# Development B

# Continuous Delivery

# Assignment C: Release-documentation output-subsystem

By Robin Bakker en Robert Kraaijeveld, INF2C

Table of contents

[Development B 1](#_Toc440623789)

[Continuous Delivery 1](#_Toc440623790)

[Assignment C: Release-documentation output-subsystem 1](#_Toc440623791)

[Stakeholders 3](#_Toc440623792)

[Asset and configuration management 4](#_Toc440623793)

[Deployment steps 6](#_Toc440623794)

[Tech usage 6](#_Toc440623795)

[(Jenkins) Deployment pipeline steps 6](#_Toc440623796)

[1. Build actions: 6](#_Toc440623797)

[2. Post-build actions: 7](#_Toc440623798)

[3. Release: 7](#_Toc440623799)

[4. Deployment: 8](#_Toc440623800)

[Release plan-timeline 9](#_Toc440623801)

# Stakeholders

Within this project, the following key stakeholders can be identified:

- The CityGis company: CityGis has a stake in this project because (within the ‘scenario’) they commissioned and provide the resources for this project. They want their data to be presented to the users and other stakeholders in a way that will reflect the quality of their work.

- The Car maintainer: The vehicles that use CityGis-equipment are maintained by a maintainer. This company/person would obviously like to know which cars have defects or exhibit strange behavior as soon as possible, since vehicles for emergency services are very scarce.

- The data-analyst/Commercial-user: The data-analyst or commercial user, from here on referred to as ‘the user’ wants to have the clearest and most complete (over)view of the CityGis data as possible to get his/hers moneys’ worth.

# Asset and configuration management

The Project56 output-subsystem has been programmed in PHP5, combined with the usage of the Laravel-framework. Below you will find all the assets, dependencies and libraries being used within our output subsystem together with their version numbers.

**Software: dependencies necessary for running and/or building.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Version number** | **Short description** | **Preferred download method** |
| PHP programming language | 5.6.16 | PHP server-side programming language. | Archive download (<http://php.net/downloads.php>) or Aptitude install on Linux. |
| Postgres | 9.3.10 | SQL-style DBMS. | Aptitude install on Linux. |
| Nginx | 1.4.6 | Lightweight webhost. | Aptitude install on Linux. |
| Git | 1.9.1 | Repository manager | Aptitude install on Linux. |
| Composer | 1.0-dev | PHP package manager. | Curl-install script. (https://getcomposer.org/download) |
| The Laravel Framework | 5.1.26 | MVC-style framework for PHP. | Composer:  ‘composer global require "laravel/installer=~1.1" ‘ |
| Phpunit\* | 5.1.3 | PHP unit-testing framework. | Composer:  ‘composer global require "phpunit/phpunit=5.1.\*" ’ |
| Apache Ant | 1.9.3 | Building engine. | Archive download (<https://ant.apache.org/bindownload.cgi>) or Aptitude install on Linux. |
| PHPDox\* | 0.8.0 | PHP XML/HTML  Documentation generator. | Phar file: ‘https://github.com/theseer/phpdox ‘ |
| PHPLoc\* | 2.5.1 | PHP static code analysis interface. | Composer:  ‘composer global require 'phploc/phploc=\*' ‘ |
| PHPMailer | 5.2 | PHP mailing library. | Composer  ‘composer require phpmailer/phpmailer’ |

\*= These dependencies should only be downloaded within the Jenkins shell script mentioned later.

**Hardware: hardware requirements recommended for running the system.**

|  |  |  |  |
| --- | --- | --- | --- |
| **CPU** | **Memory** | **Hard disk-space** | **Operating System** |
| Intel Xeon e5-2650 @2.00 Ghz per core | 3953 Megabytes | 15 Gigabytes free without OS | Ubuntu 14.04.3 |

Note that these hardware requirements are the hardware requirements that we (I.E: Team Cooperatio) use; The system could also be run with less RAM and about +- 10 gigabytes hard disk-space excluding the OS. A Linux OS is not mandatory for the output-subsystem, but it is required to run the Input- and processing-subsystems.

# Deployment steps

# Tech usage

In order to continuously build, test and deploy our subsystem we use the Jenkins Continuous Integration system. Jenkins runs locally on our Ubuntu server and is accessible from our servers’ IP-address. Jenkins builds and tests our source files and also creates (or rather, displays already created) HTML reports from PHPDox, one of the dependencies mentioned in the previous chapter. More on that subject later.

# (Jenkins) Deployment pipeline steps

Note: In Jenkins, we make the distinction between Build actions and Post-build actions. Build actions are any kind of actions that create output (usually in the form of usable software + reports) and Post-build actions use that output. A build can be triggered by a certain time-interval or by a git push.

# 1. Build actions:

**-** Within Jenkins, run a shell script consisting of the following commands:

cd /

cd /home/username/preferredProjectLocation/

sudo ./Buildscript.sh

This shell script (Buildscript.sh) should:

**Repo initialization:**

- Initialize a new local Git repository in the current directory and clone the Project56 git repository if it is not already present. If the repo is already present, the script should only try to pull any changes to the repository using git pull.

**Dependency downloading:**

- Proceed to download several PHP-phar archives (containing the dependencies marked with an asterisk in the last chapter.) and unpack them in the usr/bin/ directory, so these dependencies can be used directly via their name.

**Building through ant; analyzing, testing and reporting:**

- Proceed to call ant on the build.xml file located in preferredProjectLocation/Project56/http/Laravel. This build file is too long to fully disclose in this document, but it should do the following things:

Run PHPLoc on all the PHP files within project56/Project56/http/public. PHPLoc produces human readable output that lists a static code analysis of all the PHP source code it finds. PHPLoc lists mostly simple statistics like the amount of methods and classes etc.

Run PHPUnit; PHPUnit runs 2 simple unit-tests contained in the project56/Project56/http/Laravel/tests directory. The result of these tests is displayed in the console output. Note that the build will not pass unless all tests pass. More unit-tests may be added freely.

Run PHPDox; PHPDox saves the contents and structure of all the source files in preferredProjectLocation /Project56/http/Laravel/public and saves the output in an index.xhtml file in the preferredProjectLocation /Project56/http Laravel/build/api/html directory.

# 2. Post-build actions:

**Publishing reports:**

- After the build steps above are complete, publish the index.xhtml file that PHPDox generated. Jenkins has an HTML-report plugin for this, which takes a path and a file as input. The provided file will be displayed when you click on the “HTML Report” button at each build-attempt.

# 3. Release:

**Preparing server:**

**-** Nginx should be configured in order to actually deploy the output website and make it publicly visible from our servers’ IP-address. An nginx.conf file containing the configuration for the server should be created and should overwrite the standard nginx.conf file in the nginx installation directory.

*Note that you could potentially use a different webhosting-service than Nginx, such as Apache. However, we found that nginx is much more lightweight than heavier webhosting-services like Apache and also works better with Laravel in some regards. The choice is yours.*

- PostgreSQL should be properly configured. This includes creating a user (Linux only) to contain the Postgres processes, creating a Database for the output-subsystem to use and filling this database with tables. The Artisan system, included in the Laravel framework enables “Migration”; migration basically means inserting a table-structure-snapshot into a database. You can use this feature to have your tables created much quicker than they could ever be created by hand. The migration snapshots are contained within the Laravel/migrate/ directory of the subsystem.

# 4. Deployment:

**Making files publicly visible:**

After the Jenkins build, post-build and server preparation steps are complete, the (sub) system should be made public. To make sure that deployment goes smoothly and all dependencies are up-to-date, the following steps should be taken:

*Note that for this part of the CI-pipeline, having a Linux OS makes everything significantly easier. The var/www/ directory is always connected to the systems’ webhost, so no advanced configuration is needed. Windows user should think about hosting the output-subsystem on an (external) Linux server.*

- All the files under the preferredProjectLocation/Project56/http/Laravel/ directory should be copied to the /var/www/Laravel/ directory.

**Updating dependencies and starting background processes:**

- In the /var/www/Laravel/ directory, composer update should be run. This command ensures that all dependencies that are managed by composer in this directory are updated.

- The Postgres and Nginx processes should be manually started if they haven’t started running already.

- For all the output-features to run correctly, the input- and processing-subsystem should also be properly installed and configured. The details for installing and configuring in these subsystems are described in a different document.

# Release plan-timeline

# Week 2-6: Research and design

In the first weeks of the project we will focus on trying to get a better understanding of our stakeholders’ wishes. As mentioned earlier, each stakeholders has a different vision for this product; this fact must be incorporated into our research. Week 2 to 6 are also going to be used to create initial visual designs and implement some simple functionality into our output-subsystem. UX-research will also take place, since the output-subsystem is the main visual “hook” for potential CityGis customers.

TODO: UITGEBREID RELEASE PLAN PER WEEK

ROLVERDELING IN PROJECT