RELEASE DOCUMENTATIE

Strategies of the release of the software



Group name: SDEV

Oscar Veldman 0879062  
Wesley de Roode 0896655   
Sjors Sparreboom 0890040  
Mert Öztürk 0900994

Class: INF2C  
Date: 24-01-16

# Intro

With the help of this document, we will explain how our product went through his various stages until deployment. It will contain information of the product and both its components , the environment configuration for both and the deployment pipeline. The application itself will be used by members of the CityGis company. The end user will able to upload csv files and go look up various information offered by the application.

After reading the document, we hope the reader will have a brief understanding of how our project went from source code to product.

Index

[1. Intro 2](#_Toc441343869)

[2. Stakeholders 4](#_Toc441343870)

[3. Source code turns into project 5](#_Toc441343871)

[4. Environment configuration 6](#_Toc441343872)

[4.1 Environment upgrade 6](#_Toc441343873)

[4.2 Data migration 6](#_Toc441343874)

[4.3 Testing phase 6](#_Toc441343875)

[5. Deployment pipeline 7](#_Toc441343876)

[5.1.1 Code quality assurance tools 7](#_Toc441343877)

[5.1.2 PHP Lines of Code 7](#_Toc441343878)

[5.1.3 PHP CodeSniffer 7](#_Toc441343879)

[5.1.4 Dead Code & Copy Paste Detector 7](#_Toc441343880)

[5.1.5 PHP Unit & Google Test 7](#_Toc441343881)

[5.1.6 CPPCheck 7](#_Toc441343882)

[5.2 Implementation of the deployment pipeline 8](#_Toc441343883)

[5.2.1Laravel/PHP 8](#_Toc441343884)

[5.2.2 C++ 8](#_Toc441343885)

[7. Smoke test 9](#_Toc441343886)

[8. Monitoring 9](#_Toc441343887)

[8.1 Release Problems 9](#_Toc441343888)

# Stakeholders

There are multiple stakeholders involved by the application. The stakeholders of the application are:

* Mohamed Abbadi (Project Owner)
* Citygis
* Costumers of the application
* The admins of the application
* Owners of the vehicles
* Hosting company where runs the application

# Source code turns into project

The application can be separated in two parts. One handles the backend and the other handles the frontend. The back end is programmed in C++ and the front end is programmed in combination of PHP, Javascript and HTML. They also work together by accessing the database they are connected with. For the database we use PostgreSQL with the use of ORM.

**Front end**

* Homestead vagrant box
* Laravel, Vieuxjs and Angularjs framework
* Composer, npm, Nginx and Node (with PM2, Bower, Grunt and Gulp)

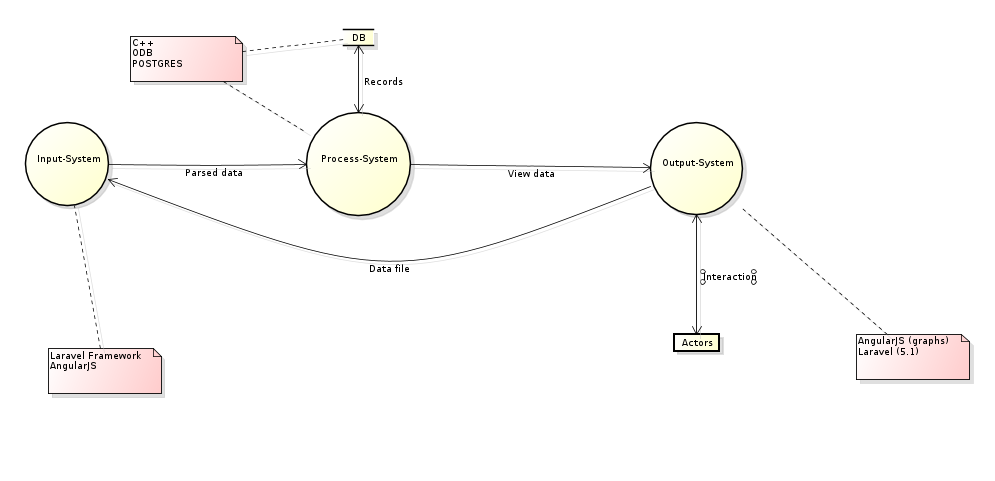
**Back end**

* Custom vagrant box
* Entity framework
* NuGet packet manager

The application can be split up in three components:

1. **Input**The input component of the system is where the user of the application can import files in the system. The user can also change the profile settings and change GUI options to its own likings.
2. **Processing**The processing component puts the files that are put in through the input component in the database connected to the system. The processing component also has several functions to calculate statistic’s.
3. **Output**The output component turns values of the input system into an interface. This component is responsible for representing the view of the application for the user.

The picture below will show how the component are connected with each other.



# Environment configuration

For the environment configuration we will use two Linux servers. One will be used for the processing part and the other will be used for the database. The front end will run on a different server with PHP installed and will be put into production with a deployment script. This deployment script will keep the code up to date of their own repositories and connects it with the necessary dependencies installed. The data transfer will be done through a migration

## 4.1 Environment upgrade

When there’s an update ready for the environment, it shall be tested before putting it online. The new environments will be put online through Docker, the rest of the installation of the application will continue with the deployment script. The next phase consists of testing the update. These tests contain acceptance, capacity, integration and user acceptance testing.

If you need to upgrade the server itself without down-time. You need a other server with the same configuration. It called a mirror server. So if you go to the server, it sends you automatic to the mirror server. Then the new server is up. You put down the mirror server. On this way you have no down-time.

Sometimes the application a software update. These software is first tested on a testing server. Then you know for sure that the software works. And with the test you have a mirror server for the update. Most of the time a update for the library doesn’t care for the application. Once it does, we must fix it.

## 4.2 Data migration

Data migrations are done with the help of migration tools in Laravel. With the use of pre-programmed schema builders, there will be a new database created and filled with migration seeders. This will also seed the structure of the database. This will result in easy adjustments in the development phase without have impact on the data of the system.

## 4.3 Testing phase

The testing phase ,after an environment update, will contain testing tools out the commit stage and tests of the deployment phase. The first part is a double check for the code quality and will also contain unit tests that are a part of the integration test. After these tests, the tools will run tests for the capacity and integration. User acceptance tests will only be done when there’s a modification to the interface or when it has influence on the behavior of the system like loading times. The test will be explained in the next chapter.

# Deployment pipeline

In the development stage code will be developed and / or tests will be written. This stage will occur locally and will be pushed to its own repository(Git). when there has been an update in code that impacts the performance of the application.

### 5.1.1 Code quality assurance tools

To keep the transaction from source code to product easy and qualitative we make use of a so called ‘quality assurance check’. This check contains of using the following tools:

### 5.1.2 PHP Lines of Code

Phploc is a tool where the code gets analyse based on certain rules, classes, interface classes and variables. This generates information of the scope, comments and visibility of the code. With this information you can check for useless comments, comments that are unusually large or when there’s no comment available. It also checks if the code is object oriented and if the methods aren’t too long.

### 5.1.3 PHP CodeSniffer

With PHP Codesniffer the code gets checked based on coding standards like Zend. With this you can backfire problems like methods that are too long and long lists of parameters. It also checks the JavaScript and CSS code if that is applicable on your project.

### 5.1.4 Dead Code & Copy Paste Detector

The name of the tool pretty much gives away what it does. When you continuously write code for hours and hours we tend to copy paste code to develop faster instead of better solutions. To backfire this problem this tool will help us find the copy pasted code and easier refactoring.

### 5.1.5 PHP Unit & Google Test

Unit tests are used to test hard and potential risky pieces of code. The goal of using unit tests is testing independent pieces of code based on input and output. The tests are based on test cases that are written beforehand by the developer.

**PHP Unit** is a unit testing framework for PHP. It is an instance of the **xUnit** architecture for unit testing frameworks.

**Google Test** is a unit testing library for the C++ programming language, based on the **xUnit** architecture.

With the use of Google Test we will fire unit tests on the C++ code written for the process.

Both of the unit testing frameworks are based on the **xUnit** architecture, which will result in better test cases for the code.

### 5.1.6 CPPCheck

CPPCheck is a static analysis tool for C++. Which means that the analysis is performed without actually executing programs. In our case it will be done on the version of the source code that is been pushed on git. Cppcheck only detects the types of bugs that the compilers normally fail to detect. The goal is to detect only real errors in the code. This tool performs checks on out of bounds, memory leaks, uninitialized variables. It also detects possible null pointer deference’s and warns you for obsolete functions and redundant code.

## Implementation of the deployment pipeline

You find the step to install the application down on this page.

### 5.2.1Laravel/PHP

1. You start the Linux server
2. On the Linux server you start Docker
3. Install PostgreSQL 9.4
4. Install composer, DO: apt-get install composer
5. You get the code from Git: <https://github.com/SDEV56/INFPRJ0156>
6. DO: composer install (It is for library’s for the application)

### 5.2.2 C++

1. You start it on Linux server.
2. On the Linux server you start Docker
3. Install PostgreSQL 9.4
4. On Docker you get the code from Git: <https://github.com/SDEV56/INFPRJ0156>
5. In the repository you see a make file
6. With the make file, you can compile the Linux C++ code
7. Start the filewatcher.out

# Smoke test

# Monitoring

For the monitoring we are going to use “New Relic”. It’s check for PHP in the front end and C++ in the back-end. If it’s something wrong in the performance. Then in some cases it’s take it away and report it in a log file. Also you can check what the performance is from the old application and from the application with a update. If it change badly, then we can fix it. All the test result from every commit will be in a log file on the server.

## Release Problems

If there is any problems with the new version of the application. Then you can start easy an older server with the older version. There are no other steps in this. The failures will be saved in a log file. So we can look in the further what is wrong with the application.