most probably run into CSRF errors when you try to use the application, specially if you configured a domain and django's CSRF protection kicks in. To solve this, update the env file and either

- manually set a list of your domain names and/or server IPs `CSRF_TRUSTED_ORIGINS=https://my.domain.example.com,https://118.999.881.119:8008` If you are unsure what origin to add here, set set `DJANGO_DEBUG` to true, restart the service and the error message will tell you exactly which one django has a problem with. Note: the port is important!
- or set the `X-Forwarded-Proto` header like in the example and set `X_FORWARDED_PROTO_HEADER_SET=True`. If you do this consult the documentation as there are some security considerations.

3.8.4 Wrong pagination links

(note that this mostly applies if you are running your own reverse proxy)

For the application to work correctly, you need to make sure that the reverse proxy is correctly configured.

If this is not the case, some features might break in subtle ways. E.g. the pagination links in the api are constructed by django from the passed headers. In this case you might be able to correctly reach and fetch some data but the "next" link might then point to "localhost", which will not work, only work while in your home network, etc.

Make sure that you forward the host header as well as the protocol header to the application. Consult the Django docs for details.

3.9 Storage

This page covers the storage choices for your wger instance: which database backend to use, and where to keep media and static files.

By default, wger uses Postgres for the database and stores media and static files on local volumes (served by the bundled nginx). Both can be swapped out.

3.9.1 Database backend

The default Postgres setup is recommended for production. For small or test installations, SQLite is also supported.

SQLite

If you want to use an existing database, copy it next to the docker compose file. Otherwise, create an empty file and make sure permissions are correct:

```
touch ./database.sqlite
chmod 664 ./database.sqlite
```

In your env file:

```
DJANGO_DB_ENGINE=django.db.backends.sqlite3
DJANGO_DB_DATABASE=/home/wger/db/database.sqlite
```

In `docker-compose.yml`, mount the SQLite file into the web and celery services, remove the dependency on the `db` service, and delete the `db` service definition entirely:

```
web:
  image: docker.io/wger/server:latest
  depends_on:
    # delete this
    db:
      condition: service_healthy
    [...]
  volumes:
    - ./database.sqlite:/home/wger/db/database.sqlite
    [...]

celery_worker:
  image: docker.io/wger/server:latest
  volumes:
    - ./database.sqlite:/home/wger/db/database.sqlite
    [...]

# remove the db service
db:
  [...]
```

3.9.2 S3 / object storage

wger supports serving static files (JS, CSS, etc.) and media files (gallery images, exercise and ingredient images, etc.) from an S3-compatible object store.

If you enable this, you don't need nginx to serve these files anymore, a plain reverse proxy in front of the application is enough. You will likely need to adjust the CORS settings on your storage provider so the application can fetch the files from the S3 domain.

Settings

USE_S3_MEDIA_FILES
bool, default: `False`

Enable S3-backed media storage.

USE_S3_STATIC_FILES
bool, default: `False`

Enable S3-backed static files storage. When enabled, set `DJANGO_COLLECTSTATIC_ON_STARTUP` to false, the collectstatic command will otherwise needlessly increase startup time.

AWS_ACCESS_KEY_ID
access key

AWS_SECRET_ACCESS_KEY
secret key

AWS_STORAGE_BUCKET_NAME
bucket name

AWS_S3_REGION_NAME
region used to build endpoints such as `eu-central-1` or `hel1`

AWS_S3_DOMAIN
base domain used to build endpoints such as `amazonaws.com`

AWS_S3_ENDPOINT_URL
string, default: `https://{AWS_S3_REGION_NAME}.{AWS_S3_DOMAIN}`

explicit endpoint, change if needed

AWS_S3_CUSTOM_DOMAIN
string, default: `{AWS_STORAGE_BUCKET_NAME}.{AWS_S3_REGION_NAME}.{AWS_S3_DOMAIN}`

custom domain, change if needed

USE_S3_URL_FOR_MEDIA
bool, default: `True`

When true, the app sets `MEDIA_URL` to `https://{AWS_S3_CUSTOM_DOMAIN}/` (otherwise the configured `MEDIA_URL` value is used). Set this to false only if you want to use a reverse proxy to make the media files appear under the same domain as the application, e.g. with nginx's `proxy_pass`.

USE_S3_URL_FOR_STATIC
bool, default: `True`

When true, the app sets `STATIC_URL` to `https://{AWS_S3_CUSTOM_DOMAIN}/` (otherwise the configured `STATIC_URL` value is used).

S3_MEDIA_FILES_LOCATION and S3_STATIC_FILES_LOCATION
string, default: `media` and `static`

Used to control the prefix for media and static files in the bucket.

AWS_QUERYSTRING_AUTH
bool, default: `False`

Controls signed URLs; currently set to `False` (public URLs).

Examples

AWS

- `AWS_STORAGE_BUCKET_NAME=my-bucket`
- `AWS_S3_REGION_NAME=eu-central-1`
- `AWS_S3_DOMAIN=amazonaws.com`
- `AWS_S3_ENDPOINT_URL=https://s3.eu-central-1.amazonaws.com`
- `AWS_S3_CUSTOM_DOMAIN=my-bucket.s3.eu-central-1.amazonaws.com`

Hetzner

- `AWS_STORAGE_BUCKET_NAME=my-bucket`
- `AWS_S3_REGION_NAME=hel1`
- `AWS_S3_DOMAIN=your-objectstorage.com`

For other providers and options, consult the django-storages documentation: https://django-storages.readthedocs.io

Note

If you already have files in a local folder, you'll need to copy them over to the bucket with a tool like `awscli` or `rclone`.

3.10 Authentication Proxy (SSO)

This feature allows you to delegate user authentication to a trusted reverse proxy server (e.g., Nginx, Caddy, Apache, Traefik) or an external authentication provider (e.g., Authelia, Authentik, Keycloak, systems using LDAP, SAML, OAuth2) fronting the application.

When enabled, if the trusted proxy successfully authenticates a user and sets a specific HTTP header containing their username, wger will automatically log that user in. If the user doesn't exist, it can optionally be created automatically.

> **Warning**
>
> Misconfiguration of this feature can lead to severe security vulnerabilities, potentially allowing attackers to impersonate any user!

The core risk is that an attacker could directly send the authentication header (e.g., `X-Remote-User: admin`) to the application, bypassing the proxy's authentication, and gain access as that user. Because of this, your reverse proxy **must** be configured to overwrite the authentication header from any incoming request before it processes the request, e.g. pseudocode for Caddy:

```
your.wger.domain.com {
    # ... authentication configuration (e.g., forward_auth to Authelia/Authentik) ...
    # Assuming auth service sets Upstream-User header upon success

    reverse_proxy http://<wger_backend_ip>:<port> {

        # Remove header
        header_up delete X-Remote-User

        # Set header based on auth service response
        header_up +X-Remote-User {http.reverse_proxy.upstream.header.Upstream-User}
    }
}
```

If you plan to use the mobile app, make sure to bypass the `api/*` subfolder from your proxy auth so that the app can access the api endpoint. In caddy, it might look something like this:

```
wger.example.com {
  import cert
  # allow api path to bypass auth (requires api key)
  handle /api/* {
    reverse_proxy wger_server:8000
  }
    # Enclosing in `route` forces execution order
    route {
        # Forward outpost path to actual outpost
        reverse_proxy /outpost.goauthentik.io/* authentik_server:9000
  ...
}
```

3.10.1 Settings

AUTH_PROXY_HEADER
Required. The name of the HTTP header that the reverse proxy sets with the authenticated user's username. Usually `Remote-User`, but can be any name you choose. Default is empty, which disables the feature.

Apply Django's header-name convention: uppercase, replace hyphens with underscores, prefix with `HTTP_`. So if the proxy sets `X-Remote-User`, configure `HTTP_X_REMOTE_USER` here.

AUTH_PROXY_TRUSTED_IPS
Required. Comma-separated list of trusted IP addresses from which the application accepts authentication headers. Usually this is the IP of your reverse proxy. If the proxy runs on the same host, include `127.0.0.1` and/or `::1`.

AUTH_PROXY_CREATE_UNKNOWN_USER
Default `False`. Whether to automatically create a new user account if the authenticated user does not exist in the database yet.

AUTH_PROXY_USER_EMAIL_HEADER
The header to read the email from for auto-created users. Ignored if `AUTH_PROXY_CREATE_UNKNOWN_USER` is not set. Same Django header-name convention as `AUTH_PROXY_HEADER`.

AUTH_PROXY_USER_NAME_HEADER
The header to read the display name from for auto-created users. Ignored if `AUTH_PROXY_CREATE_UNKNOWN_USER` is not set. Same Django header-name convention as `AUTH_PROXY_HEADER`.

3.11 Anubis (anti-AI-crawler proxy)

For public-facing instances, AI scrapers can and will flood the application with requests, resulting in high database loads. Anubis is a proof-of-work proxy that sits in front of wger and challenges suspicious clients before they reach the application. The wger.de instance runs this setup in production.

Legitimate users might see a brief "checking your browser" page on their first visit, then continue as normal. API clients (including the wger mobile app) and known search engines are not challenged.

3.11.1 Setup

The docker repo ships everything you need:

- the `anubis` service block in docker-compose.override.example.yml
- a curated rules file at config/anubis-rules.yml, mounted into the container automatically

Three steps:

1. **Generate an ED25519 private key.** Anubis uses it to sign challenge cookies, so don't reuse the example value from the override file.

```
openssl rand -hex 32
```

2. **Add a new anubis service to your docker-compose.override.yml.** Copy the block from `docker-compose.override.example.yml`, paste the key into `ED25519_PRIVATE_KEY_HEX` and set `WEBMASTER_EMAIL` to a real address.

3. **Point your reverse proxy at Anubis instead of the web service.**

   With Caddy, swap the target in your `Caddyfile`:

```
reverse_proxy anubis:3000 {
    header_up Host {host}
    # ... keep your existing header_up directives
}
```

With nginx, change the upstream in `config/nginx.conf`:

```
upstream wger {
    server anubis:3000;
}
```

Restart with `docker compose up -d`. Hitting the site in a browser should briefly show the Anubis challenge page; `/api/v2/*` stays challenge-free.

3.11.2 Rules

The bundled `anubis-rules.yml` is a curated preset that:

- always allows `/api/v2/*` (so the mobile app and other API clients are never challenged)
- allows well-known good search engine crawlers (Google, Bing, DuckDuckGo, and others)
- allows `robots.txt`, `favicon.ico` and `/.well-known` routes
- blocks pathological bots and AI scrapers (Anubis's bundled `ai-block-moderate` preset)
- applies browser-fingerprint heuristics so requests that look like a real browser get an easier challenge

Read the full rules at config/anubis-rules.yml.

Warning

The `allow-api-v2` rule sits at the top of the file for a reason. Anubis evaluates rules top to bottom and stops at the first match. If you reorder or remove this rule, the mobile app and other API clients will start getting proof-of-work challenges they can't solve, and effectively break.

3.11.3 Monitoring

Anubis exposes Prometheus metrics on its `METRICS_BIND` port (`:9090` in the example override). The docker repo ships a Grafana dashboard at `grafana/dashboards/anubis.json` that visualises traffic, challenge rate and pass/block ratio.

For general Grafana setup, see *Monitoring*.

Note

On from-source installations, Anubis runs as a separate service alongside gunicorn and your reverse proxy. The wger docs don't cover that setup; refer to the upstream Anubis project for installation instructions.

3.12 Postgres major version upgrade

Postgres does not automatically upgrade between major versions, you need to perform the upgrade manually. Since the amount of data the application generates is small, a simple dump-and-restore is the easiest approach.

If you pulled new changes from the docker-compose repository and got an error message like *"The data directory was initialized by PostgreSQL version 12, which is not compatible with this version 15"*, this page is for you.

See also: https://github.com/docker-library/postgres/issues/37

3.12.1 Procedure

```
# Check out the last version of the compose file that uses Postgres 12
git checkout pg-12

# Stop the other services
docker compose stop web nginx cache celery_worker celery_beat

# Dump the database, then remove the container and volume
docker compose exec db pg_dumpall --clean --username wger > backup.sql
docker compose stop db
docker compose down
docker volume remove docker_postgres-data

# Switch back to the current version, import the dump, and start everything
git checkout master
docker compose up db
cat backup.sql | docker compose exec -T db psql --username wger --dbname wger
docker compose exec -T db psql --username wger --dbname wger -c "ALTER USER wger WITH␣
↪PASSWORD 'wger'"
docker compose up
rm backup.sql
```

The `pg-12` tag in the docker-compose repository pins the previous Postgres version. Adjust the tag if you're upgrading from a different version.

> **Note**
>
> On a from-source installation, follow the standard Postgres upgrade procedure for your distribution (e.g., `pg_upgradecluster` on Debian/Ubuntu) or use `pg_dumpall` / `pg_restore` against your existing service.

3.13 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.

3.13.1 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.

3.13.2 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.

3.13.3 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.

3.13.4 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

Lifecycle
: Stop, start, inspect, and auto-start the application via systemd.

Updating wger
: Update wger to a new release.

Backup
: Backup and restore your wger instance.

Syncing exercises and ingredients
: Sync exercises and ingredients from upstream.

Monitoring
: Monitor the application with Grafana and Prometheus.

Settings
: Available configuration options for the application.

Commands
: Reference for the available administration and management commands.

Common errors and pitfalls
: Common errors and how to fix them.

Storage
: Database backend (Postgres, SQLite) and S3-compatible object storage for media and static files.

Authentication Proxy (SSO)
: Delegate authentication to a reverse proxy or external SSO provider.

Anubis (anti-AI-crawler proxy)
: Protect public-facing instances from AI scrapers.

Postgres major version upgrade
: Upgrade Postgres to a newer major version.

Gym administration
: Manage gyms, members, trainers and contracts.

CHAPTER

FOUR

DEVELOPMENT

For developers who want to work on wger itself. The application is split across three main repositories (backend, frontend, mobile app), plus a few supporting topics for the development workflow.

4.1 Backend dev with Docker

You can get a development environment up and running in a few minutes with docker.

Clone https://github.com/wger-project/docker as well as https://github.com/wger-project/wger and `cd` in the docker repo into the environment of your choice (`dev` or `dev-postgres`). Then copy `.env.example` to `.env` and set the path to correspond to the location where you have checked out the wger server git repo:

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

cd docker/dev
cp .env.example .env
echo "WGER_CODEPATH=/my/wger/path" > .env
```

Start docker watch in the docker folder:

```
docker compose up --watch
```

Docker watch will now automatically sync the code changes to the container and rebuild the image when needed. This solves the problem of using a file mount where the user ID of the host and the container are different and works by having docker automatically sync the files using the user defined in the docker file. When you edit the pyproject.toml or package.json files docker automatically rebuilds the image and restarts the container. When this happens you will see messages like:

- Syncing service "web" after 1 changes were detected
- Rebuilding service(s) ["web"] after changes were detected...

> **Note**
>
> Files that only exist within the container will be lost when the container is disposed! This applies to generated files like migrations or the db when it's first created. Make sure to copy them over to the host (see below).

For more information: https://docs.docker.com/compose/how-tos/file-watch/

Now you can bootstrap the application initially, download the required JS and CSS libraries and start the development server.

```
docker compose exec web /bin/bash

# this creates initial db tables, runs npm install, npm run build:css:sass, etc
wger bootstrap
```

After this you can run the following commands to load the initial data:

```
# pull exercises from wger.de (or other source you have defined)
python3 manage.py sync-exercises

# pull nutrition information
wger load-online-fixtures

# finally, this is important, start the actual server!
python3 manage.py runserver 0.0.0.0:8000
```

You can now login on http://localhost:8000 with the default administrator user (afterwards you just need to start the server, no need to bootstrap again):

- username: `admin`
- password: `adminadmin`

To copy the database to your source folder:

```
docker compose cp web:/home/wger/src/database.sqlite /my/wger/path

# at this time you can make a backup if you want such that if you mess
# something up, you don't have to start from scratch (just copy orig back)
cd /my/wger/path
cp database.sqlite database.sqlite.orig
```

You will probably want to configure your IDE to use the python interpreter in the container so you can start the server from there. Since this is very dependent on the one you are using, consult its documentation for how to do this:

- https://code.visualstudio.com/docs/containers/quickstart-python
- https://www.jetbrains.com/help/pycharm/using-docker-as-a-remote-interpreter.html

Note that while the default configuration for settings.py uses environmental variables to configure the behaviour of the application, these only take effect when you restart the container. Since during development you probably want the changes to take effect immediately, you can just set the values in settings.py, the development server will pick it up automatically and restart. For example, in settings.py you will find:

```
WGER_SETTINGS["EXERCISE_CACHE_TTL"] = env.int("EXERCISE_CACHE_TTL", 3600)
```

where the value is read from the env variable EXERCISE_CACHE_TTL. But this can be changed to just:

```
WGER_SETTINGS["EXERCISE_CACHE_TTL"] = 123
```

to take effect immediately directly.

Other useful commands:

```
# Apply new db migrations (e.g. if you get some OperationalError)
# safe to ignore: Your models in app(s): 'exercises', 'nutrition' have changes
# that are not yet reflected in a migration, and so won't be applied.
```

(continues on next page)

(continued from previous page)

```
python3 manage.py migrate

# If you edited the theme CSS files
npm run build:css:sass

# Access the db directly
python3 manage.py dbshell

# Sync exercises
python3 manage.py sync-exercises
python3 manage.py download-exercise-images
python3 manage.py download-exercise-videos

# Force a rebuild of the image (compose watch usually handles this,
# but useful e.g. after switching branches)
docker compose build web

# Prune the build cache if it has grown too large over time
docker builder prune
```

4.2 Backend

The backend is a Django application, the repository can be found at:

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

> **Tip**
>
> The easiest way to get a backend running is via the Docker dev environment, see *Backend dev with Docker*. The instructions below cover a native install on your machine.

4.2.1 Local installation

Install `npm`, `sass` and, optionally, `uv`

Download the source code:

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

If using `uv`:

```
uv sync
uv pip install -e .
source .venv/bin/activate
```

Otherwise, manually create a new virtualenv and install everything:

```
python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
```

(continues on next page)

(continued from previous page)

```
pip install --group dev .
pip install -e .
```

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

```
export DJANGO_SETTINGS_MODULE=settings.local_dev
wger bootstrap
wger load-online-fixtures
```

It's recommended to make a backup of the SQLite database after the initial bootstrap, just copy it to some other place:

```
cp database.sqlite database.sqlite.orig
```

You can of course also use other databases such as PostgreSQL or MariaDB. Create a database and user with the usual tools (`createdb`, `CREATE USER`, etc.) and point Django at it via the `DJANGO_DB_*` env vars (see *Settings*) before calling bootstrap.

Compile the translation files:

```
cd wger
django-admin compilemessages
```

After all this 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`.

4.2.2 Running the tests

The backend uses Django's built-in test runner. Each app keeps its tests under `wger/<app>/tests/`. To run the tests:

```
# Full text suite
python manage.py test

# In multi core machines
python manage.py test --parallel auto

# Only some tests
python manage.py test wger.exercises
python manage.py test wger.exercises.tests.test_categories
```

4.2.3 Code style

The project uses `ruff` for formatting and `isort` for import sorting, both configured in `pyproject.toml`. Before opening a PR:

```
ruff format
isort .
```

CI runs the same checks, so it's worth doing it locally first.

4.2.4 Next steps

- For a description of the available settings consult *Settings*.
- You might need to start *Celery* as well if you want to run certain commands in the background.
- For managing i18n files consult *Internationalization (i18n)*.
- Check the *Dummy data* for generating dummy data.
- Take a look at django extensions, a collection of custom extensions and commands:

  https://django-extensions.readthedocs.io/

  E.g. 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]
```

4.3 Frontend

Note that "frontend" is not completely correct, more like "part of the frontend". The application uses django to render the main page, react is only used for some parts of the application that need to be more interactive. These parts have their own repository which can be found at

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

4.3.1 Getting Started

Copy `.env.TEMPLATE` to `.env.development` and edit it to your needs.

You can obviously use your own instance, but feel free to use the test server (the db is reset every day):

- URL: `https://dev.wger.de`
- username: `user`
- password: `flutteruser`
- API key: `31e2ea0322c07b9df583a9b6d1e794f7139e78d4`

> **Note**
>
> If you want to run a local backend instead (useful when you're changing the backend at the same time), the quickest setup is the *Docker dev environment*.

Install node (>22) and run:

```
npm install
```

Then, in the project directory, you can run:

```
npm start
```

and open http://localhost:3000 in the browser.

4.3.2 Running the tests

The components ship with a Vitest test suite:

```
npm run test
```

4.3.3 Rendering in django

We don't render the whole page with react, just a part of it. We use `ReactView` in django to render an emtpy div with a known ID and then let react take over.

Take a look at `src/index.tsx` to see how we do this.

If you want to test the new version locally, you can use `npm link` which will create a symlink to the dev repository and allow you to instantly see the changes:

```
cd /path/to/react/repo
npm link
cd /path/to/wger/server
npm link @wger-project/react-components
```

Don't forget to unlink everything once you're done:

```
cd /path/to/wger/server
npm unlink @wger-project/react-components
cd /path/to/react/repo
npm unlink
```

4.4 Mobile App

There is a mobile app available for Android and iOS. It's a flutter application and the source code can be found at

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

1) Install flutter (the app currently uses the flutter version pinned in action.yml)

https://docs.flutter.dev/get-started/install

2) Start a wger server. You can either run your own instance (see e.g. *Backend dev with Docker*) or use the test server:

- URL: `https://dev.wger.de`
- username: `user`
- password: `flutteruser`
- API key: `31e2ea0322c07b9df583a9b6d1e794f7139e78d4`

The db is reset every day, so feel free to edit or delete anything.

3) Start the application with `flutter run` or use your IDE

Please note that depending on how you run your emulator you will need to change the IP address of the server (on iOS `http://localhost:8000` while on android you need `http://10.0.2.2:8000`).

4.4.1 Running the tests

The unit and widget tests run with:

```
flutter test
```

4.5 Celery

wger uses 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.

Celery needs a configured cache backend, redis is a good and easy solution and you might already have it running for the regular application:

```
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. These installation instructions are only for a manual setup.

4.5.1 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"
```

(continues on next page)

(continued from previous page)

```
# 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`, paste 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=${CELERYD_LOG_FILE} \
    --loglevel="${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

4.5.2 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
```

4.5.3 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
```

4.6 Internationalization (i18n)

4.6.1 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
```

4.6.2 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.

- **django:** add an entry to `LANGUAGES` in `wger/settings_global.py` and create a new empty .po file with `django-admin makemessages --extension py,html,tpl -l fr` (use the ISO 639-1 language code)
- **wger:** add the new language in the language admin page. 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 core.language --indent 4 --natural-foreign >␣
    ↪data.json
    ```

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

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

  - `languages.json` to the fixtures folder in core and optionally delete the json files:

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

4.6.3 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.

4.6.4 Server translations

There are a handful of translations that are not part of the Django translation files, but are saved in the db instead. These are things like categories and muscles. These can be extracted to be used in the Django, react and flutter applications with the following command:

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

This command will generate the following files:

- `wger/i18n.tpl`: this allows the django makemessages command to extract the strings so they can be translated.
- `wger/i18n.tsx`: this should be copied to the react repository.
- `wger/app_en.arb` and `wger/i18n.dart`: the content of this file should be copied to the end of the `intl_en.arb` file in the flutter repository.

Yes, this process is by far not optimal and should definitely be changed one of these days, however these translations are *very rarely* updated so this is kinda ok. The downside is that local instances can't easily add new categories or muscles.

4.7 Dummy data

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.

4.8 Release process

wger has three components that release independently: the Django backend, the JS/CSS frontend package, and the Flutter mobile app. The processes are similar in spirit but have their own steps.

4.8.1 Backend

Before releasing a new (major or minor) version of the backend, a few manual steps are still necessary. Some of these could and should be automated; for now they are not, which is also why there aren't that many releases.

Bump versions

Bump the app version as well as the minimum required mobile app version in:

- `wger/version.py`
- `package.json` (not strictly required, but keep it in sync)
- All the `.github/workflows/docker-*.yml` files
- `docs/conf.py` (in the docs repo)

Update the project ID in `docs/contributing.rst` (in the docs repo).

Update the contributors list

Run the script that updates the contributors list:

```
python3 extras/authors/generate_authors_api.py
```

Update the exercise fixture

It's recommended to update the exercise fixture before a release. Extract it from a current database, split the files, and copy them as appropriate:

```
python ./manage.py dumpdata --indent 4 --natural-foreign exercises > extras/scripts/data.
↪json
cd extras/scripts/
python3 filter-fixtures.py
cp categories.json ../../wger/exercises/fixtures/
```

Update translations

Update the .po files as described in *Internationalization (i18n)*.

Tag the release

Create a new tag for the release:

```
git tag -a 1.2.3 -m "Release 1.2.3"
git push origin 1.2.3
```

Create a final tag for the Docker image without the `-dev` suffix (after the images have been built):

```
docker login
docker buildx imagetools create --tag wger/server:1.2.3 wger/server:latest
```

Create a GitHub release

Create a new release on GitHub from the tag. Generate the description from the pull requests and edit as needed, then link to it from the changelog in the docs repo:

```
gh release create "X.Y" --generate-notes
```

Announce it

Write an announcement and post it on Discord, Mastodon and similar channels.

4.8.2 Frontend

Update the version in `package.json` to use the current date:

```
NEW_VERSION=$(date +%Y-%m-%d)
npm version "${NEW_VERSION}" --no-git-tag-version
```

Publish the new version to npm by manually triggering the `publish` workflow in the GitHub Actions tab.

In the Django server, update the version in `package.json` to the same version and run:

```
npm install
```

4.8.3 Mobile app

The release itself is automated with GitHub Actions and fastlane, but a few manual preparation steps are involved.

Preflight checks

1) Bump versions

Flutter: If we use a new flutter version, update the version in `.github/actions/flutter-common/action.yml` as well as `flatpak-flutter.json` in the `de.wger.flutter` repository (commit to master).

Min server version: If the new version of the app requires a new minimum server version, update the version in `MIN_SERVER_VERSION`.

2) Update cocoapods for iOS and Mac

In the `ios` and `macos` folders run `pod update` (or `gem update`). Verify that builds succeed with `flutter build macos --release` and `flutter build ios --release --no-codesign`.

3) Dry-run release before uploading

We use fastlane to automate the release process. To test that the update works, you can do a dry-run (needs the various publishing keys available):

- Increase the build number in `pubspec.yaml` (revert after the dry-run was successful): `flutter pub run cider bump build`
- `flutter build appbundle --release`
- `bundle install`

- `bundle update fastlane`
- `bundle exec fastlane android test_configuration`

It might be necessary to repeat these steps if `upload_to_play_store` returns errors such as a missing title.

If a language was added via the Weblate UI, it might be necessary to set the correct language code: https://support.google.com/googleplay/android-developer/answer/9844778?hl=en#zippy=%2Cview-list-of-available-languages

4) Flatpak / Flathub

The `de.wger.flutter` repository is a fork of the flathub metadata repo and contains the build instructions for the flatpak version of the app.

If there's a new version of sqlite (very likely), make sure the `flatpak-flutter` script supports it. A PR upstream might be needed; until then, the `de.wger.flutter.json` can be generated locally. After making any changes to `flatpak-flutter.json`:

```
git clone https://github.com/TheAppgineer/flatpak-flutter.git
git clone https://github.com/wger-project/de.wger.flutter.git
cd de.wger.flutter
../flatpak-flutter/flatpak-flutter.py --app-module wger flatpak-flutter.json

# optional, only needed if you get an error with the flatpak-builder command
sudo apt install elfutils
flatpak --user remote-add --if-not-exists flathub https://flathub.org/repo/flathub.
↪flatpakrepo

flatpak-builder --repo=repo --force-clean --sandbox --user --install --install-deps-
↪from=flathub build de.wger.flutter.json
flatpak run de.wger.flutter
```

All these steps need to happen on a Linux (virtual) machine.

5) Update screenshots

There are different screenshots for the different platforms and languages. They live in `fastlane/metadata/android/` and can be automatically generated. Consult `integration_test/README.md` in the flutter repo.

Takeoff

1) Trigger a release

Trigger the release manually on GitHub (click "run workflow", use the `x.y.z` format for the version). This sets the given version in `pubspec.yaml`, creates a tag, and bumps the build number. The workflow then builds the app for the different platforms and uploads it to the Play Store as well as to a newly created release on GitHub.

https://github.com/wger-project/flutter/actions/workflows/make-release.yml

2) Merge pull requests

- In the flathub repo: https://github.com/flathub/de.wger.flutter/compare/master...wger-project:de.wger.flutter:master
- In the fork, sync master: https://github.com/wger-project/de.wger.flutter

3) Update F-Droid

F-Droid usually auto-updates when it sees a new tag in the repository, but it might be necessary to manually update the metadata file and open a pull request:

https://gitlab.com/fdroid/fdroiddata/-/blob/master/metadata/de.wger.flutter.yml

Backend dev with Docker
: **Recommended starting point.** A Docker-based dev environment for the Django backend, no local Python install needed.

Backend
: Native install of the Django backend on your machine. Useful if you prefer to debug directly from your IDE without going through a container.

Frontend
: Set up a development environment for the web frontend.

Mobile App
: Set up a development environment for the mobile app.

Celery
: Set up Celery for running background tasks (development and bare-metal installations).

Internationalization (i18n)
: Manage and update translation files.

Dummy data
: Generate dummy data for development and testing.

Release process
: Steps for cutting a new backend release.

CHAPTER

FIVE

API

REST API reference for client integrations with the wger backend.

5.1 Using the API

The wger REST API is served under `/api/v2/`. It returns JSON by default, supports filtering, ordering and pagination, and uses standard HTTP status codes and verbs.

Public endpoints, such as the list of exercises or the ingredients, can be accessed without authentication. For user-owned objects such as routines, you need to authenticate.

For specific info on how to create routines over the API, see *Using the routine API*.

5.1.1 Interactive reference

wger ships a complete OpenAPI specification for every endpoint, generated automatically from the code. The following (interactive) viewers are exposed:

- `/api/v2/` returns a JSON listing of every endpoint (great for discovery), or renders an HTML index when opened in a browser
- `/api/v2/schema` the raw OpenAPI schema in JSON
- `/api/v2/schema/ui/` Swagger UI, lets you try requests directly in the browser
- `/api/v2/schema/redoc/` ReDoc, cleaner read-only view

5.1.2 JWT Tokens

The recommended authentication mechanism. You exchange a username/password pair for a short-lived access token (10 minutes in the Docker default) and a long-lived refresh token (120 days), then send the access token in the header with each authenticated request. The lifetimes are configurable via the `ACCESS_TOKEN_LIFETIME` and `REFRESH_TOKEN_LIFETIME` env vars (see *Settings*).

1. Get the tokens

Send your username and password to the `/api/v2/token` endpoint, you will get an `access` and a `refresh` token back:

```
result = requests.post(
    'https://wger.de/api/v2/token',
    data={'username': 'user', 'password': 'admin'}
)
access_token = result.json()['access']
refresh_token = result.json()['refresh']
```

(continues on next page)

(continued from previous page)

```
print(result.json())
>>> {'refresh': 'eyJhbGciOiJIUzI1...', 'access': 'eyJhbGciOiJIUzI...'}
```

2. Authenticate

Pass the access token in the Authorization header as `"Bearer your-token"`:

```
result = requests.get(
    'https://wger.de/api/v2/routine/',
    headers={'Authorization': f'Bearer {access_token}'}
)

print(result.json())
>>> {'count': 5, 'next': None, 'previous': None, 'results': [{'id':.....
```

Additionally, you can send the access token to the `/api/v2/token/verify` endpoint to verify it:

```
result = requests.post('https://wger.de/api/v2/token/verify', data={'token': access_
↪token})
```

3. Refresh

When the short-lived access token expires, use the longer-lived `refresh` token to obtain a new pair. Both tokens get rotated: the response contains a fresh access *and* refresh token, and the previous refresh token is blacklisted immediately. Store the new refresh token for the next cycle:

```
result = requests.post(
    'https://wger.de/api/v2/token/refresh',
    data={'refresh': refresh_token}
)

print(result.json())
>>> {'access': 'eyJhbGciOiJI...', 'refresh': 'eyJhbGciOiJI...'}
```

5.1.3 Permanent Token

You can also pass a permanent token in the header to authenticate, but this method is intended for personal scripts or one-off integrations:

```
token = 'abcdef123...'
result = requests.get(
    'https://wger.de/api/v2/routine/',
    headers={'Authorization': f'Token {token}'}
)

print(result.json())
>>> {'count': 5, 'next': None, 'previous': None, 'results': [{'id':.....
```

To generate a key, log in to the web app and open the "API key" section in your user settings.

You can also get a permanent token programmatically by sending a username and password to the `/api/v2/login/` endpoint. If the user doesn't have a token yet, a new one is generated.

5.1.4 Pagination

By default lists are paginated at 20 elements per page. Override with `?limit=<n>`. There is no hard cap on the limit, but very large pages will be slow and may put load on the server. The response JSON contains `next` and `previous` URLs for navigation, and `count` for the total number of results.

5.1.5 Rate limiting

A few sensitive endpoints are rate-limited (per IP for anonymous callers, per user for authenticated ones):

- `/api/v2/login/` and `/api/v2/token` (authentication): 10 requests/min
- `/api/v2/userprofile/` (registration): 5 requests/min
- `/api/v2/ingredient/` and `/api/v2/ingredientinfo/` (list): 120 requests/min
- `/api/v2/ingredient/<id>/` and `/api/v2/ingredientinfo/<id>/` (detail): 300 requests/min
- `/api/v2/ingredient-sync/` (bulk sync): 600 requests/min

Exceeding a limit returns HTTP 429 with a `Retry-After` header. All other endpoints are unthrottled.

5.1.6 Format negotiation

The API returns JSON by default. For clients you usually don't need to set anything explicitly. To request a specific format:

- Set the `Accept` header to `application/json`, or `application/json; indent=4` for indented output (useful when debugging).
- Or use a URL suffix: `/api/v2/<endpoint>.json` for raw JSON, or `/api/v2/<endpoint>.api` for the browsable HTML view (same as opening the endpoint in a browser).

5.1.7 Ordering

Use `?ordering=<fieldname>` to order results by that field. Combine multiple fields with commas: `?ordering=<field1>,<field2>`. Prefix a field with `-` to reverse, as in Django: `?ordering=-date`.

5.1.8 Filtering resources

Filter list endpoints by appending query parameters: `?<fieldname>=<value>`. Multiple filters are AND-joined: `?<f1>=<v1>&<f2>=<v2>`.

For boolean fields, pass `True` or `False` (case-sensitive); other values like `1`, `0` or `false` are ignored.

Specifying multiple values for the same field (e.g., category 1 *or* 2) is not currently supported.

5.2 Using the routine API

The routine data model is split across a handful of related objects and has a few non-obvious moving parts (iterations, day sequencing, progression rules). This page explains how everything fits together. It is **not** a field reference: for the full request/response schemas of every endpoint, use `/api/v2/schema/redoc/`.

A routine has a maximum duration of 120 days.

5.2.1 Object hierarchy

The routine itself is just a shell. The actual training data is split across several linked objects, each with its own endpoint:

```
Routine                          /api/v2/routine/
└── Day                          /api/v2/day/
    └── Slot                     /api/v2/slot/
        └── SlotEntry            /api/v2/slot-entry/
            ├── WeightConfig     /api/v2/weight-config/      (+ max-)
            ├── RepetitionsConfig /api/v2/repetitions-config/ (+ max-)
            ├── SetsConfig       /api/v2/sets-config/        (+ max-)
            ├── RirConfig        /api/v2/rir-config/         (+ max-)
            └── RestConfig       /api/v2/rest-config/        (+ max-)
```

The actual training history lives in two more objects, both linked back to the routine:

```
WorkoutSession                   /api/v2/workoutsession/
└── WorkoutLog                   /api/v2/workoutlog/
```

Templates are exposed read-only through `/api/v2/templates/` (your own) and `/api/v2/public-templates/` (those shared by others).

5.2.2 Iterations

An "iteration" is one complete cycle through all the days of a routine. Once the last day has been done, the next occurrence of the first day starts a new iteration. In practice this will most often be a week, but the system doesn't require that.

The iteration number is the key that ties progression rules to specific points in the routine: a `WeightConfig` with `iteration=3` and `value=60` applies starting on the third pass through the days.

5.2.3 Days and day sequencing

A routine is built from an ordered list of days (controlled by each day's `order` field). From this list, plus the routine's start and end date, wger computes a concrete date-by-date sequence.

For a routine that starts on the 1.1 with three days, the basic sequence is:

	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10
Day	Day 1	Day 2	Day 3	Day 1	Day 2	Day 3	Day 1	Day 2	Day 3	Day 1
Iteration	1	1	1	2	2	2	3	3	3	4

Two flags change this default behaviour:

need_logs_to_advance (on the day) stalls the sequence on a day until the user logs a session for it. Below, Day 3 has the flag set, the user finally logged a session on 1.8, and the sequence resumes on 1.9:

	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10
Day	Day 1	Day 2	Day 3	Day 1	Day 2	Day 3	Day 3	Day 3	Day 1	Day 2
Iteration	1	1	1	2	2	2	2	2	3	3

fit_in_week (on the routine) pads each iteration with empty placeholder days (no day, no exercises) so the routine always restarts on a fixed weekday:

	1.1 (Mon)	1.2	1.3	1.4	1.5	1.6	1.7	1.8 (Mon)	1.9	1.10
Day	Day 1	Day 2	Day 3	–	–	–	–	Day 1	Day 2	Day 3
Iteration	1	1	1	1	1	1	1	2	2	2

Rest days are regular days with `is_rest=true` and no slots attached. They occupy a position in the sequence but produce no exercises.

5.2.4 Slots and slot entries

Inside a day, exercises aren't attached directly. They go through a two-level structure:

- a **slot** is one "position" in the day: roughly one exercise (or one superset)
- a **slot entry** attaches a specific exercise to that slot

If a slot has more than one entry, it automatically becomes a superset. The `/date-sequence-gym` view interleaves the sets across the entries; they don't need to have the same number of sets. With

- Exercise 1, 4 sets
- Exercise 2, 2 sets
- Exercise 3, 3 sets

the gym-mode output is:

```
Exercise 1
Exercise 2
Exercise 3
Exercise 1
Exercise 2
Exercise 3
Exercise 1
Exercise 3
Exercise 1
```

The `repetition_rounding` and `weight_rounding` fields on a slot entry control how the computed values are rounded for display. If left empty when creating the entry, the defaults from the user profile are copied in.

5.2.5 Progression rules

The actual values for weight, repetitions, sets, RiR and rest aren't stored on the slot entry. They live in separate config objects (one endpoint per field), each with a `max-` variant for ranges like "8 to 10 reps".

Each config record applies starting at a given `iteration`. The value at iteration *n* is the stacked result of all preceding records: each record either replaces the value outright (`operation=r`) or adds / subtracts a value (`+` / `-`), either as an absolute number (`step=abs`) or a percentage (`step=percent`). The `repeat` flag keeps a rule active for every following iteration until another rule takes over, so "+1 kg every week" is a single record.

A weight config example:

Iteration	1	2	3	4	5	6	7	8
Config	50kg	–	–	+10%	–	+2kg	+1kg	45kg
Result	50kg	50kg	50kg	55kg	55kg	57kg	58kg	45kg

All configs are optional. If none is set for a field, that field will be `null` in the computed output, except for sets, which defaults to 1.

Requirements

A progression rule can be made conditional on the user actually hitting the prescribed values in the previous iteration. Set the `requirements` field to a JSON object of the form:

```
{
    "rules": [
        "weight",
        "repetitions",
        "rir",
        "rest"
    ]
}
```

If the `rules` list is non-empty, the rule only applies when *all* listed fields were met in at least one log of the previous iteration. For example: if the target is to move from 8x60 kg to 8x65 kg, with `rules` containing `weight` and `repetitions`, the increase only happens once the user has actually logged 8 reps at 60 kg. Until then the field stays at the previous value.

Custom calculation logic

If the rule-based system is not flexible enough, a slot entry's `class_name` field can point to a Python class under `wger.manager.config_calculations` that takes over all calculations for that entry. This is an escape hatch and not currently used; please get in touch if you have a use case for it.

5.2.6 Sessions and logs

The training history is stored in two objects. A **WorkoutSession** represents one workout (one date, optional notes and impression). A **WorkoutLog** is one performed set, attached to a session and optionally back-linked to a routine, day, slot entry and iteration.

Each log carries both what was actually performed (`weight`, `repetitions`, `rir`, `rest`) and the originally prescribed values (`weight_target`, `repetitions_target`, `rir_target`, `rest_target`), so a log preserves both sides even when the routine later changes.

5.2.7 Computed endpoints

Once a routine is set up, these read-only sub-resources return calculated views of it:

`/api/v2/routine/{id}/structure/`
: The full nested structure (routine → days → slots → slot entries), suitable for an editor.

`/api/v2/routine/{id}/date-sequence-display/`
: One entry per date, with the matching day and slots already resolved. Slots that contain repeated sets of the same exercise are folded together for display.

`/api/v2/routine/{id}/date-sequence-gym/`
: Same data, but the slots are split into individual sets and supersets are interleaved as described above. Use this for the gym-mode view.

`/api/v2/routine/{id}/logs/`
: The workout sessions and logs for this routine, grouped by session.

`/api/v2/routine/{id}/stats/`
: Aggregated statistics from the logs: volume, set count and average intensity (estimated 1RM via the Brzycki

formula), each broken down by **day**, **ISO week**, **iteration** and the **whole routine**, and further split into total, upper/lower body, per muscle and per exercise.

All five endpoints are cached server-side and invalidated when the underlying routine changes.

Using the API
General API documentation, authentication, and an overview of the available endpoints.

Using the routine API
Data model used for the flexible routines and how to create and read them via the API.

CHAPTER

SIX

CONTRIBUTING

First off, thanks for taking the time to contribute!

Our goal is to build an awesome and flexible fitness and nutrition manager, along with a comprehensive list of exercises and ingredients, all released under a free license.

All types of contributions are encouraged. And if you like the project but just don't have time to contribute, that's fine. There are other easy ways to support the project and show your appreciation, which we would also be very happy about:

- Talk about it on social media, at local meetups, or tell your friends/gym bros
- Consider *supporting us with a donation*
- Star the project on GitHub

6.1 Code

Obviously, you can also contribute code. Before starting working on a new feature, please open an issue to discuss it with us. This is important to avoid duplicating work and to make sure that your contribution is in line with the project's goals. A good starting point could be one of the issues marked with "good first issue" or the roadmap for the next release:

> https://github.com/orgs/wger-project/projects/6

This application has three main repositories, each with its own purpose (and quirks). The *Development* section covers how to set up each:

- *Backend*: Django backend
- *Frontend*: React components
- *Mobile App*: Flutter mobile app

In any case you should have a basic grasp of git and GitHub, as well as how to create pull requests. If you are not familiar with these concepts, please consult one of the many online resources available, such as

- https://docs.github.com/en/pull-requests
- https://docs.github.com/en/get-started/git-basics

Make sure to always use a feature branch, even if the change is minor.

Once you have the code ready:

- make sure to write good commit messages. A good commit message should explain what the change is about and why it was made.
- make sure the tests pass before opening the PR. CI will catch failures, but it's faster to know locally, see the test command on the dev page for the component you touched (*backend*, *frontend*, *mobile*).

- if you write new code, write new tests. These don't need to test absolutely everything, but they should cover the most important parts of the code. If you are not sure what or how to test, just ask us.
- run the formatter for the language you're touching. For Python that's `ruff format && isort .` with a line length of 100 characters (see *Backend*). Frontend (React) and mobile (Flutter) have their own formatters configured in their respective repos.
- think about UI/UX. If you are adding a new feature, make sure it is easy to use and understand. Nobody here is a designer, but we try our best!
- finally open a new PR. You can expect a response from a maintainer within a week, if you haven't heard anything by then, ping the thread.
- don't mix different features in the same pull request. If you have multiple changes, just create a separate pull request for each one.

Also, these are mostly guidelines, not rules. As everywhere in life, use your best judgment, and feel free to propose changes to this document in a pull request.

Is this the first time you contribute to an open source project? No problem! Feel free to ping us if you need help setting everything up, it can be very overwhelming at first. We are happy to help you get started.

What about AI?

We are open to contributions that use AI, but we ask you to read and understand the code that was produced. It's very tempting to let the agent do its thing and just accept everything, resist this! Specially when attempting to do bigger features, it's important to split the work into smaller chunks.

6.2 Exercises

You can contribute new exercises, images or videos, and edit or translate the existing ones. These contributions are just as important as code contributions, as they help improve the overall quality and usability of the application. Please use the search before to make sure your exercise doesn't already exist.

Note that your account must be at least 3 weeks old and have a verified email.

6.3 Translations

You can help translate the application online using Weblate and help make the application more accessible. To start just visit

> https://hosted.weblate.org/engage/wger

A list of everyone who has contributed translations so far is on the translators page.

6.4 Support the Project

This project is free and open-source, but running it isn't! Your support helps keep the server running, funds new development, and improves the overall experience for everyone. If you enjoy using this app, please consider making a small contribution. Every bit helps!

> https://www.buymeacoffee.com/wger

Thank you for supporting open-source fitness & nutrition tools!

CHAPTER

SEVEN

CHANGELOG

7.1 2.5

2026-04-16

See https://github.com/wger-project/wger/releases/tag/2.5

7.2 2.4

2026-01-18

See https://github.com/wger-project/wger/releases/tag/2.4

7.3 2.3

2025-04-05

7.3.1 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`

7.3.2 Features

- Axes behind reverse proxy by **:github:user:`@bbkz <bbkz>`** in **:github:pull:`1521`**
- Fix wrong default setting in docker image by **:github:user:`@bbkz <bbkz>`** in **:github:pull:`1531`**
- Add sync method for ingredients by **:github:user:`@rolandgeider <rolandgeider>`** in **:github:pull:`1546`**
- Add progress bar to load-online-fixtures by **:github:user:`@ebwinters <ebwinters>`** in **:github:pull:`1561`**
- Move to pyproject.toml by **:github:user:`@rolandgeider <rolandgeider>`** in **:github:pull:`1565`**
- Add full text search by **:github:user:`@rolandgeider <rolandgeider>`** in **:github:pull:`1594`**
- put building/obtaining image instructions before trying to run it. by **:github:user:`@Dieterbe <Dieterbe>`** in **:github:pull:`1655`**
- add support for fiber goal by **:github:user:`@Dieterbe <Dieterbe>`** in **:github:pull:`1664`**

- Add myself to the Polish translators by :github:user:`@Maniues <Maniues>` in :github:pull:`1661`
- fix calendar accordion by :github:user:`@bbkz <bbkz>` in :github:pull:`1681`
- Refactor product/ingredient import by :github:user:`@rolandgeider <rolandgeider>` in :github:pull:`1666`
- Export prometheus metrics by :github:user:`@rolandgeider <rolandgeider>` in :github:pull:`1685`
- Fixed a typo by :github:user:`@JLaField <JLaField>` in :github:pull:`1695`
- Allow deactivating the language filter when searching for ingredients and exercises by :github:user:`@rolandgeider <rolandgeider>` in :github:pull:`1687`
- docker image: support loading exercise videos, fixtures, nutrition info by :github:user:`@Dieterbe <Dieterbe>` in :github:pull:`1746`
- Allow env specification of Redis connection SSL parameters by :github:user:`@taylor-fuller <taylor-fuller>` in :github:pull:`1751`
- Improve docker image by :github:user:`@rolandgeider <rolandgeider>` in :github:pull:`1786`
- give dummy meals names by :github:user:`@Dieterbe <Dieterbe>` in :github:pull:`1795`
- Update django.po by :github:user:`@victorbmlabs <victorbmlabs>` in :github:pull:`1798`
- Dynamically update WeightUnit from user preferences when creating workout by :github:user:`@kmoy1 <kmoy1>` in :github:pull:`1807`
- Rework the calendar page by :github:user:`@rolandgeider <rolandgeider>` in :github:pull:`1824`
- make __main__ use pyinvoke `Program` as entrypoint by :github:user:`@eyJhb <eyJhb>` in :github:pull:`1833`
- Fix incorrectly placed <h1> tag and replace it with an <h4> tag for the "Members List" DataTable. by :github:user:`@navyjosh <navyjosh>` in :github:pull:`1843`
- Fix/registration by :github:user:`@Maralai <Maralai>` in :github:pull:`1855`
- fixes #1278 by :github:user:`@blsouthcott <blsouthcott>` in :github:pull:`1365`
- Add Management Command for Async Ingredient Synchronization by :github:user:`@crypto-a <crypto-a>` in :github:pull:`1876`
- Add language filter to sync-ingredients management command. by :github:user:`@navyjosh <navyjosh>` in :github:pull:`1875`
- Fixes for adding language filter to sync-ingredients management command. by :github:user:`@scrapcode <scrapcode>` in :github:pull:`1894`
- Clean apt temporary files in the base Docker image by :github:user:`@PeterDaveHello <PeterDaveHello>` in :github:pull:`1906`
- Flexible routines by :github:user:`@rolandgeider <rolandgeider>` in :github:pull:`1827`

7.4 2.2

2023-11-06

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

7.4.1 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

7.4.2 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

7.4.3 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

7.4.4 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

7.5 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

7.6 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),

7.7 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

7.8 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

7.9 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

7.10 1.6.1

2015-07-25

Bugfix release

7.11 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

7.12 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

7.13 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.)

7.14 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

7.15 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

7.16 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

7.17 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

7.18 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

7.19 1.0.2

2012-11-02

Bugfix release

7.20 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

7.21 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

EIGHT

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).

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

CHAPTER

NINE

SOURCES

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

See contributors and translators for everyone who has helped build the project.

CHAPTER

TEN

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.