GemsTracker procedures

and best practices for maintenance

# Summary

This document describes how we use different servers for testing and production and the process by which new code is tested and deployed on GemsTracker systems.

In short the code migrates from **development** to  **testing** to **acceptance** andin the end to **the production** server, using different git-branches of the code.

# Server setup

A standard GemsTracker project uses four different servers:

1. **Production**: the main server
2. **Acceptance**: a copy of the production server with real patient data
3. **Testing**: a copy of the software (usually on carefacts.nl) with a dummy database
4. **Development**: a local copy of the software with a dummy database

The software itself is stored spread over several projects hosted on Github.com.

# The code development and rollout procedures

A GemsTracker project consists of a private Github project that uses several other Github library projects (both private and public). The code in these projects is saved in several branches.

* The *main* or *master* branch usually contains the latest production version.
* When maintaining multiple production branches we add branches named according to version number: *p1.8.9*, *p1.9.0*, *p1.9.1*.
* The *development* branch contains the latest changes not committed to production.
* **Note:** on some older projects we use *master* / *production* branches as development branch and production branch instead. We expect to move these to a *main* / *development* branch model in the near future.
* When releasing a new version, that version is tagged, e.g. *r1.9.1p1* for the first patched version of *p1.9.1* or in the case of a main project a tag like *r1.9.1-20230505* for a 1.9.1 release on the 5th of May 2023.

Older versions of the production branches of the main library are maintained as long as there is a need, but patches will only consist of bug- and security-fixes. For the oldest versions we only add security fixes.

E.g. we are currently developing version 2.x (in a separate branch), while new functionally is still added to production version 1.9.2, bug fixes and security are maintained for 1.9.1 and only security fixes are added to 1.9.0.

Using these branches, the software test & release cycle works as follows:

1. Programming is done and first tested at the **development** server.
2. When the code runs on **development** the code is committed to a git *development* branch and moved to the **test** server for functional test by the end users.
3. When the changed have to be implemented on **production**, the first step is to merge the *development* branch into the *main* branch of the Github project code and to tag a release version.
4. The tagged version is implemented on the **acceptance** server for testing with production data.
5. When the new version is accepted on the **acceptance** server a date is planned for release on the **production** server.
6. Depending on the number of changes the production installation may be put into **Maintenance mode** for the duration of the update.
7. Usually we also make a MySQL backup on **production**, especially when there are changes to the database structure.
8. Then we update the **production** code to the current tag and run *composer update* (the library manager we use).
9. Optionally we then run database and code patches and disable the **Maintenance mode**. The code is then ready for further production use.

# The patch and update system

Sometimes upgrades entail changes to the database or updates to e.g. encryption code that requires PHP code to be executed during an update. The changes to the database are prepared and tested in advance (and are stored in patches.sql files).

Most updates contain at most some database patches or new tables, but some require the execution of PHP code. Both are therefore added to the **upgrade system** that can be executed from the command line or from the browser and that executes the requested changes in the order specified by the programmer.