

---

# AN INTERACTIVE VISUALIZATION TOOL FOR POPULATION SIMULATIONS

---

Master Thesis

Sarah Sophie Lafaire

JUNE 8, 2018

AALBORG UNIVERSITY COPENHAGEN

Master of Science (MSc) in Technology, Surveying, Planning and Land Management / Geoinformatics



**AALBORG UNIVERSITY**  
COPENHAGEN

**Study program and semester:**

Geoinformatic, 4th semester

Aalborg University Copenhagen  
A.C. Meyers Vænge 15  
2450 Copenhagen SV

**Project title:**

An interactive visualization tool for population simulations

Secretary: Janni Rise Frellsen  
Tel: (+45) 99 40 25 35  
Email: jrl@plan.aau.dk

**Project period:**

19.01.2018-08.06.2018

**Semester topic:**

Master thesis

**Supervisor:**

Carsten Keßler

**Attendees:**

Sarah Sophie Lafaïre

**Number of pages:** 64

**Number of appendix:** 11

**Abstract:**

As the urbanization of the world is increasing, it is important to know, where people are exactly going to live. Simulations can help gaining knowledge about which cities are going to increase and which might even vanish. These simulations are very large in size, which makes it hard to process. This thesis found different solutions to develop an interactive visualization tool, that is capable of large datasets. Two Python libraries were tested, and two web servers were compared. Additionally, visualization conventions were researched and applied. It was found that the Python libraries were not suited for these very large datasets. Therefore, the two server solutions were compared on a usability level and on a performance level. The comparison showed that GeoServer is a user-friendly server but slower in the presentation of rasters than MapServer.

## Contents

<b>List of Code .....</b>	<b>V</b>
<b>List of Figures .....</b>	<b>V</b>
<b>List of Tables .....</b>	<b>VI</b>
<b>Abbreviations.....</b>	<b>VI</b>
<b>1 Introduction.....</b>	<b>1</b>
1.1 Problem statement.....	1
1.2 Research question .....	2
1.3 Report structure .....	2
<b>2 Background.....</b>	<b>3</b>
2.1 FLOW and population simulations .....	3
2.2 Visualization conventions.....	4
2.2.1 Data visualization and Cartography .....	4
2.2.2 Conventions.....	6
2.2.3 Visualization processes.....	7
2.2.4 Color .....	9
<b>3 Tools and Related work.....</b>	<b>10</b>
3.1 Python .....	11
3.1.1 NumPy .....	11
3.1.2 GDAL.....	11
3.2 Bokeh.....	12
3.3 Folium .....	12
3.4 QGIS and QGIS Standalone app.....	13
3.5 Server.....	13
3.5.1 GeoServer .....	14
3.5.2 MapServer .....	15

---

3.6	OpenLayers.....	16
3.7	Comparison tools.....	16
<b>4</b>	<b>Development of the tool .....</b>	<b>16</b>
4.1	Reference version.....	16
4.2	First trial versions .....	17
4.2.1	Folium.....	18
4.2.2	Bokeh.....	19
4.2.3	Tiles.....	20
4.3	Final tool.....	20
4.3.1	GeoServer .....	21
4.3.2	MapServer .....	33
<b>5</b>	<b>Comparison and Results .....</b>	<b>39</b>
5.1	Visualization with Python .....	39
5.2	Map layout .....	39
5.3	Comparison of GeoServer with MapServer.....	40
<b>6</b>	<b>Conclusion .....</b>	<b>50</b>
<b>7</b>	<b>Bibliography .....</b>	<b>52</b>
	<b>Appendix .....</b>	<b>57</b>

## List of Code

Code 1 Folium: getting raster with NumPy .....	18
Code 2 Folium: Creating the map.....	19
Code 3 Bokeh: Creating the map.....	19
Code 4 GeoServer: SLD style NamedLayer .....	28
Code 5 GeoServer: SLD Style: FeatureTypeStyle .....	29
Code 6 GeoServer: OpenLayers library .....	30
Code 7 GeoServer: OpenLayers overview map style .....	30
Code 8 GeoServer: OpenLayers map element .....	31
Code 9 GeoServer: OpenLayers overview script .....	31
Code 10 GeoServer: OpenLayers - map layers .....	32
Code 11 GeoServer: OpenLayers map script.....	32
Code 12 MapServer: Mapfile: map object .....	33
Code 13 MapServer: Mapfile: WEB object .....	34
Code 14 MapServer: Mapfile: LAYER object .....	34
Code 15 MapServer: Mapfile: CLASS object.....	35
Code 16 MapServer: Mapfile: LEGEND and SCALEBAR objects .....	35
Code 17 MapServer: template .....	36
Code 18 MapServer: template map and scalebar .....	36
Code 19 MapServer: template zoom .....	37
Code 20 MapServer: template legend .....	37
Code 21 MapServer: OpenLayers legend element.....	37
Code 22 MapServer: OpenLayers map object.....	38

## List of Figures

Figure 1 QGIS - preview .....	17
Figure 2 GeoServer: Workspace .....	21
Figure 3 GeoServer: Raster Data Source .....	22
Figure 4 GeoServer: Edit Layer: Data .....	23
Figure 5 GeoServer: Edit Layer: Publishing .....	24
Figure 6 GeoServer: Edit Layer: Tile cache .....	25

Figure 7 GeoServer: WMS Metadata .....	25
Figure 8 GeoServer: WMS: Resource consumption limits .....	26
Figure 9 GeoServer: Style Editor .....	27
Figure 10 GeoServer: Style Editor validation.....	28
Figure 11 GeoServer web administration: Layer Preview .....	42
Figure 12 GeoServer Preview .....	42
Figure 13 MapServer Preview .....	43
Figure 14 GeoServer Network Performance .....	45
Figure 15 MapServer Network Performance .....	46
Figure 16 Zoom to Cairo and Faiyum Oasis.....	48

## List of Tables

Table 1 Visual variables after Bertin (1981, p. 187) .....	8
Table 2 Runtime with Bokeh .....	20
Table 3 Comparison of GeoServer and MapServer .....	40
Table 4 Soft-level comparison GeoServer vs. MapServer .....	44
Table 5 Loading times GeoServer vs. MapServer.....	47
Table 6 Time after zoom comparison.....	49

## Abbreviations

API	Application Programming Interface
B.C.	Before Christ
Chrome DevTools	Chrome Developer Tools
CGI	Common Gateway Interface
CIESIN	Center for International Earth Science Information Network
CITE	Compliance and Interoperability Testing Initiative
DD	Decimal Degrees
DOM	Document Object Model
EPSG	European Petroleum Survey Group
EU	European Union
GB	Giga Byte

---

GDAL	Geospatial Data Abstraction Library
GeoTIFF	Georeferenced Tagged Image File Format
GIS	Geographic Information System
GRUMP	Global Rural Urban Mapping Project
HEX	hexadecimal
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
KB	Kilo Byte
MB	Mega Byte
MS4W	Map Server For Windows
OGC	Open Geospatial Consortium
OSGeo	Open Source Geospatial Foundation
OSM	Open Street Map
OWS	Open Web Service
RAM	Random-Access Memory
RGB	Red Green Blue
SLD	Styled Layer Descriptor
TTFB	Time To First Byte
UN DESA	United Nations Department of Economic and Social Affairs
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WCS	Web Coverage Service
WFS	Web Feature Service
WGS	World Geodetic System
WMS	Web Map Service
WMTS	Web Map Tile Service
XML	Extensible Markup Language

## 1 Introduction

It is common knowledge that the world population is constantly growing, and the number will most likely cross the 8 billion mark in 2025. At the state of this report, there are over 7,6 billion people living on Earth. Over half of the population lives in urban areas, in 2014 it was 54 percent. According to the United Nations Department of Economic and Social Affairs (UN DESA) this number will rise to approximately 66.4 percent in the year 2050. Not only are there more people living in urban areas because the overall population is increasing, but the statistics show that the number of people residing in rural areas will decline, while the population in urban areas will grow (UN DESA, 2014). These numbers show that urban areas are growing. Simulations about estimated populations per country until 2100 exist and are easily accessible through various websites ([worldometers.info](http://worldometers.info); [worldpopulationhistory.org](http://worldpopulationhistory.org); [worldpopulationreview.com](http://worldpopulationreview.com)), though most of them are based on the World Population Prospect: The 2017 Revision published by UN DESA (2017).

However, the question where those people are going to live is more difficult to predict as many factors are involved. The charts that can be found online show estimated numbers for each country and it is also possible to find percentages of the distribution between rural and urban areas. But it is harder to find out which cities might grow and on the other hand where will cities decline and maybe even vanish, i.e. where exactly do people live. Simulations can support an effective understanding of possible changes and lead to predictions. These predictions are important for future decision-making processes. To make decisions concerning policies and about the distribution of resources, such as water, food and energy, it is of great importance to know more about the future developments of population distributions and growth of cities.

To gain knowledge and access the information from the simulations, these need to be visualized.

### 1.1 Problem statement

To visualize the simulations several issues have to be dealt with. The biggest issue when it comes to simulations that span the entire world, is the size of the dataset. Not only is there a lot of information that needs to be displayed all at once but also the file sizes need to be processed in a user-friendly timespan. To deal with the limited space an interactive map can help with a pan and zoom function. But to process very large rasters with interactivity, a powerful tool is needed. The aim of this project is to develop an interactive visualization tool that can handle large file sizes and makes it possible to interact with the end product. Another aspect of the visualization process is the styling of layers, which should be easily



adjustable to match new simulation outcomes. The tool has been developed for the use case of population simulations, but it is also adaptable for other large rasters. Another application for example could be the visualization of remote sensing data, which is likewise large in file size.

## 1.2 Research question

The following questions will be answered in this thesis:

How can an interactive tool improve the visualization of population simulations and what standards need to be met?

1. What are the visualization conventions that population simulations should follow?
2. Which tools already exist and what can they do?
3. How do these tools compare?
4. Can these tools be adapted for further development?

The research questions are built around the exploration of eligible tools to create a visualization, which will be based on a literature review. It will be researched which tools already exist and if they are capable of handling large datasets. The research will also include the invention of appropriate comparison methods. The comparison will be based on processing time and workflow of the final tool.

## 1.3 Report structure

To answer the research questions, the background to the population simulations is given in chapter 2.1. Here the project FLOW is described and the simulations behind the raster used in this project are explained. To give an introduction into map making and data visualizations a brief theory background is given in chapter 2.2. This also explains conventions in a visualization context and point out the importance of color. Thereafter, tools and related work are described in chapter 3. Here different libraries are explained, followed by a server solution, which is explained in more detail, considering the two chosen servers. The development of the visualization tool is described in chapter 4, where a first trial version was tested and afterwards two servers implemented. In chapter 5 the two servers are compared and finally a conclusion is drawn in chapter 6.

## 2 Background

### 2.1 FLOW and population simulations

There are different simulations of future population deployment, for this thesis the one from Keßler and Marcotullio (2017) was chosen, as this visualization tool has the aim to help with further projects such as “Global flows of migrants and their impact on North European Welfare States (FLOW)” at Aalborg University.

FLOW is an interdisciplinary research project combining the Technical Faculty of IT and Design with the Faculties of Social Sciences and Humanities, under the theme “Denmark and the global flows of the future” (“FLOW,” 2018). The aim of the project is to find out more about migration flows, in particular those connected to climate change, and their effect on the developed Northern European welfare states. Besides those migration scenarios, application-oriented research will be done to find solutions for socioeconomic and sociocultural integration of immigrants. The project has started in Spring 2018, with a duration of 36 month (ibid.).

“A Geosimulation for the Future Spatial Distribution of the Global Population” was implemented by Keßler and Marcotullio (2017) and is the base for this project. The simulations are using the Python programming language throughout the entire process. The simulations are based on a 1x1 kilometer grid at the equator and calculate an approximate population number for each 1 square kilometer. The present number of urban and rural population distribution originates from the Global Rural Urban Mapping Project (GRUMP) and is based on a 1 square kilometer raster cell (CIESIN et al., 2011). The predictions come from UN DESA, these global population prospects deliver population numbers until 2100 and distinguish rural and urban areas by country (UN DESA 2015). As GRUMP overestimates urban areas, they were combined with the European Space Agency’s Global Land Cover Map. They have a higher resolution of 300 meters and are more accurately in terms of defining urban areas. After down sampling them to GRUMP’s resolution they were overlaid and the areas that were marked as urban areas in the GRUMP raster file but not classified as such in the other one, were marked as suburban areas. Each of these layers fill up approximately 3 GB in uncompressed format. To make it processable each layer was split to smaller rasters, each containing the datasets for one country and after computation they were stitched back together.

The simulation is based on a few repeating steps. After generating a copy of the urban/suburban/rural layer from the previous iteration, the projected population numbers for urban and rural areas were retrieved from the UN DESA dataset. The numbers from the copy were randomly added or removed from the urban and rural areas to match the UN DESA predictions. When population numbers became too high

they were pushed into neighboring cells and change the area to fit the country-specific threshold for each class. After that, the new numbers for rural/suburban/urban areas were compared to match with the UN DESA predictions again. These steps are done for every ten years until 2100 and for every country. The countries are stitched together to make two global rasters in the Georeferenced Tagged Image File Format (GeoTIFF) for every ten-year interval. One showing the population number of each cell and one showing the distribution of rural/suburban/urban areas. One of these GeoTIFFs is approximately 500 MB big in compressed form, so even visualizing one raster at a time needs a powerful visualization tool.

## 2.2 Visualization conventions

With an ever-growing number of datasets, the need to come up with ways to display more complex and difficult data, other than in tables, is on hand. In this chapter, a brief history of data visualization and cartography is given, followed by a short explanation about what data visualization is and what issues come with it. This leads to visualization conventions, that can make it easier to find the right visualization methods. The different aspects of data representation that are related to maps are also described and a focus is laid on the right choice for colors.

### 2.2.1 Data visualization and Cartography

The history of data visualizations is going back to the very first tables of star positions and the first idea of coordinates came from ancient Egyptians as early as 200 B.C. In a brief history of data, Friendly (2008) describes the development of data visualizations, beginning from ancient Egypt to new formations of graphical display accompanied by first studies about statistics and probability theory in the 1600s. Although, early cartographic maps and statistics were used for their own purposes, it took until the 17<sup>th</sup> century to combine the two (Tufte, 2015). Always lead by new technological developments, such as color printing, the base for modern day visualization was made during that time (Friendly, 2008). One of the early and worth mentioning mapping of patterns was done by Dr. John Snow, who mapped deaths by cholera on a map of London in 1854. He found out that the most cholera deaths occurred close to specific street water pumps, which then were found to be the source of the disease and were decontaminated to stop the epidemic (Snow, 1855). This is an early example of using maps to reveal more information, then can be seen by simply looking at the numeric data. Other methods of displaying statistics, as we know them today, such as pie charts, line graphs, histograms and scatter plots took until the 1900s to develop (Friendly, 2008). However, Tufte says that maps are the most powerful method of displaying statistical

information when it comes to very large datasets. Only with a map can millions of data points be shown on a very small space (Tufte, 2015). This is especially important to notice, when approximately eighty percent of all digital data is geospatially referenced (MacEachren & Kraak, 2001).

Today it is easier than ever to access data. There is a lot of open source data on the internet, besides many others do the United States (with over 280.000 datasets at the state of this report) and the European Union have open data pages (data.europa.eu, 2018; data.gov, 2018). The data is freely available and often downloadable in different formats, to fit all kinds of needs. With this huge amount of data, it is more important than ever to think about proper visualizations to access the information of the data and turn it into knowledge. But, to find a suitable visualization it must be explained what data visualization is.

Visualization is a process where numerous people, amongst others including the people who want to have the visualization and the ones making it, make several choices e.g. about the data structure and the design (Kennedy et al., 2016). To make this visualization process systematically, spatial data is transformed into a visual display. For this display to be scientific, it should follow certain cartographic design methods to make it systematic, reproducible, and transparent as well as follow certain aesthetics (Garlandini & Fabrikant, 2009).

The choices that comprise cartographic design and aesthetics have a political and historical background, even though they are not always intentionally implemented in the result (Kennedy et al., 2016). But they have an effect and can also be used for political interests and must therefore be dealt with. Monmonier (1996) already claimed that maps can be used to tell lies. Different perspectives and representations can be achieved through specific projections, where either country borders are distorted, or their size is shown larger than the actual size, hence a more powerful appearance (Kennedy et al., 2016). With the same data many different maps can be made.

These negative and critical perspectives on data visualizations are in contrast with the view of visualization designers, who claim to 'do good with data'. This statement is the main statement of the visualization firm Periscope that want to gain transparency and public awareness through their work (Periscope, 2018). Even though Monmonier addresses mapmakers that are lying with maps, he also argues that it is needed to not tell the whole truth, which is similar to telling lies (Monmonier, 2005). It is an important part of map making to distort reality, in order to be able to focus on relevant relationships (Monmonier, 1996). The process of creating maps is a process of selecting and generalizing data, to show only the most important information (Mose & Strüver, 2009).

These diverse ways of seeing visualizations make it clear that maps should be both read and made with care, and attention should be paid to possible misleading visualizations.

### 2.2.2 Conventions

In the map making process, numerous conventions can help achieving a properly made map and help avoiding misleading visualizations. What conventions are – regarding visualizations – must be clarified, and examples of some conventions are given to understand and work with them.

“A convention is a symbolic or social practice that is shared, readily understood and widely accepted by members of a cultural group” (Kennedy et al., 2016). Often symbols and practices are intuitively used, due to their common appearance. There are different conventions, the most important ones, that are connected to map making are explained in the following.

Some intuitive design principles have become internationally accepted conventions, such as “light is less–dark is more” (Garlandini & Fabrikant, 2009). Studies show, that yellow to red colors are identified as warm colors whereas blue-green colors are seen as cold colors (Moreland, 2009). These can be translated into blue colors as low values and red color as high values, according to temperatures. Therefore, they are easily understood and even without a legend an idea of the values is given. Other conventions come from the 18<sup>th</sup> century topographic maps, especially the use of color, where for example water is rendered in blue, while forests have a dark green color (Kraak & Ormeling, 2010). These concepts are widely accepted and often intuitively followed.

When these conventions are followed, visualizations can be seen as professional and well made, as they align with the user’s expectations of a professional map. This professionalism on the other hand can give quality and also objectivity to the representation (Kennedy et al., 2016). On the contrary, when these conventions are not followed, it must be clearly stated in an easily understandable legend, that other colors or color scales are used than usual, to avoid misleading information.

Conventions can likewise help in showing more information. Spence (2014) explains it with a magnification map, that shows a world map with the population density determining the size of the country representation. The usual and conventional map with a conventional extend and center of the map is generally known. This makes it possible to display the countries not in their original size but according to their population density and through that showing more information. The color was then used to show a percentage of population increase (Spence, 2014). This is meant to show that conventions are not only

helpful to make a standardized map but also to filter out distracting information and represent more relevant information.

Conventions are also used for interactivity. For example is a text in blue and possibly underlined an indicator for an interaction. Another sign of interaction, that became a convention is the change of the cursor appearance on a mouse-over (Spence, 2014). These conventions are meant to point out, that they help to reduce the cognitive effort in understanding and interacting with the map. This surplus of cognitive effort can then be used to perceive other, more relevant information.

### 2.2.3 Visualization processes

During the visualization process of a map, three basic attributes are essential to understand: the scale, the projection and symbolization (Monmonier, 1996). They will be described in their importance below. Representing reality mostly comes hand in hand with a much smaller map and therefore a scale is needed to show how much smaller the map exactly is. This is done with a map scale, which defines the ratio between a distance on the map and the corresponding distance in reality (Kraak & Ormeling, 2010). There are essentially three ways to state the scale of a map: in a simple description, such as “one inch to the mile”, as a ratio or as a graph (Monmonier, 1996). Monmonier (1996) argues that graphs are the safest way of showing the map scale, as there is no need to have a good understanding of distances. This is needed when representing the scale in a sentence, such as “one centimeter represents 500 meters”. Also, a ratio map, like 1:50.000, needs some sense of distances and arithmetic understanding. A graph on the other hand shows the relation simply on the map, and another advantage is, that it stays accurate even when the map gets reduced in size, without a change of the scale (Monmonier, 1996).

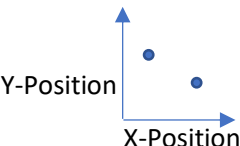






As already mentioned earlier, mapping comes hand in hand with some sort of distortion of the real world. This is due to the fact, that a three-dimensional ellipsoid is presented on a two-dimensional plane (Kraak & Ormeling, 2010). There are different ways of achieving this transformation, but all of these projections have distortions and neither of them is perfect. Which type of projection should be used, depends on the project and area that needs to be displayed on a plane. Each projection focuses on some of the distorting factors, that can be area sizes, shapes and distances.

A conformal map shows every angle on the map equal to the angles on the ellipsoid, consequently the shape is the same. The compromise is the distortion of size, that can lead to misperception of country sizes, especially at high latitudes (Bertin, 1974). A true to size projection with equal-area proportions on the other hand has quite distorted shapes. Another type of projection is equidistant, which stays true in

distances. All projections have either a point or a line of zero distortion and further away from this the distortion increases (Kraak & Ormeling, 2010). This line or point of zero distortion, also called tangential point or line, can be moved to overlap with the area to be mapped, so that this area gets the least distortion. However, this works only for a small area and not for a map of the entire world.

The third essential part in map making following Monmonier (1996) are symbols. He defines six different variables, that align with the visual variables from Bertin (1981). Bertin structures symbols into visual variables with different perceptions. He defines six as retinal variables to define the appearance of symbols and two planar variables. The latter define the dimension of the plane, which determines the location on the X and Y axis. The other visual variables are size and value as ordered variables; texture, color, orientation and shape as differential variables. They are all shown in the following table.

Table 1 Visual variables after Bertin (1981, p. 187)

<b>Planar Variables</b>		Position
<b>Ordered Variables</b>		Size
		Value
<b>Differential Variables</b>		Texture
		Color
		Orientation
		Shape

Symbol size is an obvious characteristic, that the user will automatically translate into differences in numbers (Kraak & Ormeling, 2010). The same occurs to values, where the conventions of less is light and dark is more applies, thus making these two variables ordered variables with some ranking implied. Size is especially good in representing amounts or counts, whereas value is better for rates or intensity (Monmonier, 1996). For qualitative differences symbols in varying shapes, textures and hues are best for differentiation. Orientation can be used to represent movements, such as is wind directions or migration streams (ibid.).

After clarifying these concepts, it is easier to apply them to the needs of a project. It should point out, that it is important to know the purpose of the project. Who is the audience or user of the map? What kind of data will be used? And what area should be mapped? These questions must be answered in order to ensure an efficient visualization.

#### 2.2.4 Color

Another important part of visualization is turning data into colors. This can be done in many ways and plays a key role in visualizations. Color is widely left to conventions and these colors are merely differentiated in their simple use, as in blue for water and green for forests (Nowell, 1997). But an appropriate use of color is important for data representations. The right use of color makes the content easily understandable and allows interrelationships and patterns to be recognized. On the contrary, a careless use can obscure those patterns (Brewer, 1994).

When talking about colors, a few concepts must be understood. The type of data is important to choose between different color schemes. These color schemes can then be adjusted within the three perceptual dimensions of color: hue, lightness and saturation. Hue is the name of the color, such as red, blue or green. These are also commonly referred to as colors but Brewer(1994) argues that a color is a composition of hue, lightness and saturation. Lightness can also be called value and shows the brightness or darkness. Whereas, saturation can be seen as the amount of hue in a color (Brewer, 1994). It is also known as chroma, an example for saturation would be a grayish red compared to a pure red.

Data can be de organized in two main classes: qualitative and quantitative data. Qualitative data is data that is not ordered nor has a ranking, for example nominal or categorical data. Here it is advisable to use a qualitative color scheme, where the primary differences are in the choice of hues, with constant lightness and saturation (Brewer, 1994). A color scheme based on lightness could imply an order, such as light is less and dark is more. An online tool for selecting colors is [colorbrewer2.org](http://colorbrewer2.org) (ColorBrewer, 2018). It supplies



three different color schemes within qualitative and quantitative data. Besides qualitative color schemes the website offers two color schemes that fit quantitative data (Harrower & Brewer, 2003). Quantitative data has an order or is ranked and can also be continuous. With this kind of data, lightness is the main differentiator. A diverging color scheme has two essential color components on both ends of the range. They transition from one color on the one side, over an unsaturated or light color in the middle to the second main color on the other side (Moreland, 2009). Here an equal emphasis is on the mid-range and the critical values to both sides. These low and high extremes are represented in dark colors and have contrasting hues (Brewer, 1994). The last color scheme also serves qualitative data, especially ordered data. In this color scheme the data, hence the color scale is progressing from low to high. Lightness is dominantly defining this scheme and the conventional, light represents low values and dark high values, is mostly used. In some cases, such as dark backgrounds, the emphasis could be on light values for high numbers and must be clearly stated in the legend.

Population numbers come as quantitative data, with areas of zero population at its lowest and high population numbers in very dense urban areas. Therefore, a sequential color scheme is fitting.

### 3 Tools and Related work

To find a suited interactive visualization tool, research was done looking for existing solutions and related work. Important was, that the outcome was an interactive visualization tool. There are some libraries and frameworks that can be used for his project. It must be tested if they are suitable to visualize these large datasets too. Performance plays a very important role in the project due to the very large datasets, which need a lot of processing power. Besides the interactivity and handling of big data, the tool should be run using the Python programming language. Python is the preferred language, as it is already known, and the population simulations are also written in Python, so the workflow is seamless.

The first research was done with a simple google search for interactive visualization tools using Python. During the first process a few libraries were taken out of consideration after looking into they're documentation. As this tool is also meant to be used after the end of this project, a sustainable solution must be found. Therefore, small libraries that were developed only recently or not supported from well-known organizations, were excluded. The documentation was also a big part of the decision making and a poor documentation of how to use the library was also an excluding factor. After this first research two libraries sounded promising and were therefore considered as a feasible option and further research was done. These libraries are described below, and other used software is explained.

### 3.1 Python

Python is a widely used programming language, due to its interoperability, as it runs on many different operating systems. It is an interpreted<sup>1</sup>, object-oriented<sup>2</sup> programming language that is also interactive<sup>3</sup> (Langtangen, 2009). It has a very clear syntax which enables programming for short and long scripts. Python has a vast variety of additional libraries for different purposes and can easily be extended. However, it can also be used on its own, as it has a large built-in library (Python Software Foundation, 2018b). Python is an open source programming language that is managed, protected and advanced by the non-profit corporation Python Software Foundation (Python Software Foundation, 2018c).

In this project the Python version 2.7.5 was used within the PyCharm Integrated Development Environment (IDE) from JetBrains. This IDE was used, because of its intelligent code completion and on-the-fly error checking (JetBrains, 2018). It was also used in previous projects and therefore already known and comfortable with it. To install and manage python packages, the Anaconda Distribution was used (Anaconda, 2018).

#### 3.1.1 NumPy

NumPy is a library for the Python programming language. It is a package for scientific computation of large n-dimensional array objects. It comes with a large number of mathematic functions and can be used as an multi-dimensional container of generic-data (NumPy, 2018). NumPy arrays are high-level data structures that can contain compound data, instead of only one data type. This is useful for x- and y-coordinates stored as floating-point numbers. It is also very efficient when processing large data sets, as it does not need to copy all the data in the memory (Van Der Walt, Colbert, & Varoquaux, 2011).

#### 3.1.2 GDAL

GDAL stands for Geospatial Data Abstraction Library and is a translator library for vector and raster data. It is an open source library licensed under the Open Source Geospatial Foundation<sup>4</sup> (OSGeo) and is

---

<sup>1</sup> New code can be generated while the script is running

<sup>2</sup> bundels properties with behaviors or attributes into objects

<sup>3</sup> Running a command and immediately shows the result

<sup>4</sup> The Foundation (OSGeo) is an independent not-for-profit organization to support the needs of an open source geospatial community (OSGeo, 2018a)

compiled of two libraries. OGR<sup>1</sup> Simple Features Library holds source of the vector data while GDAL designs the raster part. However they have become more integrated, and will be completely folded into each other in the future, meaning that GDAL will hold both sources of vector and raster data (OSGeo, 2018c). It supports a vast amount of data formats, for both raster and vector. In this project the handling of GeoTIFFs is of importance and supported by GDAL. GDAL is written in C++ but has several language bindings, including one for Python. But it also comes with command line utilities for data translation and processing (GDAL, 2018). In this project GDAL is used in both a Python script and using the command line.

### 3.2 Bokeh

One of the libraries that was considered feasible to use as a visualization tool is the Python library Bokeh. It is an interactive visualization library that uses web browsers for their presentation. It is especially useful when it comes to very large datasets and combines this power with high-performance interactivity (Bokeh, 2018b). Bokeh comes in two interface levels. One is `bokeh.model` which is a low level interface, and mainly for developers with special needs for a finer control over the composition of Bokeh plots and widgets. However, there is an higher level interface, `bokeh.plotting`, that handles the assembling of Bokeh models automatically (Bokeh, 2018a). For this project the `bokeh.plotting` interface was used.

### 3.3 Folium

Folium is a library for Python that uses the Leaflet.js library for visualizations on a map. Leaflet.js is an open source library for interactive web maps, that uses JavaScript. It stands out through its lightweight and simplicity with high performance and usability (Leaflet, 2018).

Folium takes advantage of those abilities and combines them with the data handling strength of Python. The data is manipulated in Python and afterwards visualized on a leaflet map through Folium. The output map is a browser-based map by leaflet. Folium is the tool that calls the leaflet map in python (Folium, 2018).

---

<sup>1</sup> “OGR used to stand for OpenGIS Simple Features Reference Implementation. However, since OGR is not fully compliant with the OpenGIS Simple Feature specification and is not approved as a reference implementation of the spec the name was changed to OGR Simple Features Library. The only meaning of OGR in this name is historical.” (OSGeo, 2018b)

### 3.4 QGIS and QGIS Standalone app

Another option that was considered is a standalone app based on the QGIS application programming interface (API). QGIS is an open source geographic information system (GIS), that is also part of OSGeo. It is widely used, because it is not only open source but it also runs on Linux, Unix, Windows and Mac OSX operating systems (QGIS, 2018a).

QGIS supports scripting in the Python language. There are different ways to make use of it, it is usable within QGIS in a python console and to create and use python plugins. It can also be used to make standalone apps. Those will work without the need to open QGIS and are therefore faster apps for specific tasks. In order to use the standalone script, the QGIS resources have to be initialized in the python script. This can lead to some errors, as the Python path must be set in the system environment to the correct Python installation (QGIS, 2018b). This differs from operating system to operating system and also the way QGIS and Python were installed make a difference. It was tested with an installation of QGIS 2.18.3 and Python 2.7.5 with various tutorials but due to a lack of time it was not possible to dive deeper into advanced system environment settings and therefore other solutions were considered.

### 3.5 Server

A web server is a program that uses Hypertext Transfer Protocol (HTTP) to serve content. A browser is used to contact the web server in form of a Uniform Resource Locators (URL) request. The server looks through its own file system for a valid file that the request points to, it is often a .html page but can also be an image or a .xml file. If the file exists and the browser is configured to display the file type, it will be rendered in the browser, otherwise it will be open for a download (Boundless, 2018).

There are different web services for different purposes and web mapping server are specialized web server. They function in the same way but with a focus on geographic information. Within web mapping services there are also different specifications, but they follow the same standards implemented and developed by the Open Geospatial Consortium (OGC). OGC is an international not profit organization with the purpose of making quality standards for a global geospatial community, that are freely available and open to everyone (OpenGeospatial, 2018). The most common open web services for geographic information are described below.

Web Map Server (WMS) is the one that is used for this project. A WMS is rendering maps as images to provide them over a simple HTTP interface (OGC, 2006). There are three operations that can be invoked:

one returns a geographically well-defined map, one returns metadata, and a third optional one returns information about particular features. These operations can be requested and defined via URLs (ibid.). The URL defines what information, and in which extent the information is shown on the map. The operations for a WMS are GetCapabilities, GetMap and GetFeatureInfo, whereas the latter is optional. With GetCapabilities an Extensible Markup Language (XML) containing metadata is responded and can usually be downloaded (OGC, 2006). The GetMap request returns the requested map according to the requested parameters. GetFeatureInfo is used to retrieve more information about the map, for example when clicking on the map the pixel value should be shown. For this kind of information, a GetFeatureInfo must be used and implemented (ibid.).

With a WMS it is also possible to overlay maps and adjust the transparency to produce composite maps. A Web Map Service publishes and produces maps rather than accessing specific data. The geographic information is classified as “Layers” and can be styled with predefined “Styles” (ibid.).

Web Feature Services (WFS) on the other hand can change the way geographic information is created and modified on the internet (OGC, 2010a). It provides the capability to access geospatial features and also perform operations such as creating, updating or deleting features from the data store (OGC, 2016). Other web servers are Web Coverage Service (WCS), that is especially useful for multi-dimensional coverage data, for example terrain models (OGC, 2012). Besides these three main web services, there is also a Web Map Tile Service (WMTS) that focuses on high performance services for scalable cartographic maps. The OGC WMTS is inspired by the OSGeo Tile Map Service Specification and has a complementary approach to a WMS. While WMS focuses on flexible custom map styles, WMTS concentrates on static data such as base maps and their scalability, here the bounding box and scales have been bound to discrete tiles (OGC, 2010b).

The main two servers for geographic imagery are GeoServer and MapServer which will be described in their specifics in the following, the development will be explained in chapter 4 and a comparison of both is done in chapter 5.

### 3.5.1 GeoServer

GeoServer is a free and open source web server that uses the open standards from the Open Geospatial Consortium as described above. It is a Java Enterprise (J2EE) web application that enables the user to publish, edit and process geographical data (Deoliveria, 2018). GeoServer can share data from many spatial data sources and is designed for interoperability. GeoServer works with a browser-based web administration interface to define all server settings (GeoServer, 2018a). These include settings of image

processing, like memory usage, tiling and handling caches. Server settings can be made globally, meaning all Open Web Services (OWS) publish all layers configured on the server. WFS publishes all vector layers (feature types), WCS publishes all coverage layers and WMS can publish all. For security reasons the client can be limited to only have access to selected layers, this can be done with a virtual service, which is a view of the global service (GeoServer, 2018c). The layers that are shared over the server can be styled using the markup language Style Layer Descriptor (SLD) in a separate style file (Iacovella & Youngblood, 2013). It is an XML-based markup language with the extension .sld in GeoServer that is supported from OGC (OGC, 2007). It can be edited within the editor in GeoServer or better in a preferred text editor and then copy it into the editor, as web administration sessions can expire and what was done, can get deleted. Within the web administration interface, the layers have a preview, that will be shown in the comparison section, with a simple OpenLayers viewer. For more advanced stylings OpenLayers can be used separately. The version that was used for this project is the 2.13.0 stable release and downloaded from the [geoserver.org](http://geoserver.org) (2018a) website.

### 3.5.2 MapServer

MapServer is also a free and open source web server. It is written in C and is one of the founding projects of the OSGeo foundation and supports numerous OGC standards (MapServer, 2018a). MapServer can be used in two different ways, as a Common Gateway Interface (CGI) or with a MapScript. For this project, the CGI program was used, because it is easy to set up and straight forward to use (Kropla, 2005).

When used as a CGI program, it receives requests and returns responses in the form of e.g. images or maps. When requested the CGI program first reads the mapfile, with all configurations about the map and layers, then it draws the map. The next step is to read the hypertext markup language (HTML) template files that are indicated in the mapfile. This is used to specify the layers that should be rendered or the zoom level. Then the values in the template are substituted with the actual renderings and send to the web server, which forwards it to the browser (Kropla, 2005). One mapfile configures one server but can have numerous templates, for different purposes. For a simple zoom function only one template is enough (MapServer, 2018c).

MapServer with all its components was installed with MapServer For Windows (MS4W), which sets it up for windows (McKenna & Gateway Geomatics, 2018).

### 3.6 OpenLayers

The visualization was then done in OpenLayers. OpenLayers is an open source JavaScript library and comes with a good documentation and many examples and tutorials. It is made to show dynamic maps and embed them into web pages. It can display maps from different sources and handles vector and tiled layers (OpenLayers.org, 2018). It can use different web servers as map sources, amongst them GeoServer and MapServer. OpenLayers also works well with the OGC standards and the WMS can be implemented. It is easy to adapt for custom web pages, which is especially useful when the visualizations should be published. The version that was used in this project is the newest version 4.6.5.

### 3.7 Comparison tools

To be able to adequately compare the different libraries and servers, two tools were used. To test the processing time of a python script, the module timeit was used. Timeit has a command line interface and is also callable within a script. To be independent from other interfering operations that could slow down the runtime timeit repeats the same steps a given number of times and prints out the average of the results (Python Software Foundation, 2018a).

For testing the server performances on an objective level, the Google Chrome development tools (DevTools) were used. After opening the Google Chrome browser DevTools opens when pressing control + shift + i (for mac: command + option + i). Besides the many uses of DevTools, for this comparison only the network and performance panels were used. With the network panel the user learns about the loading times of a page (Basques & Chrome DevTools, 2018a). In the performance panel the runtime and interaction of the page can be tested and monitored (Basques & Chrome DevTools, 2018b).

## 4 Development of the tool

### 4.1 Reference version

As a benchmark for the visualization tool the former process is described. This is meant to serve as a comparison and to evaluate possible improvements that come with a separate visualization tool.

The former process was to load the GeoTIFF in a GIS-software such as QGIS or ArcGIS and then process it from there. This included loading the full GIS software and loading the file into it. A predefined layer style could be added, so that the manual styling can be dropped. The raster could be shown within the program interface. It was tested in QGIS and a screenshot can be seen in figure 1.

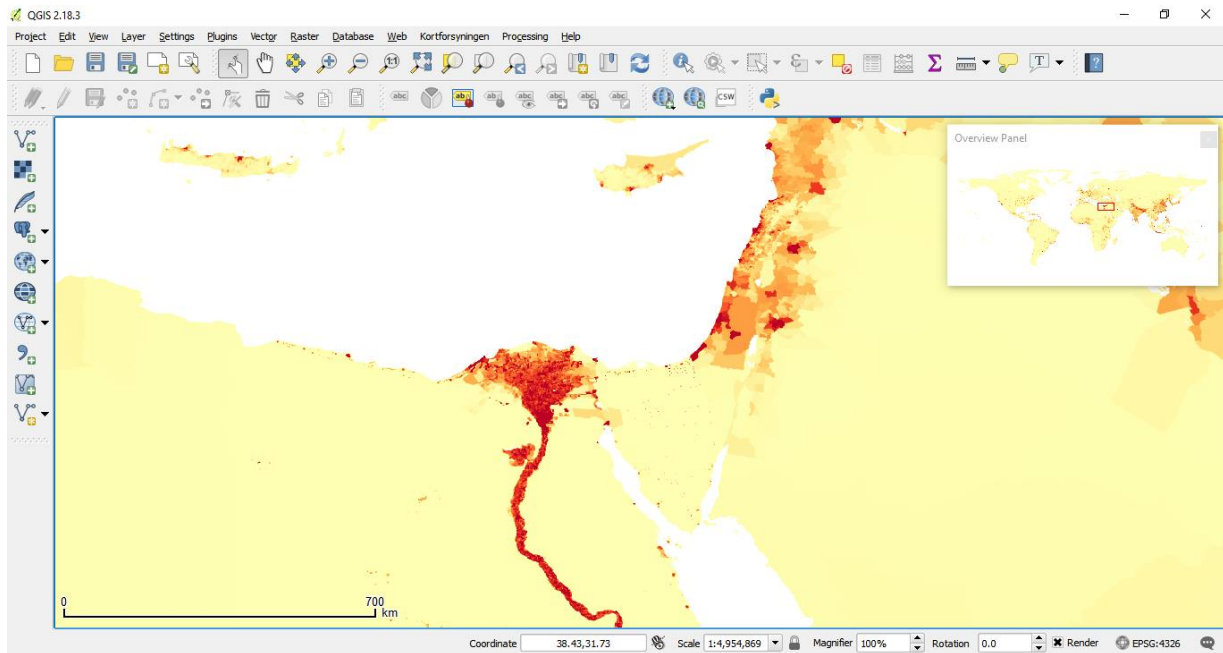


Figure 1 QGIS - preview

It can be seen, that it is possible to add an overview panel and a scale to the screen. The pan and zoom function work smoothly and the raster loads quickly. However, this is not an ideal setting for the presentation of results and the map should be exported as a static pdf file or an image and then be embedded in a presentation. Here an interaction is not possible anymore.

## 4.2 First trial versions

Before the programming and final solution can be made, a trial and example version was tested. The trial version should be a very simple version of the final tool, to see if the frameworks are eligible for further development. After testing this first attempt, further examination was done and two server versions were developed, and in the following chapter tested against each other.

The trial version should have a simple viewer with a zoom function. This is crucial for the final tool to work smoothly and therefore also included in the first attempt. The zoom function will also be used as an indicator of the performance of the interactive visualization tool in the end. If it is too time consuming to smoothly zoom in, it does not fit the criteria for the needed tool.



As the main issue with the visualization is the size of the data, the first trial version is as simple as possible, to find out if the libraries and servers can handle the data and test their usability. As the final version should be able to handle the entire dataset of 378 MB but also be able to run on a standard computer without too much trouble, a lightweight computer was used. The computer that was used throughout the process is a Lenovo laptop E31 with 4 GB RAM and 64-bit windows operating system. It has an intel core i3-5005U processor with 2.00 GHz.

The testing process included testing of the script with a smaller GeoTIFF. To see how the libraries can handle the file sizes, the original one – of the entire world – was clipped to different sized GeoTIFFs. This was meant to show that the script and the library is working and technically correctly set up. The smallest one, approximately clipped around Denmark, has 1.09 MB. One clipped around the European Union (EU) is larger with 82.4 MB and two different clips around Australia were made: one with a file size of 218 MB and a bigger one, closer to the original size with 295 MB. These clips were simply clipped from the original 378 MB population GeoTIFF in QGIS using the raster extraction tool. The process for each library and the results from it are explained below. The complete scripts can be found in the appendix in the order they are described here.

#### 4.2.1 Folium

The first trial was done with folium. It was installed through Anaconda and used with the PyCharm IDE. Folium is a plotting library that can plot data on leaflet maps, for the visualization of raster data the GDAL and NumPy libraries were used. The GeoTIFF was opened with GDAL (line 8), the raster band extracted (line 9) and read as a NumPy array (line 10).

```
8  src = gdal.Open('PopDK.tif', gdal.GA_Update)
9  band = src.GetRasterBand(1)
10 imarray = np.array(band.ReadAsArray())
```

*Code 1 Folium: getting raster with NumPy*

Then a map object was created with the starting zoom level of 3 in line 13. The NumPy array was then added to the map with the `folium.plugins.ImageOverlay` command (line 15-20) and the map was saved as a html file to be opened in a browser (line 23).

```
13 map= folium.Map(zoom_start=3)
14
15 folium.plugins.ImageOverlay(
16     image=imarray,
17     bounds=[[-180, -60], [180, 85]],
18     colormap=lambda x: (1, 0, 0, x),
19     origin='lower',
20 ).add_to(map)
21
22 #Save html
23 map.save('folium_test.html')
```

*Code 2 Folium: Creating the map*

During the running process with the original file size the computer crashed and turned itself off. To pinpoint the issue and to clarify if the library and the script worked, a small GeoTIFF was used. It showed that it worked well with the 1.09 MB GeoTIFF of Denmark. A larger raster with the size 82.4 MB took 3,5 minutes run time. As this is only a fraction of the actual size, the library folium was not considered as an applicable package for this type of project.

#### 4.2.2 Bokeh

The first trial with the Python library Bokeh also used the large 378 MB GeoTIFF. Here it worked better than with the folium library, but a memory error occurred during the process. The script was then again tested with the smallest raster of Denmark and the run time was very short and the browser opened immediately afterwards showing the raster.

The GeoTIFF was opened with GDAL and read as a NumPy array in the same way as with folium, which can be seen in code snippet 2. The entire Bokeh script is in the Appendix including the timeit statements used to measure the time. In Bokeh an output file was created as a html document in line 11. In line 13 the image gets turned around, as Bokeh would otherwise show it upside down. A figure is created with the coordinates as axes (line 14), it displays the previously created NumPy arrays in the figure (line 15, 16) and afterwards it is saved (line 17).

```
11 output_file(r"C:/master/imageDK.html")
12
13 imarray = imarray[::-1]
14 p = figure(x_range=(-180,180), y_range=(-60,85))
15 p.image_rgba(image=[imarray], x=[-180], y=[-60], dw=[360], dh=[145],
16             dilate=False)
17 save(p)
```

*Code 3 Bokeh: Creating the map*

The script was run with all different raster sizes to see how large they can be without difficulties. It was distinguished between the running time of the code, measured with the Python module `timeit`, and the following loading time in the browser, measured with a simple clock timer. The following table (2) shows the different results.

Table 2 Runtime with Bokeh

Country GeoTIFF	Size	Timeit (seconds)	Loading time (mm:ss)
DK	1.09 MB	3.08	00:01
EU	82.4 MB	113.75	00:30
AUS	218 MB	345.14	00:50
AUS	295 MB	479.68	01:16

It can be seen, that the largest file size, which is still smaller than the original one, already took almost eight minutes (479.68 seconds) to process the raster and over a minute to load completely in the browser. This is better than with Folium, but not perfect for a smooth visualization.

#### 4.2.3 Tiles

Another option that was considered, was splitting the image up in tiles. For this purpose, the GDAL tool `gdal2tiles.py` was used. To generate the tiles for 8 zoom levels, it took one hour and eighteen minutes to receive the tiles. A styling after the tiles are created is not possible, which means for a new styling new tiles must be created again. This process takes too long, especially when considering, that a change over time should be presented and there are different GeoTIFFs for different years. Therefore, this solution was dropped as not applicable due to a too long processing time.

### 4.3 Final tool

After the direct visualization with the help of Python libraries failed, a server solution was considered. How a server works is explained in chapter 3.5 and the two servers that were implemented in this project are also described in that chapter. In the following the setup of the two servers is described. At first GeoServer will be explained, including the adding and organization of data and the WMS settings. Afterwards, the layer styling is explained for GeoServer, followed by the explanation of the OpenLayers implementation.

Thereafter, the MapServer configuration within the mapfile is explained, followed by the templating HTML file. In the end the OpenLayers document for MapServer is explained, where it defers from GeoServer.

#### 4.3.1 GeoServer

After installing GeoServer on the computer, the GeoServer web administration page can be accessed through this URL in the browser: `http://localhost:8080/geoserver/web/`

The web administration has numerous settings for all kinds of use cases. In the following, only the relevant ones are explained. With GeoServer, there are default values that do not need to be changed, but maybe activated to adjust the server to the user's needs. For example, access can be limited, or memory usage can be limited. For this project, nothing was limited and most of the settings were kept with the default values.

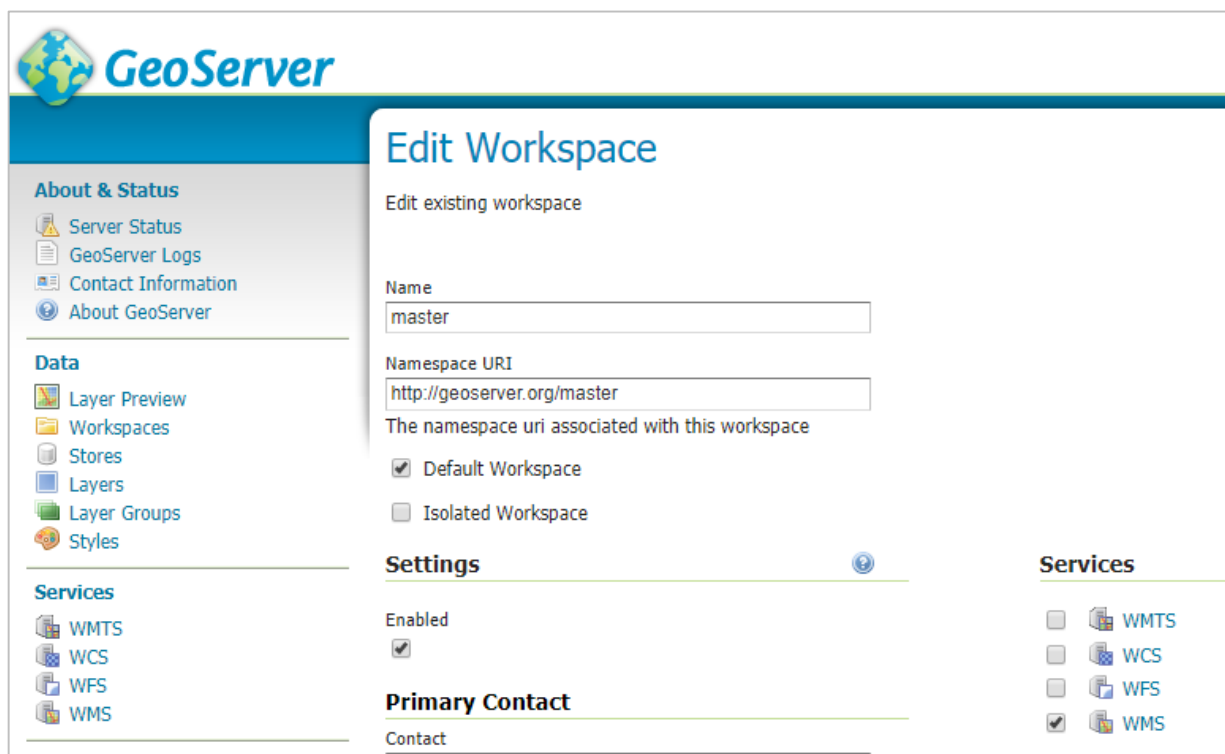


Figure 2 GeoServer: Workspace

Data in GeoServer is organized in workspaces, stores and layers, as can be seen on the left side of the screenshot above (Figure 2). A workspace organizes the items stored in it, for this project a new workspace was created called “master”, and the configuration can be seen above in the screenshot. A namespace Uniform Resource Identifier (URI) is given and the workspace is set to default. On the right side the services that the workspace should provide can be checked, here only the WMS is needed. With the settings enabled locally, opposed to global, the settings apply to only this workspace. Afterwards contact information can be given.

To add data, a new data store must be created within a workspace. The screenshot below shows the editing of a raster data source, but the window looks the same when a new store is added. When adding a new store the data type can be selected from a broad range of vector and raster data. A descriptive name must be given, and the path to the file must be set with a connection URL as shown below.

**Edit Raster Data Source**

Description

GeoTIFF  
Tagged Image File Format with Geographic information

**Basic Store Info**

Workspace \*

master ▼

Data Source Name \*

Master Population

Description

☒ Enabled

**Connection Parameters**

URL \*

file:data/master/Pop\_correct10.tiff [Browse...](#)

[Save](#) [Cancel](#)

Figure 3 GeoServer: Raster Data Source

The data store is then used to publish layers. Here a name and further description of the layer can be given in the basic resource info part, as shown below.

## master:Pop\_correct10

Configure the resource and publishing information for the current layer

Data

Publishing

Dimensions

Tile Caching

### Edit Layer

#### Basic Resource Info

Name  
Pop\_correct10

☒ Enabled

☒ Advertised

Title  
Pop\_correct10

Abstract  
World population in 2010

#### Coordinate Reference Systems

Native SRS  
EPSG:4326 [EPSG:WGS 84...](#)

Declared SRS  
EPSG:4326 [Find...](#) [EPSG:WGS 84...](#)

SRS handling  
**Reproject native to declared**

#### Bounding Boxes

Native Bounding Box

Min X	Min Y	Max X	Max Y
-180	-59.9999999999991	179.999999999999	85.0000000000008

[Compute from data](#)  
[Compute from SRS bounds](#)

Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-180	-59.9999999999991	179.999999999999	85.0000000000008

[Compute from native bounds](#)

#### Coverage Parameters

Input Transparent Color

Suggested Tile Size  
512,512

#### Coverage Band Details

Band	Data type	Null Values	minRange	maxRange	Unit
GRAY_INDEX	Real 32 bits	-340,282,300,000,000,000,0	-∞	∞	

[Reload band definitions](#)

[Save](#) [Cancel](#)

Figure 4 GeoServer: Edit Layer: Data

Further information about the layer is set in the Data panel of the layer editor. Here the coordinate reference system is set to the data native one, in this case the system with the European Petroleum Survey Group (EPSG) code: 4326. There it can also be reprojected if needed. The bounding box can be computed from the data or set manually, in this case it was computed automatically. Afterwards, coverage information can be given, and a suggested tile size is declared. In the information about the band, details are given with a definition of the null value.

The screenshot displays the 'Edit Layer: Publishing' interface in GeoServer. It is divided into three main sections:

- Interpolation Methods:** A dropdown menu for 'Default Interpolation Method' is set to 'nearest neighbor'.
- Formats:** A text input for 'Native Format' is set to 'GeoTIFF'.
- WMS Settings:**
  - Layer Settings:** Includes checkboxes for 'Queryable' (checked) and 'Opaque' (unchecked).
  - Default Style:** A dropdown menu is set to 'master:pop ras...'.
  - Legend:** A vertical color scale legend is shown with values 0, 100, 500, 1000, 2000, and 5000. The colors transition from light yellow at the bottom to dark red at the top. A red 'X' is placed over the 'nodata' label at the top of the scale.

Figure 5 GeoServer: Edit Layer: Publishing

In the publishing panel, the interpolation method is set. The nearest neighbor method was chosen, as this ensures that the image values stay the same, and the rendering is fast. Other options are bilinear interpolation with linear weighting, which gives more influence to the closer input cells, and bicubic which is used to smoothen continuous data (GeoServer, 2018e). Additionally, the native format is set to GeoTIFF and the layer style is chosen with a preview of the legend.

Furthermore, there is a “Dimensions” panel, where time and elevation data can be enabled if the data has it. The last panel manages the tile cache for this layer. It is set to create a cached layer and tile cache for the layer. Additionally, metatiling factors are defined, here the default settings are used and also the tile image formats are left to the default.

### Tile cache configuration

- ☒ Create a cached layer for this layer
- ☒ Enable tile caching for this layer
- ☒ Enable In Memory Caching for this Layer.

BlobStore

(\*) Default BlobStore ▼

Metatiling factors

4 ▼ tiles wide by 4 ▼ tiles high

Gutter size in pixels

0 ▼

Tile Image Formats

- ☐ image/gif
- ☒ image/jpeg
- ☒ image/png
- ☒ image/png8
- ☐ image/vnd.jpeg-png

Figure 6 GeoServer: Edit Layer: Tile cache

These settings are all layer or workspace settings, but the WMS must also be set up on its own. The default settings were kept and are explained briefly in the following.

## Web Map Service

Manage map publishing

### Workspace

master

### Service Metadata

- ☒ Enable WMS
- ☐ Strict CITE compliance

Maintainer

Online resource

Title

Abstract

A compliant implementation of WMS plus most of the SLD extension (dynamic styling).  
Can also generate PDF, SVG, KML, GeoRSS

Fees

Access Constraints

Figure 7 GeoServer: WMS Metadata



The service metadata for the WMS must be enabled in order to process requests the explanations are found in GeoServer (2018d). Strict Compliance and Interoperability Testing Initiative (CITE) enforces strict OGC conformance and is not selected. Required for the server to work is a given online resource, which defines the top-level HTTP URL of the server. Followed by a title and an abstract and the declaration, that no fees are imposed for the use of the server and no access constraints are imposed. Not in the screenshots shown are the options, that where not used and left blank, but they will be listed shortly. Keywords can be implemented, for better searching. Additionally, information about the root layer can be made, if left blank the WMS service information are used. An interpolation method can also be set globally for the WMS, in this case the interpolation method is also set in the layer options. Advanced projection handling and continuous map wrapping is enabled by default. These options help with good looking maps in areas at the edge of the projections and make a continuous map visualization possible (GeoServer, 2018e).

### Resource consumption limits

Max rendering memory (KB)

Max rendering time (s)

Max rendering errors (count)

Figure 8 GeoServer: WMS: Resource consumption limits

The resources that a request uses can be limited in the WMS settings, here the default settings were used and can be seen in the screenshot above.

### SLD style

To style the layer, GeoServer has an inbuild style editor, using the Styled Layer Descriptor (SLD) language, which can be seen in figure 9.

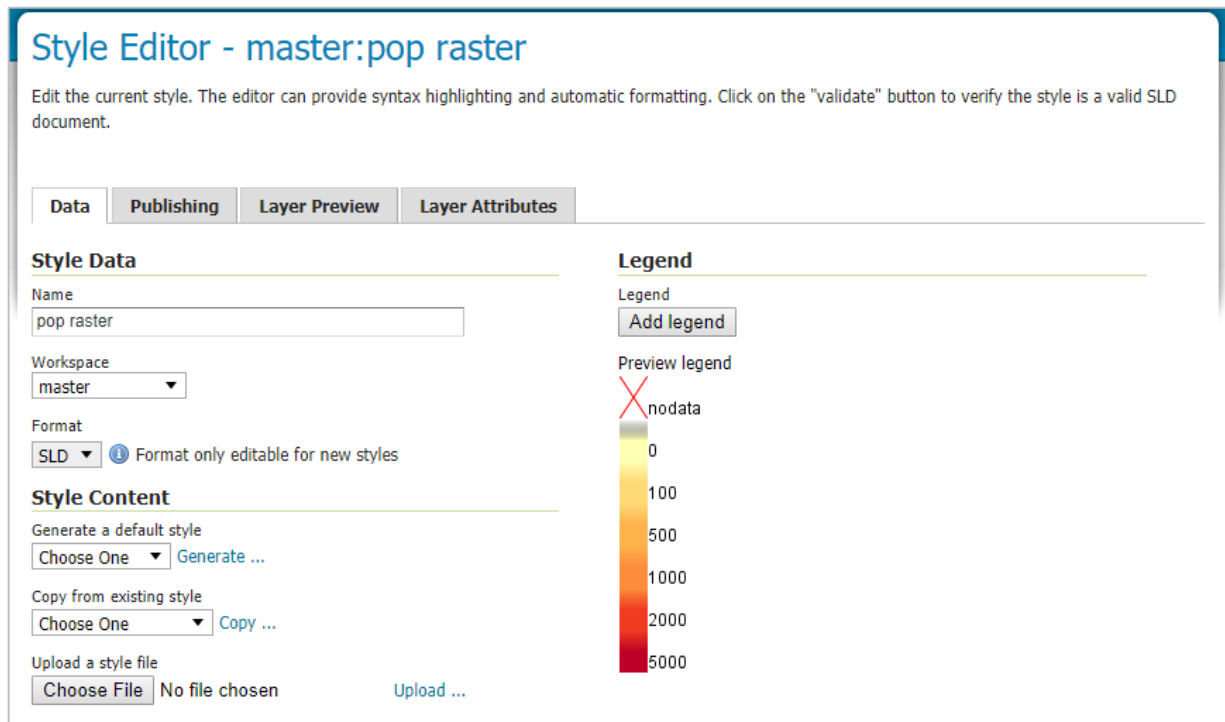


Figure 9 GeoServer: Style Editor

With the “Publishing” panel different layers from other workspaces can be associated with this style too. With the top label “Layer Preview” the style can be edited and the change can be seen immediately on the layer. The last panel “Layer Attributes” shows special attributes for the selected layer, e.g. the band and null values of the raster. Under the first, “Data” panel, default styles can be chosen in the Style content, to make sure the SLD file has the right settings. The screenshot below (figure 19) shows the Style Editor with the raster style. It can be seen, that there is a validate function at the bottom to see if there are errors in the document, that would hinder the style from working.

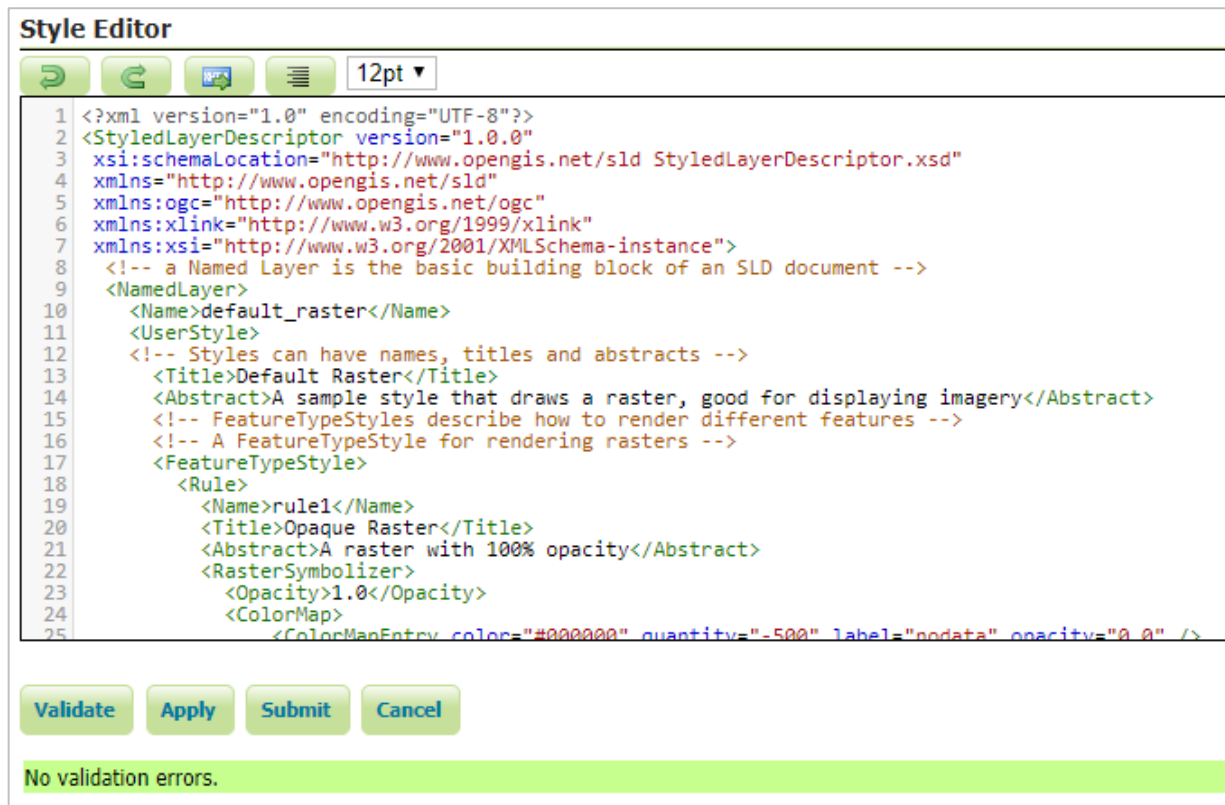


Figure 10 GeoServer: Style Editor validation

The SLD document must always start with the lines 1-7 as shown in the screenshot above. It consists of a declaration of the XML and SLD version. The entire SLD document can be found in the appendix and parts of it are described below. After the first crucial declarations, the style is defined with a `<NamedLayer>` beginning with line 9, where rules are defined for one single layer. This also includes a title (line 13) and a little description of the raster (line 14, 15).

```

8 <!-- a Named Layer is the basic building block of an SLD document -->
9 <NamedLayer>
10   <Name>default_raster</Name>
11   <UserStyle>
12     <!-- Styles can have names, titles and abstracts -->
13     <Title>Default Raster</Title>
14     <Abstract>A sample style that draws a raster, good for displaying
15       imagery</Abstract>

```

Code 4 GeoServer: SLD style NamedLayer

After that, the `<FeatureTypeStyle>` starts (line 18), where the rules for rendering are implemented. In the first part the opacity is set to 1.0 (line 21-24) which means that the layer is not transparent at all.

```

16      <!-- FeatureTypeStyles describe how to render different features -->
17      <!-- A FeatureTypeStyle for rendering rasters -->
18      <FeatureTypeStyle>
19          <Rule>
20              <Name>rule1</Name>
21              <Title>Opaque Raster</Title>
22              <Abstract>A raster with 100% opacity</Abstract>
23              <RasterSymbolizer>
24                  <Opacity>1.0</Opacity>
25                  <ColorMap>
26                      <ColorMapEntry color="#000000" quantity="-500" label="nodata"
27                          opacity="0.0" />
28                      <ColorMapEntry color="#ffffb2" quantity="0" label="0" />
29                      <ColorMapEntry color="#fed976" quantity="100" label="100"/>
30                      <ColorMapEntry color="#feb24c" quantity="500" label="500" />
31                      <ColorMapEntry color="#fd8d3c" quantity="1000" label="1000" />
32                      <ColorMapEntry color="#f03b20" quantity="2000" label="2000" />
33                      <ColorMapEntry color="#bd0026" quantity="5000" label="5000" />
34                  </ColorMap>
35              </RasterSymbolizer>
36          </Rule>
37      </FeatureTypeStyle>
38  </UserStyle>
39 </NamedLayer>
40 </StyledLayerDescriptor>

```

Code 5 GeoServer: SLD Style: FeatureTypeStyle

From line 25 the `<ColorMap>` is defined, where the pixel value is set with quantity and a color is given for every pixel that has the same or a smaller value than in the entry. The first applying rule is used, hence the importance of the order. The labels will be shown in the legend, that can be seen in the screenshot of the Style Editor (Figure 9). The colors that are used are hexadecimal (HEX) colors generated with the online tool ColorBrewer (2018).

### OpenLayers

To style the visualization and be more flexible in how it should be presented, the OpenLayers library was used. To test the servers, a very simple page was created, it should have a pan and zoom function, as well as a legend and a scale but also an overview map to have a reference when the map is zoomed in. How this was done is explained in the following and the entire script can be found in the appendix.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Tiled WMS</title>
5     <link rel="stylesheet" href="https://openlayers.org/en/v4.6.5/
      css/ol.css" type="text/css">
6     <!-- The line below is only needed for old environments like
      Internet Explorer and Android 4.x -->
7     <script src="https://cdn.polyfill.io/v2/polyfill.min.js?
      features=requestAnimationFrame,
      Element.prototype.classList,URL"></script>
8     <script src="https://openlayers.org/en/v4.6.5/build/ol.js"></script>
```

*Code 6 GeoServer: OpenLayers library*

In the beginning, the JavaScript library must be included, which is loaded directly from openlayers.org (line 5). For older browsers and other environments another script should be included (line 7). A title to the page is given in line 4.

```
9 <style>
10   .ol-custom-overviewmap,
11   .ol-custom-overviewmap.ol-uncollapsible {
12     bottom: auto;
13     left: auto;
14     right: 0;
15     top: 0;
16   }
17
18   .ol-custom-overviewmap:not(.ol-collapsed) {
19     border: 1px solid black;
20   }
21
22   .ol-custom-overviewmap .ol-overviewmap-map {
23     border: none;
24     width: 300px;
25   }
26
27   .ol-custom-overviewmap .ol-overviewmap-box {
28     border: 2px solid red;
29   }
30
31   .ol-custom-overviewmap:not(.ol-collapsed) button{
32     bottom: auto;
33     left: auto;
34     right: 1px;
35     top: 1px;
36   }
37
38 </style>
```

*Code 7 GeoServer: OpenLayers overview map style*

In the style part in the header of the script, the overview map is styled. The overview map is adapted from the OpenLayers Custom Overview Map Example (OpenLayers, 2018). Here the button is styled in all use stages, e.g. collapsed or not collapsed. The overview is set to be in the top right corner of the page (line 12-15). In an open form it has a solid black border (line 18, 19), with a width of 300 pixel (line 24). The extent of the actual map is shown as a red box on top of the overview map (line 27-28). And in the end the button to collapse or open the overview is defined (line 31-35).

```
42 <div id="map" class="map"></div>
43   <div id="legend">
44     
45   </div>
```

*Code 8 GeoServer: OpenLayers map element*

The map is contained in the `<div>` element within the `<body>` of the HTML document (line 42). The legend is also within a `<div>` element and has the identifier “legend” (line 43). It is called through a `GetLegendGraphic` request via the URL (line 44, split up for readability). The URL consist of the server access, followed by the request type, the version 1.0.0 and the format in which the legend should be returned, as well as the width and height. Afterwards the layer is defined for which the legend should be created, and a direction of the display is also added.

```
47   var overviewMapControl = new ol.control.OverviewMap({
48
49     className: 'ol-overviewmap ol-custom-overviewmap',
50     layers: [
51       new ol.layer.Tile({
52         source: new ol.source.OSM()
53       })
54     ],
55     collapseLabel: '\u00BB',
56     label: '\u00AB',
57     collapsed: false
58   });
```

*Code 9 GeoServer: OpenLayers overview script*

In code snippet 9 the `overviewMapControl` object is defined. With the class names that relate to the style as explained in code snippet 7. Here a map is put into the collapsible overview, this is done by adding a new layer (line 50, 51) and setting the source to Open Street Map, as this is where the map is coming from

(line 52). Line 55 and 56 define the sign of the collapse button, for the collapsed and opened form and it is set to be opened when loading the page (line 57).

```
61     var layers = [  
62         new ol.layer.Tile({  
63             source: new ol.source.OSM()  
64         }),  
65         new ol.layer.Tile({  
66             preload: Infinity,  
67             source: new ol.source.TileWMS({  
68                 url: 'http://localhost:8080/geoserver/master/wms',  
69                 params: {'LAYERS': 'master:Pop_correct10'},  
70                 serverType: 'geoserver'  
71             })  
72         })  
73     ];
```

*Code 10 GeoServer: OpenLayers - map layers*

In the next part, the layers are defined. The base map is the same as in the overview map (line 62, 63). Afterwards the new WMS layer is added as a tile layer (line 65) with a preload, so the tiles already preload when zooming in (line 66). The source is set to `ol.source.TileWMS` and then clarified with the URL pointing to the server (line 68). Afterwards the layer is defined in line 69 and the server set to 'geoserver' in line 70.

```
74     var map = new ol.Map({  
75         controls: ol.control.defaults().extend([  
76             overviewMapControl,  
77             new ol.control.ScaleLine({  
78                 units: 'metric'  
79             })  
80         ]),  
81         layers: layers,  
82         target: 'map',  
83         view: new ol.View({  
84             center: [2269873, 5087648],  
85             zoom: 2  
86         })  
87     })  
88     });
```

*Code 11 GeoServer: OpenLayers map script*

After the layers are defined, the map is created in line 74. In the beginning all controls are added to the map object (line 65-78), where the units of the scale are defined in line 78. The layers that the map is made of are added in line 81. With the target attribute the map is attached to the `<div>` object with the identifier 'map' (line 82). Afterwards, the view of the map is set with a defined center and zoom level (line 84, 85).

### 4.3.2 MapServer

As explained in chapter 3.5.2 MapServer is configured through a mapfile with the extension `.map`. The mapfile for the WMS is explained below and can also be found in the appendix.

#### Mapfile

The mapfile consist mainly of three elements, the map, the web and the layer object. The map object defines the server and includes all other objects in it. It starts with the `MAP` and is closed at the end of the mapfile with `END`. In the code 12 below, the beginning of the mapfile is shown with the required settings for the `MAP` object.

```
1  MAP
2      NAME "testMap"
3      IMAGETYPE png8
4      STATUS ON
5      SIZE 4320 1740
6      EXTENT -180 -90 180 90
7      UNITS DD
8      SHAPEPATH "../data"
9      IMAGECOLOR 189 201 225
10     # This is the output PROJECTION definition
11     PROJECTION
12         "init=epsg:4326"
13     END
```

Code 12 MapServer: Mapfile: map object

A name is given to the mapfile (line 2) and afterwards the image type set to Portable Network Graphic (PNG) 8, as this only supports 256 colors, hence smaller in size. With the status (line 4) the mapfile can easily be turned on and off, this is also used for every layer (see code 14, line 39). If the status is set to default, the layers are always sent to the client (MapServer, 2018b). The size is set to the size and the extent to the full extent of the original GeoTIFF, in decimal degrees (DD) (line 5-7). The shape path leads to the data folder in the root directory, where the GeoTIFF is stored. The image color in line 9 sets the background color of the map, this comes into account, when the mapfile preview is used. The color is defined in red, green, blue (RGB) color code and set to a light blue. The last part of the actual `MAP` object is the projection, which is defined as an object and therefore also has an `END` (line 11-13). The projection in the `MAP` object is the projection of the output, as opposed to the projection the original data has.



```

15  WEB
16      TEMPLATE 'test1.html'
17      IMAGEPATH "/ms4w/tmp/ms_tmp/"
18      IMAGEURL "/ms_tmp/"
19      METADATA
20          "wms_title"          "WMS Demo Server for MapServer"
21          "wms_onlineresource" "http://127.0.0.1/cgi-
22              bin/mapserv.exe?map=wms.map&"
23          "wms_srs"            "EPSG:4326"
24          "wms_enable_request" "*"
25      END
26  END # WEB

```

*Code 13 MapServer: Mapfile: WEB object*

To make the interactive and to configure the WMS a `WEB` object is needed. It is placed within the `MAP` object and ends with an `END` statement. As explained in chapter 3.5.2 MapServer uses templating and the `WEB` object links to the HTML template (line 16), which will be explained later. To store temporary files and images a path to a writable directory and the URL that directs the browser to the path is set in line 17 and 18. For a WMS metadata is required. This includes a title and a URL which will be used to access the server (line 20-22). Additionally, the projection is defined again (line 23), in this case it is not mandatory, as the layer has the same projection as the top-level map projection, however it is recommended. The last part is the enabling of requests, in this case all requests are enabled with an asterisk.

```

28  LAYER
29      NAME "pop"
30      METADATA
31          "wms_title"          "World population"
32          "wms_srs"            "EPSG:4326"
33          "gml_include_items" "all"
34          "gml_featureid"      "ID"
35          "wms_enable_request" "*"
36      END # METADATA
37      TEMPLATE "layertmp.html" # required for GetFeatureInfo requests
38      TYPE raster
39      STATUS ON
40      DATA "Pop_correct10.tiff"
41      # PROJECTION objects within the LAYER object define the input
42      # projection--this is the native projection of your data.
43      PROJECTION
44          "init=epsg:4326"
45      END

```

*Code 14 MapServer: Mapfile: LAYER object*

The code snippet above shows the `LAYER` object. It starts with an individual name (line 29), that will also be used in a GetMap request. The metadata is similar to the metadata in the `WEB` object but additionally has some features that are required for GetFeatureInfo requests (line 33, 34, 37). The layer template is similar to a map template, which is explained further down, therefore and because it is not needed for the purpose of this project it is not explained again but can be found in the appendix. Afterwards, the type of

the layer is set to raster (line 38) and the status turned on to be able to return it to the browser (line 39). The data is defined in line 40 followed by the projection that the data has.

```

46     CLASS
47         NAME "0-100"
48         EXPRESSION ([pixel] >= 0 and [pixel] < 100)
49         STYLE
50             COLOR '#ffffb2'
51         END
52     END
...
93 END # LAYER

```

Code 15 MapServer: Mapfile: CLASS object

The layer style is organized in `CLASS` objects. These classes have a name (line 47) which will be shown in a legend. A definition of the class is given with pixel values (line 48) and followed by a style object defining the color (line 49-51). The colors are the same as used for GeoServer and are also generated with ColorBrewer (2018). Here the order of the classes is important, as the first matching expression will be rendered. After all classes are described the `LAYER` object is closed in line 93.

```

95 LEGEND
96     KEYSIZE 12 12
97     LABEL
98         TYPE BITMAP
99         SIZE MEDIUM
100        COLOR 0 0 89
101    END
102    STATUS ON
103 END # legend
104
105 SCALEBAR
106     IMAGECOLOR 255 255 255
107     LABEL
108         COLOR 0 0 0
109         SIZE TINY
110    END
111    STYLE 1
112    SIZE 100 2
113    COLOR 0 0 0
114    UNITS KILOMETERS
115    INTERVALS 2
116    TRANSPARENT FALSE
117    STATUS ON
118 END # Scalebar
119 END # MAP

```

Code 16 MapServer: Mapfile: LEGEND and SCALEBAR objects

Also included in the `MAP` object are the `LEGEND` (line 95-103) and `SCALEBAR` (line 105-118) objects. Here the sizes are defined in line 96 and 112. The labels are defined in their size and color and type in lines 97-101 and 107-110. To be able to access the legend and the scalebar, their status is turned on (line 102, 117).

The scalebar is also customized with units set to kilometers (line 114) with two intervals (line 115) and not made transparent (116). Both objects are closed again in line 103 and 118. In the end the entire `MAP` object, hence `mapfile` is also closed with the `END` statement (line 119).

### HTML template

For generating an interactive preview, an HTML file for templating must be created. The template used for this project was adapted from an example template from MapServer (MapServer, 2018c). It is explained briefly in the following and can also be found in the appendix.

```

1      <!-- MapServer Template -->
...
10     <!-- The central form the application is based on. -->
11     <form namer="mapserv" method="GET" action="/cgi-bin/mapserv.exe">
12
13     <!-- CGI MapServer applications are server stateless in principle,
14          all information must be "stored" in the client. This includes
15          some basic settings as below.
16          The example is based on the pan and zoom test suite:
17          http://maps.dnr.state.mn.us/mapserver_demos/tests36/ -->
18     <input type="hidden" name="map" value="[map]">
19     <input type="hidden" name="imgext" value="[mapext]">
20     <input type="hidden" name="imgxy" value="[center]">
21     <input type="hidden" name="layer" value="pop">
22     <input type="hidden" name="mode" value="browse">

```

Code 17 MapServer: template

For the MapServer to recognize the HTML document as a template, it needs line 1 on the top of the document. In the beginning the document defines the CGI program to the location of the `mapserv.exe` file, with the name “`mapserv`” (line 11). The central part of templating is, that the information is stored in the client and `mapserv` replaces it with information. That allows information from the user to be send to `mapserv` and passed back again. In line 18 to 22 these variables are defined with a name and the value that should be returned.

```

24     <!-- A table for minimal page formatting. -->
25     <table border=0 cellpadding=5>
26     <tr>
27         <!-- First column: Map and scale bar -->
28         <td align=center>
29             <!-- The map -->
30             <input type="image" name="img" src="[img]"
31                 style="border:0;width:1000;height:600">
32             <br>
33             <!-- The scale bar-->
34             
35         </td>

```

Code 18 MapServer: template map and scalebar

As with the templating the preview is organized, the templating part is followed by some styling of the preview. It creates a table (line 25) with the map and the scale bar in the first column (line 28-35).

```

37  <!-- Second column: Zoom direction, Legend and Reference -->
38      <td valign=top>
39          <!-- Zoom direction -->
40          <b>Map Controls</b><br>
41          Set your zoom option:<br>
42          <select name="zoom" size="1">
43              <option value="2" [zoom_2_select]> Zoom in 2 times
44              <option value="1" [zoom_1_select]> Recenter Map
45              <option value="-2" [zoom_-2_select]> Zoom out 2 times
46          </select>
47          <br>

```

Code 19 MapServer: template zoom

The second column of the table consists of zoom functions shown above (code 19) and the legend, shown below (code 20). Here a drop-down menu is created for the map controls and value given to the operations. In this case there are two zoom options, one to zoom in two levels (line 42), and another one to zoom out two levels (line 44). Additionally, a re-center function is given in line 43.

```

48  <!-- Legend -->
49      <b>Legend</b><br>
50      <br><br><br><br>

```

Code 20 MapServer: template legend

### OpenLayers

To have the same interactive map as with GeoServer, OpenLayers was used to visualize the map via a web map server. As GeoServer and MapServer will be compared, the layouts of the pages are the same. Therefore, only the parts where the script differs, is explained below. The entire HTML document can be found in the appendix.

```

44  <div id="legend">
45      
46  </div>

```

Code 21 MapServer: OpenLayers legend element

The Legend is included as an image called with a `GetLegendGraphic` request via an URL (line 45, split up for readability). Here the URL directs the request to the CGI folder of the MapServer program and the mapfile. Then the service is set to WMS, the version to 1.1.1 and the correct layer for which the legend should be made stated. The request is described and the format in which the legend should be returned written in the end.

```
64  new ol.layer.Tile({
65      preload: Infinity,
66      source: new ol.source.TileWMS({
67          url: 'http://127.0.0.1/cgi-bin/mapserv.exe?
              map=/ms4w/apps/pop/htdocs/interact.map',
68          serverType: 'mapserver',
69          params: {'LAYERS': 'pop'}
70      })
71  })
    ];
```

*Code 22 MapServer: OpenLayers map object*

Here the layer is also added as a tile layer with the `ol.layer.Tile` command (line 64) and the preload of the tiles is also set like it was done in GeoServer (line 65). The source also starts in the same way, with `ol.source.TileWMS` (line 66). The URL is then set to the URL pointing to the mapfile (line 67, split up for readability). This is followed by the server type in line 68 and the definition of the layer, which in this case is 'pop' (line 69).

## 5 Comparison and Results

### 5.1 Visualization with Python

The initial idea of the project was to create an interactive visualization tool using the Python language, as this would fit with the simulation process. Therefore, different libraries were tested for their efficiency. Both libraries, Bokeh and Folium, had problems with the file size of the raster. While Folium could only handle very small rasters, Bokeh was a bit more powerful but also took a very long time to process the raster. For smaller files these two libraries would be good solutions due to their straight forward use and a good documentation. They are also not predominantly made for raster visualization but more for processing data and creating rasters out of it. Therefore, they are not suitable for the purpose of this project.

### 5.2 Map layout

After the first idea of visualizing the raster with a Python library did not work, two server solutions were implemented. The servers that were used are GeoServer and MapServer and they will be described and compared in the next stage. However, the map layout that was used and implemented using the OpenLayers library is the same for both. The data comes in the World Geodetic System (WGS 84) projection and is displayed as such.

The scale that is used is a standard graphic scale from OpenLayers and placed at the bottom left corner. Additionally, there is a legend showing the used color scheme with the population numbers. As this is the only variable that is displayed on the map there are no other map symbols.

The data of the simulations is quantitative data, the with population numbers for each cell, therefore a color scheme fitting quantitative data was used. As there is no emphasis in the mid-range or the critical values at the end of the range, hence making it suitable for a diverging scheme, a sequential color scheme was used. With the conventional use of light colors for low values and dark colors for higher values the importance of urban areas is enhanced. They are shown in a dark red. The color codes were created with ColorBrewer (2018), where a sequential color scheme with six classes was selected with a multi-hue. This means that not only the lightness of one color is changed but also the hue. Low values were represented in a bright yellow and high values in a dark red.

To meet the challenge of representing the entire world, a zoom function was implemented. This makes it possible to zoom into places of interest, e.g. to identify the urban extents of specific cities. The

accompanied loss of information, such as an overall location, is addressed with an overview map, which shows the extend of the current view on a world map.

### 5.3 Comparison of GeoServer with MapServer

The second attempt to interactively visualize the raster was through a map server. The two servers GeoServer and MapServer were tested against each other. In the following similarities and differences are drawn. The comparison happens on two levels. One on a soft and more subjective level where the ease of use is compared. Here, especially the set-up of the server and a first preview will be compared. On a more objective level, the performances are compared. This is done by using the Google Chrome Developer Tools (DevTools).

There are no hard facts that would exclude one of the servers from the beginning. They both implement most used open web services following the OGC standards. In this project, only the WMS was used, and this is supported by both servers. Both are also very well documented and examples and tutorials can be found for the two (GeoServer, 2018b; MapServer, Nacionales, & McKenna, 2017). As GeoServer is supported, and MapServer even part of OSGeo, they can be seen as steady services, as they are officially supported. This is good for a sustainable solution, that will also be supported in the future.

General differences can be seen in the following table (3) and are explained below.

*Table 3 Comparison of GeoServer and MapServer*

	GeoServer	MapServer
<b>Administration</b>	Web interface	Mapfile
<b>Services</b>	One OWS for all users	One mapfile for one server
<b>Styling</b>	Separate, predefined SLD file	Happens in the mapfile
<b>Loading of raster</b>	In web administration tool as layer	Define as layer in mapfile
<b>Preview</b>	<ul style="list-style-type: none"> <li>- Simple OpenLayers preview in web administration</li> <li>- little customizable options</li> <li>- already included zoom</li> </ul>	<ul style="list-style-type: none"> <li>- mapfile alone only static preview without zoom</li> <li>- interaction through html template file</li> <li>- with html file customizable</li> </ul>

GeoServer has a web interface administration, where the setup of the Open Web Server (OWS) is organized for all users and workspaces. However, differences can be made when global settings are disabled. These settings define how the OWS work, if the global settings are disabled only a number of selected layers are visible to the client. This can be important when having multiple layers for different use cases and users.

MapServer on the other hand handles the administration via a mapfile, where one mapfile defines one server. There is no web administration that has every setting listed and the user chooses between the differences via a drop-down list or ticking boxes as it is the case with GeoServer. This makes the start to use MapServer more complicated compared to GeoServer. The mapfile must be set up in a specific way, which can be copied to some extent from tutorials MapServer supplies. But it has a steeper learning curve to begin with.

When it comes to the styling of layers, the two servers are also different. GeoServer uses a style-editor embedded in the web administration page. With this editor, the appearance of the geospatial data can be changed using different formats. However, the default without using extensions is the Styled Layer Descriptor (SLD). These styles are saved and can be applied to the layers. MapServer uses different classes within the layer object of the mapfile for styling. When several layers should have the same style, it is more convenient to achieve in GeoServer, as the predefined style can simply be selected from a drop-down menu. In MapServer the classes and style objects can be copied and pasted under the new layer object. This is also easy, but can lead to errors, as copy mistakes can happen.

The process of adding data to the server is also contrasting. In GeoServer, this is again achieved through the web administration page and in MapServer through the mapfile. In both servers, the data has to be added into the root data directory of the server. In GeoServer that is the `geoserver/data_dir/data` folder as this is the root GeoServer data directory, which can be accessed through the web administration page. In MapServer it is located in the `ms4w/apps` folder. If added to the directory of GeoServer, it must be set up in the web administration page. A new folder can be added as a workspace and data is published as layers in a new data store. In MapServer on the other hand, it only has to be added to the data folder in the root directory and then added as a new layer object in the mapfile. Once the concept of a mapfile is understood, this is a faster way to publish a static map.



However, the next step is to preview the layer, this is easy in GeoServer in the Layer Preview, where the layer can be viewed with an OpenLayers format by simply clicking on the layer as shown in the figure (11) below.



Figure 11 GeoServer web administration: Layer Preview

Figure 12 shows a screenshot of such a preview, zoomed into Cairo. It can be seen, that there is a zoom function using buttons, but also zooming with a mouse scroll is possible. Additionally, a scale is provided in the bottom left corner and the coordinates of the mouse hover in the bottom right corner. When a pixel is clicked, the value of it is shown below the preview. This is all automatically provided by GeoServer using OpenLayers.

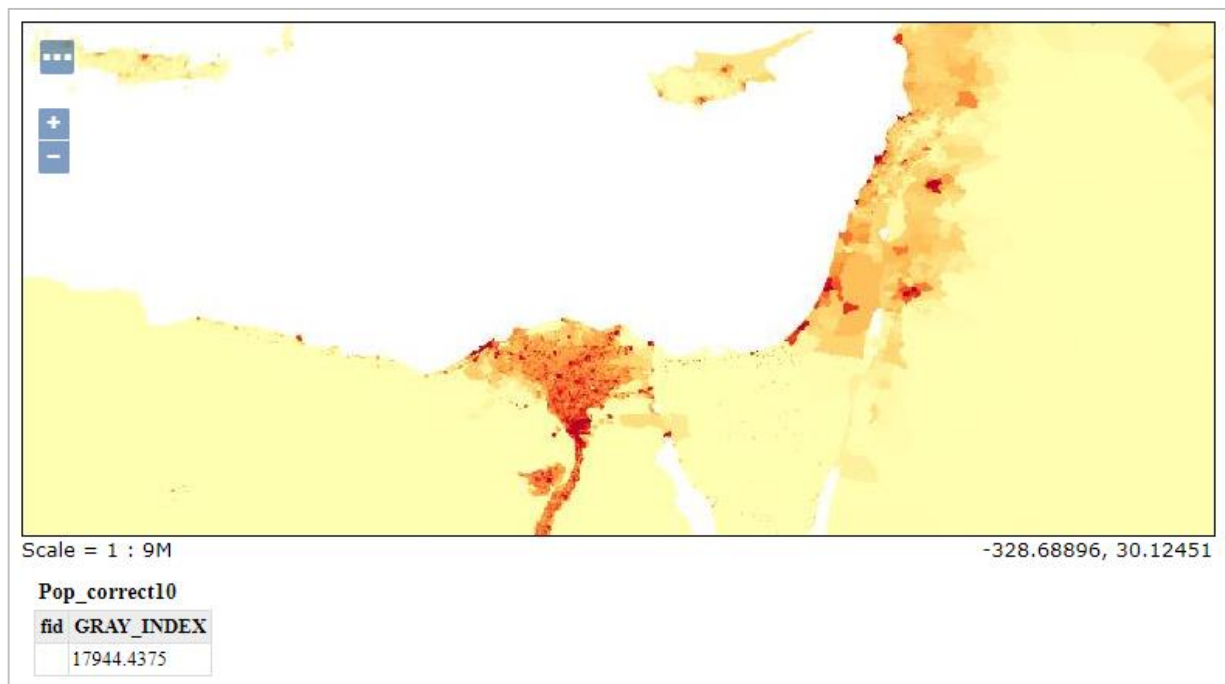


Figure 12 GeoServer Preview

MapServer also uses OpenLayers as a preview. However, this is more difficult when an interactive preview is wanted. The static map can be shown in a browser using the URL to the mapfile and setting the mode to map:

```
http://localhost/cgi-bin/mapserv.exe?map=/ms4w/apps/pop/htdocs/interact.map&layer=states&mode=map
```

To make it more interactive, the mode must be set to browse and a html file must be created, as explained in chapter 4.3.2. The result can be seen in the figure (13) below.

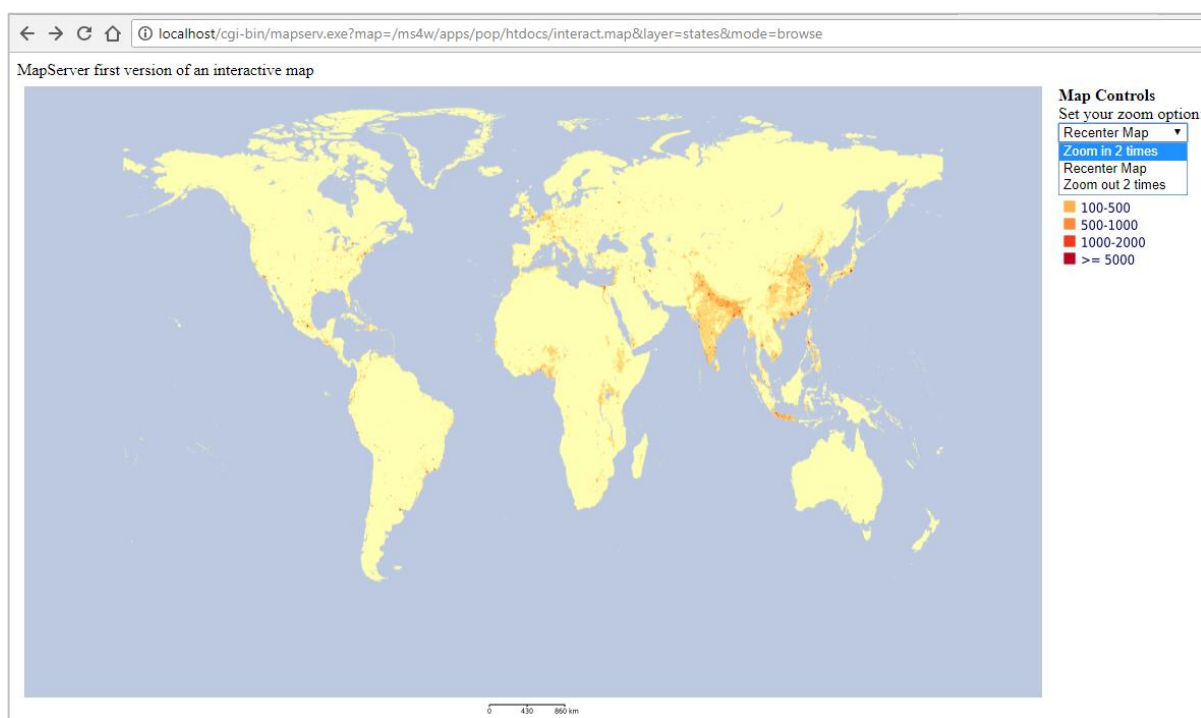


Figure 13 MapServer Preview

The advantages of GeoServer are in the ready to use preview, that works smoothly without further settings. However, the interactive browse mode from MapServer can be modified to fit the individual needs of the viewer. Once the html file is set up and the concept is understood, it can be customized to some degree. Nevertheless, there are some disadvantages with the interactivity from MapServer. The pan and zoom function, that work intuitively in GeoServer preview, can here only be controlled with a drop-down menu and clicking on the map, as shown in figure 13 above. This is to some extent inconvenient, as the zoom level must be chosen and if the user wants to pan, the recenter function must be chosen before clicking on the map again, but objectively the same pan and zoom results can be achieved. Once again,

MapServer has more set ups and is less intuitively to use. In the long run however, it can be more effectively used than GeoServer when fully understood. To conclude the comparison on a user-friendly level, GeoServer has more advantages when it comes to inexperienced users, as it is easy to use even without previous knowledge. MapServer has a steeper learning curve and a number of concepts must be understood before it can be used. Additionally, parameters can easily be overseen, because everything must be set up by the user. After the first obstacles, MapServer can be more customizable in the preview and adding of new layers is also faster than in GeoServer.

Table 4 Soft-level comparison GeoServer vs. MapServer

	GeoServer	MapServer
<b>user friendly</b>	+ easy understandable web administration + good for beginners	- more complicated mapfile - steep learning curve for beginners
<b>Styling</b>	+ easy applicable for new layers, but extra SLD	+ directly in mapfile
<b>Loading of raster</b>	- set up in stores and layers	+ simple, in mapfile
<b>Preview</b>	+ ready to use + working zoom - not customizable	- extra templating needed - zoom not intuitively + little customizable

The table above summarizes the pros and cons of both servers. It can be seen, that in most cases GeoServer is easier to use and faster to understand. However, MapServer has some advantages in controlling the server within the mapfile, rather than multiple windows and check boxes that need to be checked in GeoServer. Concluding, GeoServer is the better choice for beginners and users, that prefer administrative web interfaces over written code files.

Besides these soft levels of individual use and preferences, performance differences must be compared too. The DevTools outcome was evaluated following the guidance and documentation from (Basques & Chrome DevTools, 2018a, 2018b).

To prevent Google Chrome extensions to add noise and distractions to the result, both servers were tested in the incognito mode. To have a clear image of a first-time user, the cache is disabled, as shown in the screenshot below, which shows the network performance from GeoServer.

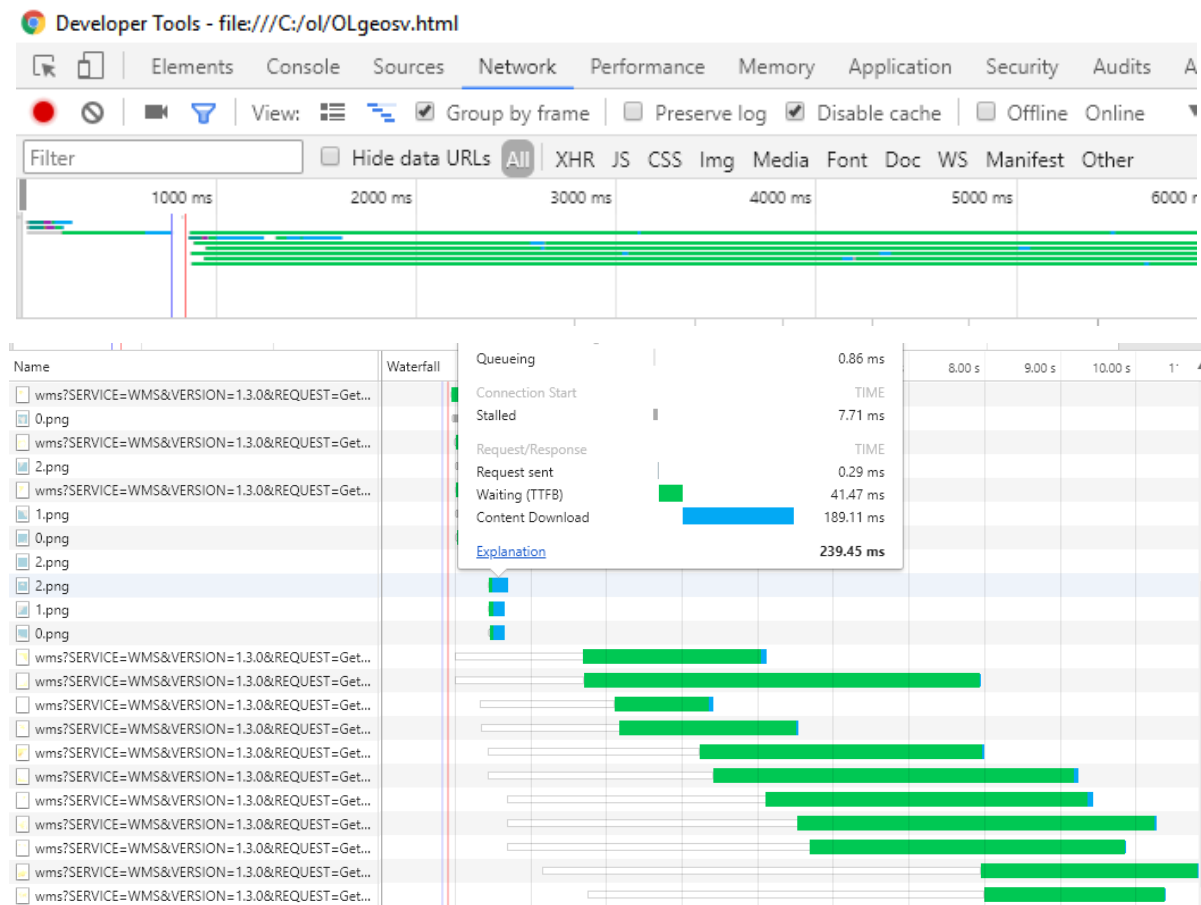


Figure 14 GeoServer Network Performance

The screenshot shows the time it takes to request and then respond from the server, for each item of the page. The waterfall diagram on the right shows the loading times split up into the subtasks. It can be seen, that the tiles from GeoServer have a longer waiting time to first byte (TTFB) shown in green and a much shorter time for downloading the content, shown in blue. Especially, when compared to the tiles requested from OpenStreetMap (OSM), for the base map, where it is the other way around. This means that the request spends a longer time waiting for GeoServer to receive the first byte than for OSM. The blue bar indicates a longer time for the request for OSM than GeoServer to be downloaded.

A similar but less extreme event happens with MapServer, this is shown on the screenshot below.

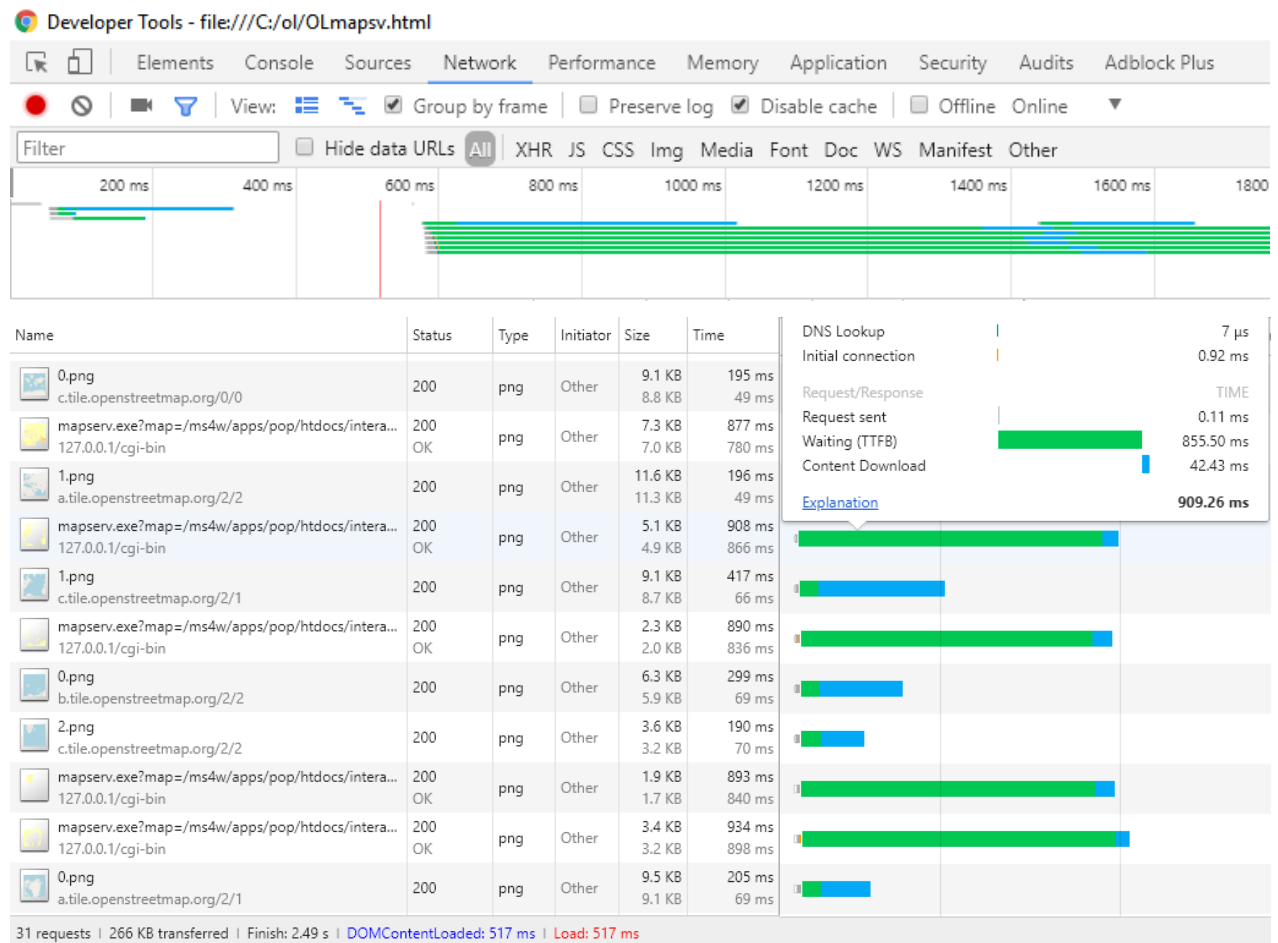


Figure 15 MapServer Network Performance

At the bottom of the screen the finishing time can be seen. In this case it took 2.49 seconds for the MapServer page to be fully loaded. It also shows that there are 31 requests, with one tile being one request, with a total size of 266 KB transferred. The DOMContentLoaded time states the time it takes for the initial HTML document to completely load, without any images. The load time is the time that it takes to load the full page in terms of the HTML document and images but not the WMS requested tiles. Besides being stated at the bottom of the screen, the DOMContentLoaded (blue) and load (red) event are shown with lines in the overview at the top. Especially in the screenshot from GeoServer, they are separated from each other, as the events happen with a little time difference. As these measurements vary every time the page is reloaded, they were written down and an average taken for both GeoServer and MapServer. The results can be seen in Table (5) below.

Table 5 Loading times GeoServer vs. MapServer

GeoServer			MapServer		
Finish (in s)	DOMContentLoaded (in ms)	Load (in ms)	Finish (in s)	DOMContentLoaded (in ms)	Load (in ms)
4,98	432	434	2,45	651	652
5,20	669	718	2,19	362	366
5,08	374	378	2,37	742	743
5,42	361	363	2,52	754	755
4,76	384	389	2,37	608	610
5,26	733	736	2,22	421	422
4,82	342	343	2,27	437	442
5,05	346	347	2,22	455	456
4,94	611	615	2,21	394	396
5,31	621	623	2,14	477	478
Average:					
5,08	487,30	494,60	2,30	530,10	532,00

Already after ten trials, it is noticeable that the time for the Document Object Model (DOM) plus the images to load take longer with MapServer. The whole page with all tiles loaded on the other hand takes twice as long with GeoServer than with MapServer. But this is what the user will notice. The page is already loaded and can be used, but the whole picture is not there yet, which can appear as not finished for the user. Even though it takes longer for Mapfile to load the initial parts, it appears faster to the user and is therefore considered the better choice, when it comes to network performances.

Another important part is the performance during the use. This is determined by the speed of loading tiles when zooming in and out, as well as panning. To be able to compare those, the Chrome DevTools were used again. This time the performance as opposed to the network was analyzed.

To be able to compare these soft use values, a standard procedure was implemented, and the results compared. The page was fully loaded before the recording of the performance with DevTools was started. Then, it was zoomed into Cairo, as it can easily be seen without zoom. The mouse will stay on the same spot and then zooming in, using the mouse scroll, until a 50-kilometer scale is reached. In the performance window of DevTools screenshots were captured to be able to tell when the desired zoom level was reached and when the pixels were fully loaded.

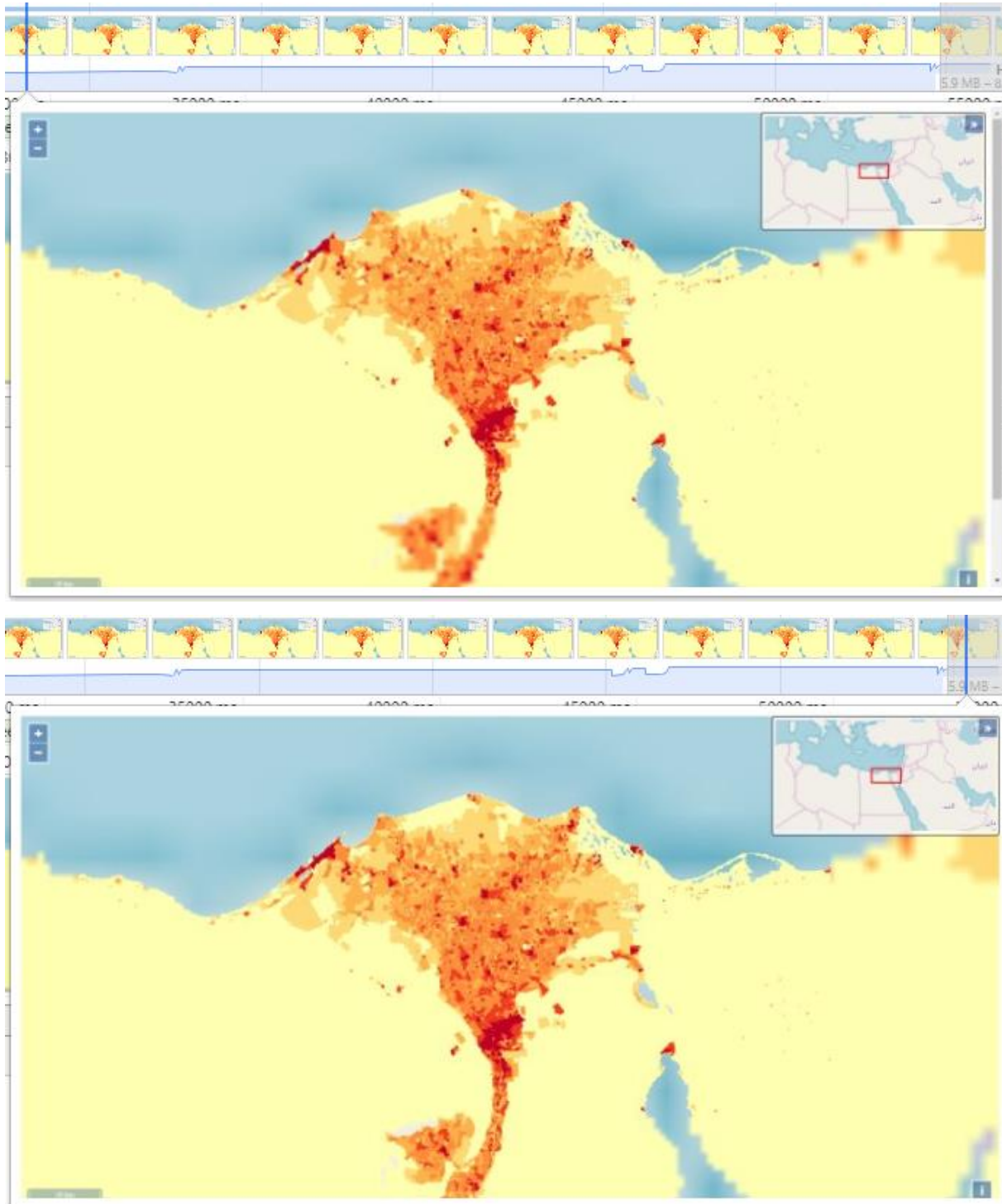


Figure 16 Zoom to Cairo and Faiyum Oasis

As the center of the map loads faster than the outsides, the south of Cairo, where the Faiyum Oasis alongside the Nile is, was the breakpoint. The time was measured until the oasis was fully loaded.

The loading time of the base map was ignored in this case as the focus is on the raster itself. However, it can be seen in the screenshots above, that the base map from OSM did not load in the time the raster loaded.

*Table 6 Time after zoom comparison*

MapServer	GeoServer
5.85 s	14.12 s
1.06 s	15.92 s
1.36 s	28.24 s
6.02 s	20,92 s
11.68 s	50.97 s
<b>Average:</b>	
5,19 s	26,03 s

It can be seen that the loading times after zooming vary. Nevertheless, it is clear that there is a difference between the two servers. While the average time after the zoom using MapServer is approximately five seconds, GeoServer has an average of 26 seconds. Which leads to better performance for MapServer, in both loading the page from the start as well as in using it.



## 6 Conclusion

In this chapter conclusions are drawn, based on the research questions in chapter 1.2:

How can an interactive tool improve the visualization of population simulations and what standards need to be met?

1. What are the visualization conventions that population simulations should follow?
2. Which tools already exist and what can they do?
3. How do these tools compare?
4. Can these tools be adapted for further development?

The improvements of an extra visualization tool over a standard GIS software are clearly in the presentation and publishing functions that come with a sperate tool. Even though the styling and interaction in QGIS are fast and easy to handle, they are not suitable for properly presenting the results. A solution that has a publishing service and can be embedded within a web page, makes the user and viewer of the raster independent from possible installations of programs.

Visualization conventions are not only an important part of a map making process but are also relevant in understanding maps. Conventions can help users to understand a map and then interpret the displayed information faster. This applies to population simulations in the same way, as the numbers are represented on a map. Conventional practices concerning color were implemented in the visualization process. However, a more profound examination and analyzation of the data could improve the outcome. An exploration of urbanization and population density in urban areas could lead to a more precise classification. So far, only the type of data and a fitting color scheme was considered. The break points in the color scheme were chosen based on previous visualizations of the same data (Keßler & Marcotullio, 2017). Here, further study could highly improve the usability of the visualizations for population simulations.

The focus of this project was not only on visualizing population numbers but also on showing large datasets with an interaction. This can also be adapted for other large datasets. Some Python libraries exist, that can visualize very those, however the focus on these libraries is the rasterization itself, i.e. turning large datasets into rasters. A customizable visualization tool to only visualize rasters and to be able to style them is harder to find, but the libraries that were found could be adapted to do so, which was done in chapter 4.1. But thereupon the raster size exceeded their capabilities, hence making them unsuitable.

Compared to a Python library, where adjustments would be made within the Python code, a server solution has more configurations. The first composition of a server is more time consuming and requires more settings. However, the solution of a web server has some advantages. Especially when considering a bigger project behind the simulations with possible publishing aims. Servers make it easy to publish the data and also make it available over web pages. With the implementation of an OpenLayers map, the basis was already made.

The disadvantages of a web server were weighed against the advantages, and the possibilities for further development were favored. So that the two servers GeoServer and MapServer were tested. Both servers were not as fast, as the view in QGIS but considering the benefits of a server, this was accepted. Performance wise, MapServer was found to be the faster solution in loading the raster as well as in interaction with it. GeoServer however is a good choice for beginners, as the setup in the web administration interface is easy to understand. However, when other datasets are used the result could be different. A common issue with spatial data are reprojections, which were not tested with the servers and could affect the performance.

Further development can be done in many ways, as the outcome of this thesis is far from being a fully deployed tool. The selection of colors and especially the breakpoints, needs further research. Additionally, the web page can be optimized in various forms. This could include multiple layers to be able to show simulations for future years and a function to click on the map and receive information about the exact population number in that cell would be advantages. Besides these layout changes, the servers are not yet online accessible, which should be changed in the future. This is accompanied by security issues, that need to be addressed in the server configurations as well.

Finally, a solution was found that is capable of the raster sizes and lets the user interact with the output and styles are easily applicable. Nonetheless, further research must be done in various directions to be able to use it for presentation and publishing purposes.

## 7 Bibliography

- Anaconda. (2018). Home - Anaconda. Retrieved June 1, 2018, from <https://www.anaconda.com/>
- Basques, K., & Chrome DevTools. (2018a). Get Started with Analyzing Network Performance in Chrome DevTools | Tools for Web Developers | Google Developers. Retrieved May 23, 2018, from <https://developers.google.com/web/tools/chrome-devtools/network-performance/>
- Basques, K., & Chrome DevTools. (2018b). Get Started With Analyzing Runtime Performance | Tools for Web Developers | Google Developers. Retrieved May 23, 2018, from <https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/>
- Bertin, J. (1974). *Graphische Semilogie. Diagramme, Netze, Karten*. (G. Jensch, D. Schade, & 2010, Trans.) (2nd french). Berlin, Boston: De Gruyter.
- Bertin, J. (1981). *Graphics and Graphic Information Processing*. (W. J. Berg & P. Scott, Trans.). Berlin, New York: De Gruyter. <https://doi.org/10.1515/9783110854688>
- Bokeh. (2018a). Defining Key Concepts — Bokeh 0.12.16 documentation. Retrieved May 31, 2018, from [http://bokeh.pydata.org/en/0.12.16/docs/user\\_guide/concepts.html](http://bokeh.pydata.org/en/0.12.16/docs/user_guide/concepts.html)
- Bokeh. (2018b). Welcome to Bokeh — Bokeh 0.12.13 documentation. Retrieved January 29, 2018, from <https://bokeh.pydata.org/en/latest/>
- Boundless. (2018). Boundless : Introduction to GeoServer : 3. Overview. Retrieved May 25, 2018, from <http://workshops.boundlessgeo.com/geoserver-intro/overview/index.html>
- Brewer, C. A. (1994). Color Use Guidelines for Mapping and Visualization. *Modern Cartography Series*, 2(C), 123–147. <https://doi.org/10.1016/B978-0-08-042415-6.50014-4>
- CIESIN - Center for International Earth Science Information Network, Columbia University, & International Food Policy Research Institute. (2011). Global Rural-Urban Mapping Project, Version 1 (GRUMPv1): Population Count Grid. Palisades, NY: NASA Socioeconomic Data and Applications Center (SEDAC). Retrieved from <http://dx.doi.org/10.7927/H4VT1Q1H>
- ColorBrewer. (2018). ColorBrewer 2.0: Color Advice for Maps. Retrieved June 3, 2018, from <http://colorbrewer2.org/#type=sequential&scheme=YlOrRd&n=5>
- data.europa.eu. (2018). Data | European Union Open Data Portal. Retrieved May 29, 2018, from <http://data.europa.eu/euodp/en/data/>
- data.gov. (2018). Datasets - Data.gov. Retrieved May 29, 2018, from <https://catalog.data.gov/dataset>
- Deoliveria, J. (2018). GeoServer: A Geospatial Server for Everyone - Geospatial World. Retrieved May 25, 2018, from <https://www.geospatialworld.net/article/geoserver-a-geospatial-server-for-everyone/>
- FLOW. (2018). Retrieved April 30, 2018, from <http://www.strategi.aau.dk/Forskning/Tvaervidenskabelige+forskningsprojekter/FLOW/>
- Folium. (2018). Folium. Python Data. Leaflet.js Maps. Retrieved March 9, 2018, from <https://folium.readthedocs.io/en/latest/>
- Friendly, M. (2008). A Brief History of Data Visualization. In C. Chen, W. Härdle, & A. Unwin (Eds.), *Handbook of Data Visualization* (pp. 15–56). Berlin, Heidelberg: Springer. <https://doi.org/10.1007/978-3-540-33037-0>

- Garlandini, S., & Fabrikant, S. I. (2009). Evaluating the Effectiveness and Efficiency of Visual Variables for Geographic Information Visualization. In *Spatial Information Theory* (pp. 195–211). Berlin, Heidelberg: Springer. Retrieved from <https://link.springer-com.zorac.aub.aau.dk/content/pdf/10.1007%2F978-3-642-03832-7.pdf>
- GDAL. (2018). GDAL: GDAL - Geospatial Data Abstraction Library. Retrieved April 6, 2018, from <http://www.gdal.org/>
- GeoServer. (2018a). GeoServer. Retrieved May 5, 2018, from <http://geoserver.org/about/>
- GeoServer. (2018b). Introduction to SLD — GeoServer 2.14.x User Manual. Retrieved May 5, 2018, from <http://docs.geoserver.org/latest/en/user/styling/sld/introduction.html>
- GeoServer. (2018c). Virtual Services — GeoServer 2.14.x User Manual. Retrieved May 5, 2018, from <http://docs.geoserver.org/latest/en/user/configuration/virtual-services.html#virtual-services>
- GeoServer. (2018d). WCS settings — GeoServer 2.14.x User Manual. Retrieved June 3, 2018, from <http://docs.geoserver.org/latest/en/user/services/wcs/webadmin.html#service-metadata>
- GeoServer. (2018e). WMS settings — GeoServer 2.14.x User Manual. Retrieved June 3, 2018, from <http://docs.geoserver.org/latest/en/user/services/wms/webadmin.html>
- Harrower, M., & Brewer, C. A. (2003). ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps. *The Cartographic Journal*, 40(1), 27–37. <https://doi.org/10.1179/000870403235002042>
- Iacovella, S., & Youngblood, B. (2013). *GeoServer Beginner's Guide*. Birmingham: Packt Pub. Retrieved from <https://ebookcentral.proquest.com/lib/aalborguniv-ebooks/reader.action?docID=1108358&ppg=1>
- JetBrains. (2018). PyCharm: Python IDE for Professional Developers by JetBrains. Retrieved April 6, 2018, from <https://www.jetbrains.com/pycharm/>
- Kennedy, H., Hill, R. L., Aiello, G., & Allen, W. (2016). The work that visualisation conventions do. *Information, Communication and Society*, 19(6), 715–735. <https://doi.org/10.1080/1369118X.2016.1153126>
- Keßler, C., & Marcotullio, P. J. (2017). A Geosimulation for the Future Spatial Distribution of the Global Population. In *AGILE 2017*. Wageningen, Neatherlands.
- Kraak, M.-J., & Ormeling, F. (2010). *Cartography: Visualization of Spatial Data*. Pearson Education (3rd ed.). London, New York: Routledge. <https://doi.org/10.1111/cag.12149>
- Kropla, B. (2005). *Beginning MapServer. Open Source GIS Development*. New York: Springer. Retrieved from <http://www.springeronline.com>.
- Langtangen, H. P. (2009). *Python Scripting for Computational Science* (3rd ed.). Berlin, Heidelberg: Springer. <https://doi.org/10.1007/978-3-662-05450-5>
- Leaflet. (2018). Leaflet - a JavaScript library for interactive maps. Retrieved March 9, 2018, from <http://leafletjs.com/>
- MacEachren, A. M., & Kraak, M.-J. (2001). Research Challenges in Geovisualization. *Cartography and Geographic Information Science*, 28(1), 3–12. <https://doi.org/10.1559/152304001782173970doi.org/10.1559/152304001782173970>
- MapServer. (2018a). About — MapServer 7.0.7 documentation. Retrieved May 5, 2018, from

- <http://mapserver.org/about.html>
- MapServer. (2018b). LAYER — MapServer 7.0.7 documentation. Retrieved June 2, 2018, from <http://mapserver.org/mapfile/layer.html>
- MapServer. (2018c). Templating — MapServer 7.0.7 documentation. Retrieved June 2, 2018, from <http://mapserver.org/mapfile/template.html>
- MapServer, Nacionales, P. S., & McKenna, J. (2017). MapServer Tutorial — MapServer 7.0.7 documentation. Retrieved May 17, 2018, from <http://www.mapserver.org/tutorial/>
- McKenna, J., & Gateway Geomatics. (2018). MapServer for Windows (MS4W) README — MS4W 3.2.7 documentation. Retrieved May 13, 2018, from [https://ms4w.com/README\\_INSTALL.html#introduction](https://ms4w.com/README_INSTALL.html#introduction)
- Monmonier, M. (1996). *How to Lie with Maps* (2nd ed.). Chicago: University of Chicago Press. <https://doi.org/10.2307/2685420>
- Monmonier, M. (2005). Lying with Maps. *Statistical Science*, 20(3), 215–222. <https://doi.org/10.1214/088342305000000241>
- Moreland, K. (2009). Diverging Color Maps for Scientific Visualization. In G. Bebis & et al. (Eds.), *Advances in Visual Computing. ISVC 2009, Part II* (pp. 92–103). Berlin, Heidelberg: Springer. [https://doi.org/10.1007/978-3-642-29066-4\\_11](https://doi.org/10.1007/978-3-642-29066-4_11)
- Mose, J., & Strüver, A. (2009). Diskursivität von Karten. In *Handbuch Diskurs und Raum. Theorien und Methoden für die Humangeographie sowie die sozial- und kulturwissenschaftliche Raumforschung* (pp. 315–325). Bielefeld: transcript.
- Nowell, L. T. (1997). *Graphical Encoding for Information Visualization: Using Icon Color, Shape, and Size To Convey Nominal and Quantitative Data*. Blacksburg, Virginia: Virginia Tech. Retrieved from <https://vtechworks.lib.vt.edu/bitstream/handle/10919/29663/ETD.PDF?sequence=1&isAllowed=y>
- NumPy. (2018). NumPy. Retrieved April 6, 2018, from <http://www.numpy.org/>
- OGC - Open Geospatial Consortium Inc. (2007). *Styled Layer Descriptor profile of the Web Map Service*. (M. Lupp, Ed.), *OGC Implementation Specification* (V 1.1.0, Vol. 0). Retrieved from <http://www.opengeospatial.org/standards/sld>
- OGC - Open Geospatial Consortium Inc. (2006). *OpenGIS Web Map Server Implementation Specification*. (J. de la Beaujardiere, Ed.), *OpenGIS Implementation Specification* (V 1.3.0). <https://doi.org/http://www.opengeospatial.org/standards/wms>
- OGC - Open Geospatial Consortium Inc. (2010a). *OpenGIS Web Feature Service 2.0 Interface Standard*. (P. A. Vretanos, Ed.), *OpenGIS Implementation Standard* (V 2.0.0).
- OGC - Open Geospatial Consortium Inc. (2010b). *OpenGIS Web Map Tile Service Implementation Standard*. (J. Maso, K. Pomakis, & N. Julia, Eds.), *OpenGIS Implementation Standard* (V 1.0.0). Retrieved from <http://www.opengeospatial.org/standards/wmts>
- OGC - Open Geospatial Consortium Inc. (2012). *OGC WCS 2.0 Interface Standard- Core: Corrigendum*. (P. Baumann, Ed.), *OGC Interface Standard* (V 2.0.1). Retrieved from <http://www.opengeospatial.org/standards/wcs%5Cnpapers2://publication/uuid/D303A640-AF41-432D-B72C-593E1BDCFF47>

- OGC - Open Geospatial Consortium Inc. (2016). *DGIWG – Web Feature Service 2.0 Profile Copyright*. (S. Strobel, D. Sarafinof, D. Wesloh, & P. Lacey, Eds.), *OGC Best Practice* (V 2.0). Retrieved from <http://www.opengeospatial.org/standards/wfs>
- OpenGeospatial. (2018). Welcome to the OGC | OGC. Retrieved May 4, 2018, from <http://www.opengeospatial.org/>
- OpenLayers.org. (2018). OpenLayers - Welcome. Retrieved June 1, 2018, from <http://openlayers.org/>
- OpenLayers. (2018). Custom Overview Map. Retrieved June 2, 2018, from <https://openlayers.org/en/latest/examples/overviewmap-custom.html>
- OSGeo. (2018a). About OSGeo - OSGeo. Retrieved April 6, 2018, from <https://www.osgeo.org/about/>
- OSGeo. (2018b). FAQGeneral – GDAL. What does OGR stand for? Retrieved May 21, 2018, from <https://trac.osgeo.org/gdal/wiki/FAQGeneral#WhatdoesOGRstandfor>
- OSGeo. (2018c). FAQGeneral – GDAL. Retrieved April 6, 2018, from <https://trac.osgeo.org/gdal/wiki/FAQGeneral#FAQ-General>
- Periscopic. (2018). Periscopic: Do good with data. Retrieved April 11, 2018, from <http://www.periscopic.com/>
- Python Software Foundation. (2018a). 26.6. timeit — Measure execution time of small code snippets — Python 2.7.15 documentation. Retrieved May 21, 2018, from <https://docs.python.org/2/library/timeit.html>
- Python Software Foundation. (2018b). General Python FAQ. Retrieved April 6, 2018, from <https://docs.python.org/3/faq/general.html#what-is-python>
- Python Software Foundation. (2018c). Python Software Foundation. Retrieved April 6, 2018, from <https://www.python.org/psf/>
- QGIS. (2018a). Discover QGIS. Retrieved April 17, 2018, from <https://qgis.org/en/site/about/index.html>
- QGIS. (2018b). PYQGIS Cookbook (QGIS 2.18) - Python Applications. Retrieved April 18, 2018, from [https://docs.qgis.org/2.18/en/docs/pyqgis\\_developer\\_cookbook/intro.html#running-custom-applications](https://docs.qgis.org/2.18/en/docs/pyqgis_developer_cookbook/intro.html#running-custom-applications)
- Snow, J. (1855). *On the Mode of Communication of Cholera* (2nd ed.). London: J. Churchill. Retrieved from <http://gallica.bnf.fr/ark:/12148/bpt6k856189z/f5.item>
- Spence, R. (2014). *Information Visualization* (3rd ed.). Cham: Springer International Publishing. <https://doi.org/10.1007/978-3-319-07341-5>
- Tufte, E. R. (2015). *The Visual Display of Quantitative Information* (2nd ed.). Cheshire, Connecticut: Graphics Press. Retrieved from <https://www.colorado.edu/geography/foote/maps/assign/reading/TufteCoversheet.pdf>
- United Nations, Department of Economic and Social Affairs, P. D. (2014). World Urbanization Prospects: The 2014 Revision. Retrieved April 9, 2018, from <https://esa.un.org/unpd/wup/>
- United Nations, Department of Economic and Social Affairs, P. D. (2015). World Population Prospects, the 2015 Revision. Retrieved May 5, 2018, from <https://esa.un.org/unpd/wpp/>
- United Nations, Department of Economic and Social Affairs, P. D. (2017). World Population Prospects:

- The 2017 Revision. Retrieved April 10, 2018, from <https://esa.un.org/unpd/wpp/>
- Van Der Walt, S., Colbert, C. S., & Varoquaux, G. (2011). The NumPy array: a structure for efficient numerical computation. *Computing in Science and Engineering*, 13(2), 22–30.  
<https://doi.org/10.1109/MCSE.2011.37>
- worldometers.info. (2018). World Population Clock: 7.6 Billion People (2018) - Worldometers. Retrieved May 27, 2018, from <http://www.worldometers.info/world-population/>
- worldpopulationhistory.org. (2018). World Population | An Interactive Experience - World Population. Retrieved May 27, 2018, from <http://worldpopulationhistory.org/map/2050/mercator/1/0/25/>
- worldpopulationreview.com. (2018). 2018 World Population by Country. Retrieved May 27, 2018, from <http://worldpopulationreview.com/>

## Appendix

1. Folium
2. Bokeh with timeit
3. GeoServer
  - 3.1. Raster Style – SLD
  - 3.2. GeoServer – OpenLayers
4. MapServer
  - 4.1. Mapfile interact.map
  - 4.2. Mapfile template test1.html
  - 4.3. Layer template layertmp.html
  - 4.4. MapServer – OpenLayers



## 1. Folium

```
1  import gdal
2  import numpy as np
3  import folium
4  from folium.plugins import ImageOverlay
5
6  print(folium.__version__)
7
8  src = gdal.Open('PopDK.tif', gdal.GA_Update)
9  band = src.GetRasterBand(1)
10 imarray = np.array(band.ReadAsArray())
11
12
13 map= folium.Map(zoom_start=3)
14
15 folium.plugins.ImageOverlay(
16     image=imarray,
17     bounds=[[-180, -60], [180, 85]],
18     colormap=lambda x: (1, 0, 0, x),
19     origin='lower',
20 ).add_to(map)
21
22 #Save html
23 map.save('folium_test.html')
```

## 2. Bokeh with timeit

```
1  from bokeh.plotting import figure, save, output_file
2  import gdal
3  import numpy as np
4  import timeit
5
6  def popDK():
7      src = gdal.Open('PopDK.tif', gdal.GA_Update)
8      band = src.GetRasterBand(1)
9      imarray = np.array(band.ReadAsArray())
10
11      output_file(r"C:/master/imageDK.html")
12
13      imarray = imarray[::-1]
14      p = figure(x_range=(-180,180), y_range=(-60,85))
15      p.image_rgba(image=[imarray], x=[-180], y=[-60], dw=[360], dh=[145],
16      dilate=False)
17      save(p)
18
19  ta = timeit.Timer(popDK).timeit(number=5)
20  print 'PopDK took ', ta, ' seconds'
21
22  def popEU():
23      src = gdal.Open('PopEU.tif', gdal.GA_Update)
24      band = src.GetRasterBand(1)
25      imarray = np.array(band.ReadAsArray())
26
27      output_file(r"C:/master/imageEU.html")
```

```
28
29     imarray = imarray[::-1]
30     p = figure(x_range=(-180,180), y_range=(-60,85))
31     p.image_rgba(image=[imarray], x=[-180], y=[-60], dw=[360], dh=[145],
32     dilate=False)
33     save(p)
34
35     tb = timeit.Timer(popEU).timeit(number=5)
36     print 'PopEU took ', tb, ' seconds'
37
38
39     def popAus2():
40         src = gdal.Open('PopAus2.tif', gdal.GA_Update)
41         band = src.GetRasterBand(1)
42         imarray = np.array(band.ReadAsArray())
42
43         output_file(r"C:/master/imageAus2.html")
44
45         imarray = imarray[::-1]
46         p = figure(x_range=(-180,180), y_range=(-60,85))
47         p.image_rgba(image=[imarray], x=[-180], y=[-60], dw=[360], dh=[145],
48         dilate=False)
49         save(p)
50
51     tc = timeit.Timer(popAus2).timeit(number=5)
52     print 'PopAus2 took ', tc, ' seconds'
53
54
55     def popAus():
56         src = gdal.Open('PopAus.tif', gdal.GA_Update)
57         band = src.GetRasterBand(1)
58         imarray = np.array(band.ReadAsArray())
59
60         output_file(r"C:/master/imageAus.html")
61
62         imarray = imarray[::-1]
63         p = figure(x_range=(-180,180), y_range=(-60,85))
64         p.image_rgba(image=[imarray], x=[-180], y=[-60], dw=[360], dh=[145],
65         dilate=False)
66         save(p)
67
68     td = timeit.Timer(popAus).timeit(number=5)
69     print 'PopAus took ', td, ' seconds'
```

## 3. GeoServer

### 3.1 Raster Style - SLD

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <StyledLayerDescriptor version="1.0.0"
3    xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
4    xmlns="http://www.opengis.net/sld"
5    xmlns:ogc="http://www.opengis.net/ogc"
6    xmlns:xlink="http://www.w3.org/1999/xlink"
7    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8    <!-- a Named Layer is the basic building block of an SLD document -->
9    <NamedLayer>
10      <Name>default_raster</Name>
11      <UserStyle>
12        <!-- Styles can have names, titles and abstracts -->
13        <Title>Default Raster</Title>
14        <Abstract>A sample style that draws a raster, good for displaying
15          imagery</Abstract>
16        <!-- FeatureTypeStyles describe how to render different features -->
17        <!-- A FeatureTypeStyle for rendering rasters -->
18        <FeatureTypeStyle>
19          <Rule>
20            <Name>rule1</Name>
21            <Title>Opaque Raster</Title>
22            <Abstract>A raster with 100% opacity</Abstract>
23            <RasterSymbolizer>
24              <Opacity>1.0</Opacity>
25              <ColorMap>
26                <ColorMapEntry color="#000000" quantity="-500" label="nodata"
27                  opacity="0.0" />
28                <ColorMapEntry color="#ffffffb2" quantity="0" label="0" />
29                <ColorMapEntry color="#fed976" quantity="100" label="100"/>
30                <ColorMapEntry color="#feb24c" quantity="500" label="500" />
31                <ColorMapEntry color="#fd8d3c" quantity="1000" label="1000" />
32                <ColorMapEntry color="#f03b20" quantity="2000" label="2000" />
33                <ColorMapEntry color="#bd0026" quantity="5000" label="5000" />
34              </ColorMap>
35            </RasterSymbolizer>
36          </Rule>
37        </FeatureTypeStyle>
38      </UserStyle>
39    </NamedLayer>
40  </StyledLayerDescriptor>
```

## 3.2 GeoServer – OpenLayers

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Tiled WMS</title>
5      <link rel="stylesheet" href="https://openlayers.org/en/v4.6.5/
6        css/ol.css" type="text/css">
7      <!-- The line below is only needed for old environments like
8        Internet Explorer and Android 4.x -->
9
10     <script src="https://cdn.polyfill.io/v2/polyfill.min.js?
11       features=requestAnimationFrame,
12       Element.prototype.classList,URL"></script>
13
14     <script src="https://openlayers.org/en/v4.6.5/build/ol.js"></script>
15
16     <style>
17     .ol-custom-overviewmap,
18     .ol-custom-overviewmap.ol-uncollapsible {
19       bottom: auto;
20       left: auto;
21       right: 0;
22       top: 0;
23     }
24
25     .ol-custom-overviewmap:not(.ol-collapsed) {
26       border: 1px solid black;
27     }
28
29     .ol-custom-overviewmap .ol-overviewmap-map {
30       border: none;
31       width: 300px;
32     }
33
34     .ol-custom-overviewmap .ol-overviewmap-box {
35       border: 2px solid red;
36     }
37
38     .ol-custom-overviewmap:not(.ol-collapsed) button{
39       bottom: auto;
40       left: auto;
41       right: 1px;
42       top: 1px;
43     }
44
45   </style>
46   </head>
47   <body>
48
49     <div id="map" class="map"></div>
50     <div id="legend">
51       
56
57     </div>
58     <script>
59     var overviewMapControl = new ol.control.OverviewMap({
60       className: 'ol-overviewmap ol-custom-overviewmap',
61       layers: [
```

```
51         new ol.layer.Tile({
52             source: new ol.source.OSM()
53         })
54     ],
55     collapseLabel: '\u00BB',
56     label: '\u00AB',
57     collapsed: false
58 });
59
60
61 var layers = [
62     new ol.layer.Tile({
63         source: new ol.source.OSM()
64     }),
65     new ol.layer.Tile({
66         preload: Infinity,
67         source: new ol.source.TileWMS({
68             url: 'http://localhost:8080/geoserver/master/wms',
69             params: {'LAYERS': 'master:Pop_correct10'},
70             serverType: 'geoserver'
71         })
72     })
73 ];
74 var map = new ol.Map({
75     controls: ol.control.defaults().extend([
76         overviewMapControl,
77         new ol.control.ScaleLine({
78             units: 'metric'
79         })
80     ]),
81     layers: layers,
82     target: 'map',
83     view: new ol.View({
84         center: [2269873, 5087648],
85         zoom: 2
86     })
87 });
88 });
89 </script>
90 </body>
91 </html>
```

## 4. MapServer

### 4.1 Mapfile interact.map

```
1  MAP
2      NAME "testMap"
3      IMAGETYPE png8
4      STATUS ON
5      SIZE 4320 1740
6      EXTENT -180 -90 180 90
7      UNITS DD
8      SHAPEPATH "../data"
9      IMAGECOLOR 189 201 225
10     # This is the output PROJECTION definition
11     PROJECTION
12         "init=epsg:4326"
13     END
14
15     WEB
16         TEMPLATE 'test1.html'
17         IMAGEPATH "/ms4w/tmp/ms_tmp/"
18         IMAGEURL "/ms_tmp/"
19         METADATA
20             "wms_title" "WMS Demo Server for MapServer"
21             "wms_onlineresource" "http://127.0.0.1/cgi-
22                 bin/mapserv.exe?map=wms.map&"
23             "wms_srs" "EPSG:4326"
24             "wms_enable_request" "*"
25         END
26     END # WEB
27
28     LAYER
29         NAME "pop"
30         METADATA
31             "wms_title" "World population"
32             "wms_srs" "EPSG:4326"
33             "gml_include_items" "all"
34             "gml_featureid" "ID"
35             "wms_enable_request" "*"
36         END # METADATA
37         TEMPLATE "layertmp.html" # required for GetFeatureInfo requests
38         TYPE raster
39         STATUS ON
40         DATA "Pop_correct10.tiff"
41         # PROJECTION objects within the LAYER object define the input
42         # projection--this is the native projection of your data.
43         PROJECTION
44             "init=epsg:4326"
45         END
46
47         CLASS
48             NAME "0-100"
49             EXPRESSION ([pixel] >= 0 and [pixel] < 100)
50             STYLE
51                 COLOR '#ffffb2'
52             END
53         END
54
55         CLASS
56             NAME "100-500"
57             EXPRESSION ([pixel] >= 100 and [pixel] < 500)
58             STYLE
```

```
58         COLOR '#fed976'
59     END
60 END
61
62 CLASS
63     NAME "100-500"
64     EXPRESSION ([pixel] >= 500 and [pixel] < 1000)
65     STYLE
66         COLOR '#feb24c'
67     END
68 END
69
70 CLASS
71     NAME "500-1000"
72     EXPRESSION ([pixel] >= 1000 and [pixel] < 2000)
73     STYLE
74         COLOR '#fd8d3c'
75     END
76 END
77
78 CLASS
79     NAME "1000-2000"
80     EXPRESSION ([pixel] >= 2000 and [pixel] < 5000)
81     STYLE
82         COLOR '#f03b20'
83     END
84 END
85
86 CLASS
87     NAME ">= 5000"
88     EXPRESSION ([pixel] >= 5000)
89     STYLE
90         COLOR '#bd0026'
91     END
92 END
93 END # LAYER
94
95 LEGEND
96     KEYSIZE 12 12
97     LABEL
98         TYPE BITMAP
99         SIZE MEDIUM
100        COLOR 0 0 89
101    END
102    STATUS ON
103 END # legend
104
105 SCALEBAR
106     IMAGECOLOR 255 255 255
107     LABEL
108         COLOR 0 0 0
109         SIZE TINY
110    END
111    STYLE 1
112    SIZE 100 2
113    COLOR 0 0 0
114    UNITS KILOMETERS
115    INTERVALS 2
116    TRANSPARENT FALSE
117    STATUS ON
118 END # Scalebar
119 END # MAP
```

## 4.2 Mapfile template test1.html

```

1  <!-- MapServer Template -->
2  <html>
3    <head>
4      <title>Interactive Map</title>
5    </head>
6
7    <body>
8      MapServer first version of an interactive map<br>
9
10     <!-- The central form the application is based on. -->
11     <form namer="mapserv" method="GET" action="/cgi-bin/mapserv.exe">
12
13     <!-- CGI MapServer applications are server stateless in principle,
14          all information must be "stored" in the client. This includes
15          some basic settings as below.
16          The example is based on the pan and zoom test suite:
17          http://maps.dnr.state.mn.us/mapserver_demos/tests36/ -->
18     <input type="hidden" name="map" value="[map]">
19     <input type="hidden" name="imgext" value="[mapext]">
20     <input type="hidden" name="imgxy" value="[center]">
21     <input type="hidden" name="layer" value="pop">
22     <input type="hidden" name="mode" value="browse">
23
24     <!-- A table for minimal page formatting. -->
25     <table border=0 cellpadding=5>
26     <tr>
27       <!-- First column: Map and scale bar -->
28       <td align=center>
29         <!-- The map -->
30         <input type="image" name="img" src="[img]"
31             style="border:0;width:1000;height:600">
32         <br>
33         <!-- The scale bar-->
34         
35       </td>
36
37       <!-- Second column: Zoom direction, Legend and Reference -->
38       <td valign=top>
39         <!-- Zoom direction -->
40         <b>Map Controls</b><br>
41         Set your zoom option:<br>
42         <select name="zoom" size="1">
43           <option value="2" [zoom_2_select]> Zoom in 2 times
44           <option value="1" [zoom_1_select]> Recenter Map
45           <option value="-2" [zoom_-2_select]> Zoom out 2 times
46         </select>
47         <br>
48         <!-- Legend -->
49         <b>Legend</b><br>
50         <br><br><br><br>
51
52       </td>
53     </tr>
54   </table>
55 </form>
56 </body>
57 </html>

```



### 4.3 Layer template layertmp.html

```

1  <!-- MapServer Template -->
2  <html>
3    <head>
4      <title>Population layer</title>
5    </head>
6
7    <body>
8      MapServer first version of an interactive map<br>
9
10     <!-- The central form the application is based on. -->
11     <form namer="mapserv" method="GET" action="/cgi-bin/mapserv.exe">
12
13     <!-- CGI MapServer applications are server stateless in principle,
14          all information must be "stored" in the client. This includes
15          some basic settings as below.
16          The example is based on the pan and zoom test suite:
17          http://maps.dnr.state.mn.us/mapserver_demos/tests36/          -->
18     <input type="hidden" name="map" value="[map]">
19     <input type="hidden" name="imgext" value="[mapext]">
20     <input type="hidden" name="imgxy" value="[center]">
21     <input type="hidden" name="layer" value="pop">
22     <input type="hidden" name="mode" value="browse">
23
24     <!-- A table for minimal page formatting. -->
25     <table border=0 cellpadding=5>
26     <tr>
27       <!-- First column: Map and scale bar -->
28       <td align=center>
29         <!-- The map -->
30         <input type="image" name="img" src="[img]"
31           style="border:0;width:1000;height:600">
32         <br>
33         <!-- The scale bar-->
34         
35       </td>
36
37       <!-- Second column: Zoom direction, Legend and Reference -->
38       <td valign=top>
39         <!-- Zoom direction -->
40         <b>Map Controls</b><br>
41         Set your zoom option:<br>
42         <select name="zoom" size="1">
43           <option value="2" [zoom_2_select]> Zoom in 2 times
44           <option value="1" [zoom_1_select]> Recenter Map
45           <option value="-2" [zoom_-2_select]> Zoom out 2 times
46         </select>
47         <br>
48         <!-- Legend -->
49         <b>Legend</b><br>
50         <br><br><br><br>
51
52       </td>
53     </tr>
54   </table>
55 </form>
56 </body>
57 </html>

```

## 4.4 MapServer – OpenLayers

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Single Image WMS</title>
5      <link rel="stylesheet" href="https://openlayers.org/en/v4.6.5/
        css/ol.css" type="text/css">
6      <!-- The line below is only needed for old environments like
        Internet Explorer and Android 4.x -->
7      <script src="https://cdn.polyfill.io/v2/polyfill.min.js?
        features=requestAnimationFrame,
        Element.prototype.classList,URL"></script>
8      <script src="https://openlayers.org/en/v4.6.5/build/ol.js"></script>
9      <style>
10     .ol-custom-overviewmap,
11     .ol-custom-overviewmap.ol-uncollapsible {
12       bottom: auto;
13       left: auto;
14       right: 0;
15       top: 0;
16     }
17
18     .ol-custom-overviewmap:not(.ol-collapsed) {
19       border: 1px solid black;
20     }
21
22     .ol-custom-overviewmap .ol-overviewmap-map {
23       border: none;
24       width: 300px;
25     }
26
27     .ol-custom-overviewmap .ol-overviewmap-box {
28       border: 2px solid red;
29     }
30
31     .ol-custom-overviewmap:not(.ol-collapsed) button{
32       bottom: auto;
33       left: auto;
34       right: 1px;
35       top: 1px;
36     }
37
38   </style>
39   </head>
40   <body>
41
42
43     <div id="map" class="map"></div>
44     <div id="legend">
45       
46     </div>
47     <script>
48     var overviewMapControl = new ol.control.OverviewMap({
49
50       className: 'ol-overviewmap ol-custom-overviewmap',
51       layers: [
52         new ol.layer.Tile({

```

```
53         source: new ol.source.OSM()
54     })
55     ],
56     collapseLabel: '\u00BB',
57     label: '\u00AB',
58     collapsed: false
59 });
60 var layers = [
61     new ol.layer.Tile({
62         source: new ol.source.OSM()
63     }),
64     new ol.layer.Tile({
65         preload: Infinity,
66         source: new ol.source.TileWMS({
67             url: 'http://127.0.0.1/cgi-bin/mapserv.exe?
68                 map=/ms4w/apps/pop/htdocs/interact.map',
69             serverType: 'mapserver',
70             params: {'LAYERS': 'pop'},
71         })
72     });
73 var map = new ol.Map({
74     controls: ol.control.defaults().extend([
75         overviewMapControl,
76         new ol.control.ScaleLine({
77             units: 'metric'
78         })
79     ]),
80     layers: layers,
81     target: 'map',
82     view: new ol.View({
83         center: [2269873, 5087648],
84         zoom: 2
85     })
86 });
87 });
88 </script>
89 </body>
90 </html>
91
92
```