Development of an interactive dashboard for performance reports on IBM Z systems with Linux

Term Paper T3000

PAUL BAURIEGEL

Contents

1 Introduction

Data is a valuable resource [@Hottelet.2017], especially when analyzing or comparing the performance of different technologies. With test data, it is possible to find bottlenecks in applications or errors in the process flow. However, most large data set are useless until they have been evaluated or visualized to gain insights. Visualization makes it easier for the analyst to understand the relations and evolutions inside the data. As a result, they allow him to work more efficient. Nevertheless, visualization takes valuable time, especially when it has to be done over and over again. There are several software solutions on the market [Meek.2017] to automate these visualizations and get fast insights, like *Tableau* or *Voyager 2* [@Wongsuphasawat.2016]. Still, there are also circumstances which require highly customized reports, as it is necessary for the *IBM z Linux Performance* department.
The *IBM z Linux Performance* department has several test procedures to evaluate the IBM Z Mainframes on Linux for several workloads. These test runs result in different kinds of reports. Until now these reports are analyzed manually. The goal of this project is to support performance analysists by creating an interactive dashboard for their requirements. The dashboard aims to automate the process of visualization without losing the flexibility in the data presentation.

This paper presents a working dashboard with specialized performance visualizations. It discusses the design choices for technology and architecture in this application and the ideas behind the different visualizations. The report highlights the advantages of this tool as well as further challenges.

Therefore the reader will be introduced to the design aspects of data visualization and the requirements for an interactive performance visualization dashboard. The first chapter explains how effective data visualization has to be implemented. The second provides a case study for the application, which includes detailed information about the data source.
On this foundation, the architecture and the technologies of this application are evaluated. This part compares the different technology options and underlines the design decisions.
With the architecture and technology decisions in mind, the following part will explain the dashboard implementation. After explaining the implemented architecture and some implementation examples, the last chapter will summarize the architectural choices and evaluate the developed prototype. The last chapter will also provide an overview of further improvements that could be implemented.

2 Concepts of data visualization

What means data visualization

Visualization meant in its original sense to “construct a visual image in the mind” [@McIntosh.2013]. In the computer science the understanding for this term is different, as Earnshaw defines it:
“Scientific Visualization is concerned with exploring data and information in such a way as to gain understanding and insight into the data.” [@Earnshaw.1992]
The term is now referring to building a visual image based on data. Alternatively, in a generalized context, it means data visualization. There are multiple definitions for the term data visualization on different levels of abstraction [@Owen.1999]. In the words of Ignatius and Senay:
“The primary objective in data visualization is to gain insight into an information space.” [@Senay.1994]
Data visualization has varieties like the information or scientific visualization [@Friendly.2009]. The two terms have a slightly different semantical definition. Information visualization is more precisely because it presents the data in a specific interpretation context [p. 4f, @Posada.2017]. The term scientific visualization is mostly used in the context of visualization for science research [@Earnshaw.1992]. Because the distinction is not relevant for this paper, the terms will be used equally in the further chapters.
The dashboard described in this paper is a tool for information visualization. Therefore information visualization will be described shortly. According to Gershon information visualization “combines aspects of scientific visualization, human-computer interfaces, data mining, imaging and graphics” [@Gershon.1997].
Information visualization has two primary goals [@Krypczyk.2014]:

- data representation, by transferring the data in a suitable layout for presentation
- data exploration or visual data mining meaning the discovery of new knowledge by transforming and interconnecting the data.

The data representation allows to find and view the data on specific criteria. The representation contains multiple views on the selection with a different context, and moreover, it is easy to remember for the analyst.
The data exploration highlights the relations and structures in the data. Together with multiple interconnected data sets, the data exploration then discovers the insights [@Dassler.1999].

The process of data visualization

Russell Ackoff defines the process of discovering new knowledge in three abstract levels for the perceptual and cognitive space [@Ackoff.1989]. The paper by Chen et al. extends these definitions for the computational space [@Chen.2009]. These levels are:
Data - Data is a concatenation of different symbols, for example to strings. In the computer science, this data can be any stored representations of models or attributes in the memory.
Information - Information is processed data that has been given a context. The context can be assigned by either a computational process or humans annotating the data.
Knowledge - Knowledge is an application of data and information to understand the processes and developments that created the data. Knowledge can be the results of a machine learning process or analysis by subject matter experts.
Understanding - Understanding extends the knowledge by providing explanations for the developments in the data.

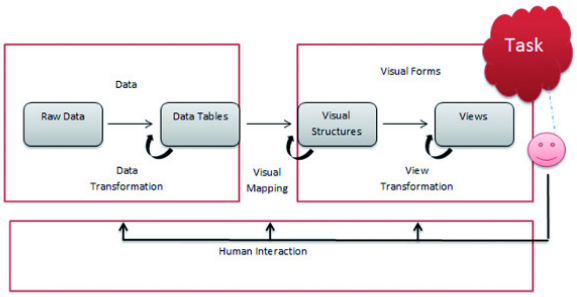

Figure 2.1: Process of visualization

The process of visualization has been defined in a reference model by Card, Mackinlay, and Shneiderman [@Shneiderman.1999, @Chi.Jan.2000, @Krypczyk.2014]. The model has three transformation phases, to transform the data between the different levels of knowledge. The process begins with the raw data being transformed into structured tables. Data in tabular form is easier to visualize. The *data* becomes *information*.
The visual mapping converts the data tables into visual structures. These have graphical properties instead of the mathematical relation inside the tables. The mapping into structures results in possible *knowledge* for the viewer. Such a mapping depends on the chosen chart type and if the data is discrete or continuous.
In the last transformation step, the visualization is adjusted to highlight parts of the data and get a better *understanding*. On a user-driven dashboard, the user **interacts with the visualization** to create different views.

The model has also been refined by dos Santos [@DosSantos.2004], adding new data filtering step

after the general data transformations. In the option of the author this model over specified for most modern use cases, where big data sets are evaluated in real time and can be filtered in the frontend. Furthermore, the model is specialized on generating images. When the visualization is rendered for example on the server side, this model is applicable, but in a modern visualization frontend, the visualization does not have to be a picture or in Ben Shneiderman's words: "The purpose of visualization is insight, not pictures." [@Shneiderman.1999]

Visualization best practices

In an interactive dashboard, the user can adjust his view of the data, but not the kind of representation. The representation type is stills defined in the visual structures. Best practices for visualization help to develop an appealing visual mapping of the data. The chapter summarizes the Graphical excellence points laid out by Edward Tufte [@Tufte.2001, p.13; @Hamed.2013].

Show the complete data - Provide the data source, with its limitations
Data over Visualization - Encourage the viewer to think about the substance rather than its representation or methodology
Assume an Intelligent Audience - Don't oversimplify the data or the visualization, induce the viewer to think about the data
Encourage data exploration - show the data at several levels of detail, from broad overview to the fine structure, enable comparisons for different pieces of data
Make data understandable - integrate verbal descriptions in the dataset, avoid distorting what the data have to say and follow conventions when presenting the data.
Avoid confusing charts - choosing the wrong representation can unintentionally confuse or deliberately mislead the viewer.

Choosing the best chart type

How intuitively the user can work with the data depends mainly on the type of chart. Mackinlay and Tufte use their publication to explain their selection of visualization type based on sets of examples. This paper will not summarize their main works. Instead, it will use simple classifications guidelines for the choice of visualization. To decide if either a chart or another representation is used, the Visual Thinking Codex by Dan Roam will be applied. This paper expects that all data will be represented as a chart. Because of that a chart selection diagram by Andrew Abela should help to use the best chart visualization [@Abela.2006].

The Visual Thinking Codex is a tool for visual thinking by Dan Roam [@Roam.2008]. Visual thinking is a methodology to organize the thoughts by visualizing them. The codex is using another tool

called SQVID. The idea behind SQVID is that the user can think of a thing in a variety of different ways. **S**imple or Elaborate, **Q**ualitative vs. Quantitative, **V**ision and Execution, **I**ndividual against Comparison, **D**ifference/change or As is. [@Roam.2008b]
The codex suggests different types of visualization. The codex suggests a visualization based on two questions.
1. What wants the user to know or How id he asking his question?
2. What weighting he has in mind based on the SQVID methodology.
Any z performance analyst asks primarily two kinds of question *How* is sth. and *Why* is it like this. For example *How* much memory is used?, or *Why* are we losing performance?. These questions come from different view angles so that no filtering based on the kind thinking can be made. However, on the kind of questions, it can be seen that plots and charts are the primary visualization type. Therefore a further chart study can provide some advice for choosing the right chart. [@Roam.2008]

As part of the Extreme Presentation ™ Method, Abela did provide a Chart Suggestion graphic which can help to choose the right presentation method. The main question is what should be shown. A comparison can be shown in bar or column charts. Distribution may be better with histograms. A relationship requires a bubble or scatter chart, and a composition can be shown with stacked or pie charts. This source is just meant to be a simple overview and is not in any meaning complete. [@Abela.2006]

Good resources [@Few.2006] for a more detailed exploration of graphs are the following. For the fundamentals of graph design “Show Me the Numbers” by Stephen few is a helpful resource. The best encyclopedic reference for visualization is “Information Graphics” by Robert Harris. For an even more detailed exploration of visualization “The Visual Display of Quantitative Information” by Edward [@Tufte.2001] and “The Elements of Graphing Data” by William Cleveland can be used.
The “perception edge” blog [@PerceptualEdge.2018] provides a collection of different papers, with solutions for more or less common problems when developing chart. Some of these suggestions will be used in the further implementation.

3 Use case study of the application requirements

Building a good visualization is not easy. To build a helpful visualization this chapter will introduce design principles, processed and an overview how the actual data looks like. This information should help to develop a dashboard on the needs of the user and with the right processes or technologies.

Use Case Analytics with User Experience principles

Choosing a matching visualization type makes the data understandable. According to Tufte good graphics are furthermore encouraging the viewers also to explore the dataset. Therefore this paper is proposing an interactive dashboard. To ensure the best user experience on the dashboard, this chapter will introduce the User Interface (UI) Design and User Experience (UX) Design. In simple words is UI how things **look** and UX how things **work**. [@Arora.2018] This chapter will outline the main UI design principles and which UX processes help to develop the best user experience.

UI design principles

Good UI design and suitable design principles have been defined by many authors over the time. The paper will cite three publications on a different degree of details. At first, the main principles of design will be presented based on the Donald Normans book “The Design of Everyday Things”. Afterwards, these principles will be specialized for the UI design. At last, the principles of website design will be explained based on “Don’t Make Me Think” by Steve Krug.

Thinking about the design of a product before building it, results in more user-friendly products. Norman provides seven principles, which good designers should follow.

Provide the Necessary Knowledge - The user builds his conceptual model how the thing is working. Functional design provides the information to interpret and understand the object. It contains visible clues for their function.
Simplify - The complexity of the product increases as the square of the features. To simplify tasks, four methods can be used. Provide the user a simple, intuitive mental aid. Show feedback to the user instead of making they remembering things. Third and Fourth, automate functions and change the

way of presenting the information.
Show How to Use a Tool and Explain Its State - The user should know about the consequences of their actions and the devices gave them feedback promptly.
Map Correctly - Developing a product should consider the user's natural mappings (e.g., "up" more naturally suggests "louder" than does "left")
Use Constraints - Contains help to understand the product. They lead the user to use the product correctly.
Expect Errors - If the user can create an error, it is very likely he will do so. Good design prevents as many errors as possible and gives informative feedback, for those that cannot be prevented. Most errors are either "slips" or "mistakes." Slips are errors by accident, like throwing the wrong document in the trash. Mistakes, on the other hand, are conscious actions, taken by having the wrong goal or misleading information.
Consider Standardizing - Standardization is useful because they base the application on a ground of common knowledge for all user. Consider a car, wherever in the world you take a car, all cars are operating in the same way.

For User Interfaces, the core practices concentrate more on the style aspects of the application. There are five fundamental principles a good UI is measured on [@Saidy.2017].
Color - The color is influencing the emotions of the viewer. Some meanings of colors are predefined like red for attention. Knowing about the psychology of colors is essential. Principles of color do also include using matching color schemes for charts or web elements. [@Farley.2009]
Balance - Balance in the design context is arranging elements to establish a positive communication to the user. Getting the interest and positive feedback by the user is achieved by multiple criteria, like symmetry, asymmetry, or alignment. [@Farley.2009b]
Contrast - Contrast is used for three main reasons. 1. Contrast is attractive to the eye, 2. It helps to organize information or build a hierarchy, 3. The Contrast moves elements in the focus of the viewer. [@Farley.2009c]
Typography - Fonts and the content itself are the main influence how users process the given information. Therefore the should be chosen with caution. [@Cronin.2009]
Consistency - Consistency is the key principle in UI design. It makes the product intuitive, usable and eliminates confusion. Consistency is based on familiar patterns and common visuals like color schemes, same typography or spaces between elements. [@Nikolov.2017]

This work will create a User Interface for a web-application as explained further in chapter [@sec:frontend-req]. Steven Krug did summarize the key principles for website Design in his book. To get a basic understanding of right website design, Krug's core points will here be summarized.

Create a clear visual hierarchy – As defined by consistency principle the web page should "clearly and accurately portray the relationships among the things on the page.". If elements are conceptually linked, they are visually linked as well. Important aspects are highlighted with visual cues, like bold or

larger fonts.
Take advantage of conventions – Standardizations in web pages make use of conventions. These are recognizable icons, like a person for the login profile or design languages like Material Design for the webpage.
Break up pages into clearly defined areas – Create different views for every independent context.
Make it obvious what's clickable – People click when using a website. The indication of clickable items and action make it easier for the user to understand *how to use the tool*
Minimize noise – To avoid distractions on the web interface remove anything that draws the users away from there focus.

User Experience processes

In contrast to the UI Design, the User Experience Design is a creative process to find the best design by analyzing the user [@Chesnut.2014]. It helps the business to define the brand image, based on the target group. The sales and the site traffic are increasing because the users are enjoying the product. Moreover, in general, UX establishing customer loyalty and goodwill [@Allabarton.2016].

Excellent User Experience has three primary descriptions: Useful, Usable, Delightful. Useful means that the solution is matching all user needs, also the needs, they might not be aware of. Usable describes the product to be easy to use and intuitive, the user needs no concentration for the basic functions. Delightful is the interface or product if the user enjoys it. It motivates the user to work with it.

Every UX design process has some common features, which will be explained shortly. These steps help to develop a great user experience. Therefore these processes are used while developing the dashboard. [@Lam.2016]
The first step of each UX workflow is research [@Nunnally.2017]. This step is about requiring data and thinking about to user. Based on this data different group of people sharing similar goals can be distinguished.
Here starts the UX Analysis, also called UX Strategy [@Levy.2015]. In this part, the data about persons brought in some context. These groups are bundles into a Persona. A persona contains the behaviors, interests, and goals for a user group [@Goltz.2014]. The persona is then extended with user stories and scenario maps. The user story describes from the users perspective what the product should accomplish or how it should be working [@Ambler.2014]. In extension, a scenario map is built. The scenario map is analyzing the step a user will take to accomplish a given task [@Turner.2010]. Based on the particular UX methodology further user analysis will be done.
When the UX Analysis is completed, the actual UX Design starts. The design includes building a variety of different Wireframes to build Interaction Prototypes for the best ideas. Good design means in UX not automatically good aesthetics. Actually, aesthetics has the lowest priority in the product design, more important is a consistent, structured and understandable layout [@Dizparada.2018].

Finally, the UI implementation will be done and delivered to test and validate the design. Therefore metrics and analytics tools are used to track how the users use our product and how satisfied there are.
To achieve good user experience many companies use a specific methodology. Very popular are Design Thinking [@Dam.2018] or the Double Diamond [@Schneider.2015] process. IBM is using a modified version of Design Think called IBM Design Thinking [@IBM.2017].

Use case study of the data source

All reports are stored in one server directory which the visualization dashboard has access to. The data storage contains reports of different kinds. These reports contain monitored performance data for IBM Z systems on Linux. This section provides a short overview about the raw data formats, that can be visualized by the application stack. There are four different formats: System Activity Report reports for Linux, profiling binary data and graphviz files in general, textual log files in a specific format and internal reports for processing units.

The System Activity Reports are collected by a Linux tool called *sar*. *Sar* is part of the *sysstat* package. Sysstat is a collection of performance monitoring tools maintained by Sebastien Godard. The sar command line utility collects multiple datasets, most important of which are the activity of CPU, transfer rates of the disk or utilization of memory. A complete list of collected data can be gathered from the sar manual [@Godard.2018]. The resulting data is stored in three formats, in textual data (sartxt), tabular CSV data (sarcsv) and in JSON form (sarjson). The application works with the sarcsv data :

Listing 3.1: Sysstat report - sarcsv file

```
1 # hostname;interval;timestamp;CPU;%usr;%nice;%sys;%iowait;%steal;%irq;%
    soft;%guest;%gnice; %idle
2 r72klaus;10;2017-12-21 12:24:34 UTC
    ;-1;63.88;0.00;24.48;0.00;0.01;0.24;9.14;0.00;0.00;2.25
3 r72klaus;10;2017-12-21 12:24:34 UTC
    ;0;63.44;0.00;25.77;0.00;0.00;0.20;8.19;0.00;0.00;2.40
4 ...
5 # hostname;interval;timestamp;proc/s;cswch/s
6 r72klaus;10;2017-12-21 12:24:34 UTC;13.60;399407.80
7 r72klaus;10;2017-12-21 12:24:44 UTC;1.10;391299.20
8 ....
```

Sar is an often used tool for performance monitoring in Linux. Because of that, there exist multiple visualization tools, which can be used to get visualization inspirations.

In contrast to the other report formats, the sar files are increasing in size based on the runtime of the test. In some test cases, the files can grow up to hundred's of megabytes.

The programs running on IBM Z are also profiled by different profiling tools, such as perf or gprof. The application is analyzing these reports by its self. The application stack makes use of an existing visualization program developed by Jose Fonseca called gprof2dot. The tool can handle the data from all popular profiling tool for C, Python or Java. This tool return graphs in the graphviz dot format. The application should render these graphs.

The textual log report have a special naming conversion and marker for the relevant data inside. The naming convention distinct's them from normal text files. The log files have the following syntax:

Listing 3.2: Logfile for Tensorflow tests

```
1 WARNING:tensorflow:Using temporary folder as model directory: /tmp/
     tmp6bU9WX
2 2017-12-11 17:37:38.556092: I tensorflow/core/platform/
     cpu_feature_guard.cc:137] Your CPU supports instructions that this
     TensorFlow binary was not compiled to use: AVX512F
3 ...
4 ###START### REAL:0.000411033630371 CPU:0.000412
5 Total words: 7664
6 ...
7 ###PREPARE### REAL:2.52717900276 CPU:2.520947
8 ###TRAIN### REAL:248.245265961 CPU:383.199024
9 ...
```

The fourth format for the processing unit reports (PU Report) is yet another log format. One log files contain about a hundred headers for data that are commonly in tabular form.

Current visualization workflow for the performance analysts

At the point of this paper, a web application for browsing performance data existed. This work was based on the idea of visualizing all the data that can be accessed by this application.
The current application has only some basic features. It provides a login page, which provides a protection layer for all other pages. These pages are static web pages like a documentation site or dynamic views for a server directory to download files.
To visualize the performance reports, they have to be downloaded and then externally visualized. The data often needs to be pre-processed by some script before the visualization.
Visualizing the reports directly in the current application would simplify the visualization workflow.

It makes the pre-processing unnecessary and creates a data-driven visualization automatically. An automatic visualization is speeding things up and helps the users to solve the actual problem.

Results of the use case analysis - What are the requirements for the dashboard.

This chapter will summarize the information, goals, and wishes for the application gathered by the processes described before. The information contains technical requirements as well as abstract ideas for the application.

1. The dashboard should be a working prototype, which can visualize multiple kinds of reports. The visualizations for the reports should contain multiple views based on the data sets.
2. It should extend the existing application for browsing the web directories. There is no need to develop an independent application.
3. The dashboard should work on a diversity of devices used in the department. A platform independent solution should be intended since multiple operating systems and browser are used by the people. The main platform is Desktop, not Mobile.
4. The user can dynamically interact with the visualizations. The visualizations are mostly charts, some of them could be graphs as well. The charts should be configurable, like setting the scale or enabling gridlines. The chart should also react on mouse interactions, like showing tooltips or more specific charts.
5. The interface has short loading times even with the larger report files. A frequent use case would be just to taking a short look at the file. Therefore only the necessary data should be transmitted to the client.
6. The timeframe for the implementation is seven weeks. The project should focus on some core features, and the existing code basis should be reused.
7. The work should be documented in the source code with comments and by this paper.

4 Architecture decisions based on the use case

In the previous chapter, the ideas and goals for the new dashboard had been collected. This chapter will discuss how these ideas determine the architecture of the application. This chapter will also define the requirements for the technologies based on the architecture and the goals for the application.

Architecture overview to achieve the application goals

The dashboard application will continue using the two-tier architecture. The primary reason for this architecture decision is the simple handling of users and files [p. 43ff, @FitzGerald.2012]. The server part can handle access and provision of the files. The client visualizes the data for the user. Which options of splitting the visualization pipeline are possible discusses the graphic by Tominski shown below [@Tominski.2006].

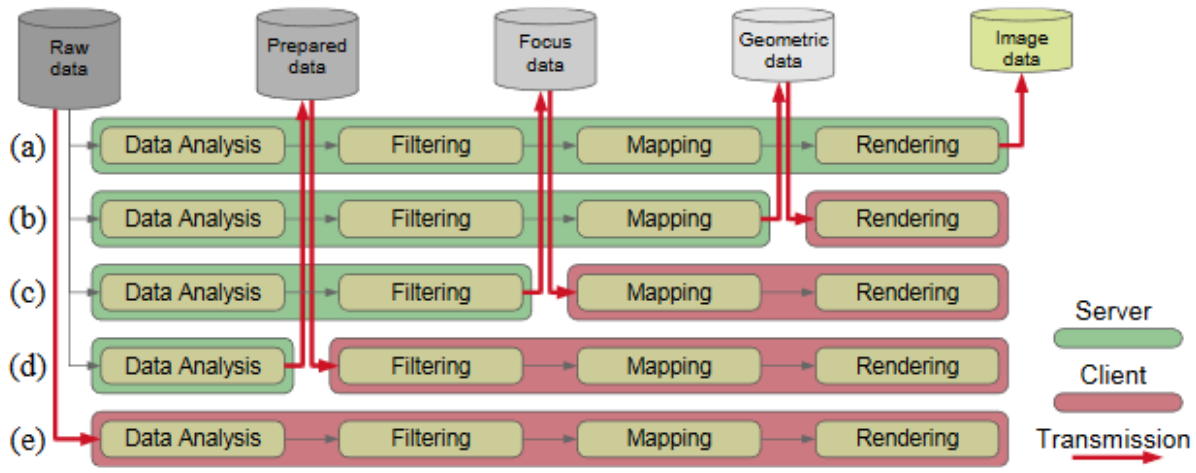

Figure 4.1: visualization pipeline in client-server scenarios

The raw data is stored on the server, sending the unprocessed data to the client is a poorly performing approach. The provisioning of large report files would take too long. Therefore the data need to be pre-processed on the server. The client should be able to get the pre-processed, but he can present

only a subset of it in each view. One user scenario is that the analyst takes just a short look at the dataset. In this case, he sees only one or a few views. Requesting the complete set of data would result in long initial loading times for each report. In an improved version the client request only the data the user wants to see. Therefore the server subsets the data for specific views.
Further work on the server side, like mapping or rendering the data [@Chen.2017] would speed up the clients loading time enormously [@Osmani.2017]. It would also increase the required processing power in the same scale on the server. For many clients, this solution would not be scalable. Because of that option *c* in the figure will be implemented.

The more generic reference model for data visualization by Ben Shneiderman helps to distinguish the tasks between server and client further. The server will handle all data, while the client will present the visual forms. The data transmission between the two components should be implemented in a standardized way, which allow parameters for the different views and files. The model distinct also the routing task between client and server. The server makes the rooting between the different files/data sources. He will route the client to the reporting component for this file. The client, on the other hand, will handle the routing between the different views of the selected dataset.

Backend technology requirements

According to the architecture, the backend has three main tasks: reading data in different formats, analysis and pre-processing the data and the provide the data in a standardized way. Additionally, the backend needs to deliver the web pages to the clients.

The server should provide an application programming interface (API) for the client. That enables the client to rely on standardized queries. The interface should be implemented with a stateless protocol. A stateless protocol is simple to use and to implement. The simple protocol makes the testing of the interface easier. A stateful implementation would perform better a few clients, but with too many open connections the application would crash. When using stateful protocol the server would need to keep the large data tables in memory, and so the application would scale poorly. Another advantage is the loose coupling between client and server. The loose coupling simplifies the independent development of front- and backend.[@Schreier.2016, @Aura.1997]
The most common programming paradigm for stateless web request is the Representational State Transfer (REST) [@Fielding.2000]. The REST interface also improves the scalability of the application. Since each query is context-free, each query needs the same amount of resources. This makes it easier to scale the application and identify bottlenecks in the application. When focusing on the frontend, the REST API will not restrict the technology or language that can be used. This enables this work to experiment with different frontend solutions without changing the backend. It is also helpful when using different types of frameworks based on the report type.

An emerging technology in this area is also GraphQL [@He.2009]. GraphQL is performing great on highly specialized query's. However, it adds unnecessary complexity to a simple API service [@Soares.2017]. Since the planned API provides only a subset of the report in a table pattern, without any filter condition REST will be the better choice in the sense of complexity and maintainability.
Together with the web request, the application has to handle standard HTTP request to deliver the frontend application. For a reliable and fast user interface these request should be handled by a fast and lightweight web framework, with REST support.

Reading and analyzing the report data from different formats is the core backend feature. The backend has to handle the diversity of the different formats without deduction of performance or extreme incensement in code complexity. Therefore the backend language should provide fast I/O operations and libraries that can work on large datasets. The libraries should be able to import, filter and provide different kinds of as textual and tabular data. The backend should also run scripts on the shell, to work with the binary blobs from profiling tools.

Dashboard technology requirements

Required frontend components

The dashboard will be integrated into the existing application for browsing the performance reports. The users do not need and want another interface for viewing the reports, building another graphical or web user interface would be counterproductive. This means the dashboard development is web-based. A web-based application also has other advantages for the use case like device diversity or the bookmarking of important views [@Nielsen.1997].

The web application is based HTML documents which will be styled by CSS stylesheets. The dashboard reacts to user interactions, routes between views and needs to visualize structured data. To modify the static HTML documents, most effort will go into the JavaScript (JS) code, which manipulates the existing Document Object Model (DOM) of the HTML files.

To implement the frontend tasks, each task will be supported by at least one frontend component. The components are simplifying the implementation by providing different programming interfaces. The frontend stack contains three components: a framework for building user interfaces, a kit of style elements and visualization library

The UI framework is required to handle the user interaction to build and route the different views. The styling components ensure a consistent user experience and provide building blocks for the UI forms. The visualization library handles the frontend part of the visualization pipeline. It will map the data into visual structures and render them. The following three chapters analyze the requirements for each of these components based on the use case study.

Requirements for the UI frameworks

The UI library will be chosen based on the following criteria's separately per report module. The main reason for using such a library should be to **improve programming speed**. Therefore a library should only be chosen, to make the code more readable and add no unnecessary extra complexity. The library should be lightweight, flexible and **easy to integrate** in the current application stack. As discussed before for the API interface the application, and therefore the frontend should be **scalable**. Effectively this should also mean that the library has a **robust performance**. The performance should also be testable without an extra application layer. Finally, the library should be **stable** and good to trust. This means it should be widely used or backed by a big company. Since this is a cooperate work, all libraries have been available under a license **free for commercial use**. The support for native development will not be taken into account. [@Neuhaus.2017]

Requirements for styling components

On top of the UI library, styling components are necessary. These improve the building speed and provide a consisted layout. The existing layout application is based on Bootstrap version 3. Since the UI will be extended extensively, the requirements for the new report modules are the determining factor. The existing UI has only one view, changing its layout can be done without much effort. Independently on the chosen UI Styling Component, it must be used in the complete frontend to provide a consisted User experience [@Nikolov.2017]. Which design language the User Interface is using is in this state of application not important. The main User Experience (UX) requirements are reliability, speed, and flexibility to structure and interact on the interface [@Dizparada.2018].

Therefore styling frameworks with a **huge number of components** are required. The dashboard reacts to user input. Therefore the framework needs a large part of form components. The components should also be **reactive**, meaning they are designed for user interaction. Instead of static components, they should be able to handle their state on their own. This simplifies the development of the components. On the other hand, the framework should be lightweight, for a good interaction speed. A **modular** UI kit can achieve this balancing act. The framework should implement **modern features** like grid breakpoints for Responsive Design and CSS Preprocessor support for SASS or LESS. [@Arsenault.2018]

Requirements of visualization libraries

While the UI library will handle all user interaction, the actual visualization of the data has to be done by a charting library. For these library counts the same as for the UI library, it has to be free for commercial use. It also should be **independent of the chosen framework** or on a huge amount of other

libraries. UI-Frameworks have a short lifecycle [@Allen.2018]. This leaves the possibility to switch the UI-Framework at a further time, without losing all effort brought into the visualization. In general **compatibility** with different UI-Frameworks and devices should be given. To ensure the future compatibility, the library should also be evaluated based on its continuous **enrichment and the maintenance over the time**. An active project also ensures the **documentation** and so the development effort to be expected. Since the data to be analyzed is huge, the library should provide the best **performance** possible, taking a more complicated implementation into account. It should also **support the API format** in which the backend is providing the data. The main goal of this work is to provide a highly customized dashboard for the requirements of the Z performance analysts. Therefore the library should provide a **wide range of visualization** and as much **customization options** as possible to increases the options in the visualization. [@Xing.2018]

5 Evaluation of the application technologies

After deciding on some basic architectural patterns and discussing the different technology requirements, this chapter can evaluate the actual technologies for the application stack. Hereafter the one technology choice in the backend and the three decisions in the frontend will be under discussion. The backend choice is primarily for the web framework with some consideration for the language environment. In the frontend, the choice for the UI-framework, the styling components, and the visualization framework is under evaluation.

Data preparation and backend technologies

The existing backend is using a Flask[@Ronacher.2018] environment on Python for browsing the server's directories and dynamically render the web pages with Jinga2[@Ronacher.2014]. This work aims to extend this application. For reasons of efficiency keeping the existing code basis would be helpful. Nevertheless if based on programming effort or performance a switch of programming language and framework would be necessary, this should be considered. This chapter will discuss benefits of the different possible web backend's and highlight the best choice for this application. Based on the choice of programming language the possibilities for data preparation will be explained.

To handle the web requests the backend requires a web framework. According to the Web Framework Benchmarks from TechEmpower [@TechEmpower.2017], the existing Python solution is not the fastest environment around. Independently of the test-type, languages like Go, Scala, Java or C++ are performing better than the Python frameworks. Additionally to the normal web requests, the application handles primarily REST calls. These REST requests require serializing the analyzed data into JSON. Looking at the TechEmpower benchmarks, the JSON serialization test runs seem to match best our utilization scenario. When only looking at request per seconds data, Java servlets are providing the best results for JSON Serialization with about 560k queries. The best python library is providing 87.5% of the top performance. The best overall results for plaintext queries are providing C and C++ libraries with about 4 million queries. Python reaches only 35.3% of this top score.
The switch to Java or C/C++ would require a considerable amount of time to set up the new environment and rewrite the code. Since the application is not meant to be a high-performance application, the extra amount of time satisfies not the benefits. A good compromise between performance and a

simple to implement solution could also be Go(73.0%) if performance would get more critical [@Bayburtsyan.2017]. Another try to improve performance if it is getting necessary could be to switch to a more performant web framework of python.

When taking a more specific look at python libraries in the TechEmpower benchmark, it is quite clear that Flask is not providing the best performance overall. The best-tested libraries for JSON and Plaintext queries in mid-2017 seem to be falcon (432k /804k) and wheezy.web (377k /956k). An older python specific benchmark test shows the same tendency [@Klenov.2016]. Both are not including a template engine for server rendering. Both points would be a requirement if handling the login mask and the file browsing component in the backend would be a requirement. In this case, bottle or weppy are better alternatives. There are multiple fast and uprising web frameworks [@Yegulalp.2016] like Sanic or Japronto that have not been covered in the latest TechEmpower benchmark. The latter promises about 1.2 million plaintext queries per second [@Przeradowski.2017], which would be even faster than the fastest frameworks from the TechEmpower benchmarks. However, since there are not comparing in the same benchmark, it is hard to compare them equally. Ether way Flask seems to have only a quarter of the performance that is possible in the Python environment. For further improvements in performance, a switch to the falcon or wheezy.web framework should be considered.

Another important use case for the application is the I/O disk performance. Reading the data possible into memory is also faster with compiled languages like C++, Java or Go. These programming languages do also provide efficient libraries for CSV files, as required by the sarcsv files. These libraries, like fast-cpp-csv-parser for C++, maybe not as intuitive as the pandas library in Python but even thought fair enough regarding usability. However, filtering these data sets in C++ would become then much more complicated in C++. In Java (or Scala) fast reading and filtering are simpler to implement by using the Apache Spark framework in local mode since the program runs on a single machine.

Even tough no programming language provides such a simple and fast implementation process for this problem. This point is also the main reason for staying with Python. The goal of this work is to implement a working prototype, within a short time frame. The main focus is more on a more extensive feature set, then on a high performant application. Since the performance of the Python environment is fast enough for the current scale of the application a switch of the programming language is not required.

Despite the fact that python may not be the fasted language [@Fulgham.2018] to solve this problem it has multiple other advantages. Python is an interpreted language with a simple and readable syntax. According to its author Guido van Rossum, one of the main concern when developing the language was "*to make Python code highly readable*" [@vanRossum.1996]. This goal has been fulfilled as shown in a comparison based on expressiveness [@Berkholz.2013], where Python is ranking significantly better than most other languages for this use case. Even with the readable language design, the syntax keeps flexible, when leveraging optional object-oriented or functional design patterns. Altogether

this minifies the development and debugging time. Regarding efficiency, this is a huge cost saver and makes the application more fault tolerant. On top of that, Python is great in combining different technologies and languages and provides a comprehensive documenting system with Sphinx [@Brandl.2018]. [@Koepke.2013, @Humrich.2017]

Handling data structures in Python is simple compared to other programming languages [@Gleeson.2017]. Even though other languages are also providing libraries for this use case. Measured by their number other alternatives are positioned better [@DeBill.2017]. However, Python is providing the most efficient libraries regarding documentation and flexibility [@Bobriakov.2017]. The core libraries that will be used in this application are Pandas and NumPy for reading and modifying the data. All other work like the visualization of the data will be done on the front end.

To summarize this chapter, there are good alternatives for the backend languages or frameworks. However, until the performance of our application is not getting the main concern. The current solution is the most efficient way to solve the given problems and develop the application. For the project time frame of 7 weeks, this is equally important and therefore the best tool of choice.

Choose of frontend technology

It is no question that JavaScript technology is changing rapidly [@Ball.2018]. Every year a new ECMA specification releases, new JavaScript frameworks are emerging or getting outdated and together with the frameworks the JS environment is shifting. This chapter will summarize the state of the JavaScript environment. How does a typical JavaScript application stack look like in 2018? Which technologies are getting preferred and what are there competitors. The main focus on the JavaScript technologies will be determined by the technologies required in the architecture [@sec:frontend-req]. In this context, the general JavaScript application context will be described shortly, continued by the evaluation of UI frameworks and style components for them. The last part of this chapter will then cover with which visualization libraries the data can be visualized.

JavaScript development stack

The development in JavaScript does not require an extra application stack. Every JavaScript can be embedded directly into a web page and would be working. Nevertheless, all tools in the JS development stack have been developed on purpose. By understanding this purpose and their functionality, they can simplify the development and even improve the web performance.

This chapter will introduce six different kinds of tools on which most JavaScript frameworks are requiring for the development [@Leon.2017].
The most important tool in this stack is the package management. The package manager creates

the project structure and installs all project dependencies. The Node Package Manager (NPM) is the de-facto standard. It can be configured through the `package.json` [@Wanyoike.2018].
JavaScript exists in different language flavors. The official JavaScript is standardized by ECMA[@ECMAInternational.2018]. Most browsers in the market are only implementing the ES5 specification. To work with newer versions of the ECMAScript (ES) or to use other flavors like TypeScript or Elm a transpiler is necessary. The transpiler converts the syntax into older versions of ES which are supported by a larger variety of devices. The most used transpiler in JS is called babel.
Like most other programming languages JS has tools for linting and testing the code. The linter is often used with a task runner, running while typing the code and detecting bugs or undefined behavior. The most common linter is ESLint. The tool can be configured with presets like the airbnb style guide, is then visualizing poorly written code segments.
To test the JavaScript app one can choose from many testing suites. Jest is very popular, but also other frameworks like Jasmine, Mocha or Tape are commonly used. Jest is very popular because it tests based on snapshots of the application. So it can test the state changes in the UI framework and runs test only on the updated components [@Beletsky.2018].
When deploying a JavaScript application, the JS files should be minified to improve the page loading. The necessary dependencies should be loaded and linked into the HTML file. A bundling tool, like Webpack, automates this task.
It is also possible to execute all the previous tools based on some conditions, like changes in the code. Therefore task runners like Grunt, Gulp or Broccoli are developed.

Dynamically updating the Document Object Model (DOM)

Based on the web statistics by W3Tech [@W3Techs.2018] jQuery is still the most used JavaScript library to modify websites. There are also multiple JavaScript Frameworks, which claim to be the best [@Allen.2018]. The main JavaScript-Framework competitors are React, Angular, and Vue.js. Indifferent which JS framework will be chosen, it should only be used if necessary. Using a UI framework will slow down the page loading as more powerful the framework gets. This means plain or jQuery web pages are faster than a web pages build by a UI framework [@Krause.2017]. Nevertheless, reporting modules with a sophisticated interface, like many subpages or complex routing between pages, will require a UI-Framework instead of plain JS.

Therefore the three UI-Frameworks will be evaluated in the further comparison. Based on web page usage [@W3Techs.2018], the GitHub stars [@Qian.2017] and the npm downloads[@Wang.2018] React is beating the other front-end frameworks. However, as seen in [@Qian.2017], Vue.js is growing faster and may become the main competitor to React [@Chartrand.2017], whereas Angular has been beaten in this ranking.

The reason to use the jQuery library instead of plain JS with ECMA5 syntax is that it makes coding

faster and easier to understand [@Oldenburg.2017]. Anyway adding the jQuery dependency to use only one feature might not be reasonable, since there are always plain JS solutions for a jQuery function [@HubSpot.2017]. This is getting even more enforced, since the latest specifications of JS EMCA6+ and now upcoming EMCA2018 [@ECMAInternational.2018] have an increased feature set of JS [@Aranda.2017]. The incompatibility of the newer specifications with an older browser can be overcome by transpilers like babel.
In large projects, however, UI-Frameworks are handling better the complex changes of the DOM. Where jQuery sites are requiring many callbacks and event listener to handle asynchronous code execution, frameworks are providing an interface for changing the state of the page. The huge amount of references in the plain JS based web pages makes it harder to understand the code basis.

Evaluated UI- Frameworks

UI frameworks bring new features in the frontend without making the source code unreadable. The web code is written with a framework usually becomes more modular and structured. This makes the application components better testable and reusable. The framework makes it possible to navigate between different subsites, without losing the data model or loading a new HTML file. Each framework implements these features slightly different. In the following, they will be described shortly.

React describes itself as “a declarative, efficient, and flexible JavaScript library for building user interfaces” [@Facebook.2018b]. The first version was released in 2013, now 16.2.0 [@Facebook.2018] is the current version of the library maintained by Facebook. The main characteristic of the library is the reactive approach. This approach means if the state of a component changes React is automatically rerendering the Virtual DOM [@Kurian.2017], by using its reconciliation algorithm [@Facebook.2018c]. The Virtual DOM is a virtual representation of the browser views in the memory. Applications written in React are divided into components by their functionality. This means JavaScript is generating the HTML by defining both business logic and HTML markup [@Alemu.2017]. React used therefore a syntax extension called JSX. Furthermore React is only a core library, it is used in general together with React-DOM for the Virtual DOM in the browser and several other extensions. [@Buna.2017]

Whereas React is backed by Facebook, Google is behind the development of Angular. The first version of Angular (AngularJS) was released in October 2010. A complete has then been done into Angular2 (September 2016), on which all newer Angular Versions are based on. The current version is 5.0.1. In contrast to its competitors, Angular relies on TypeScript. TypeScript is an extension of JavaScript based on the suggestions for EMCA6 developed by Microsoft [@Microsoft.2018]. The main features static typing and improvements for object orient concepts [@Alfonso.2017].
Angular also has a strict separation of technology and components. In contrast to React TypeScript is used to extend the HTML by creating HTML *templates* with Angularized markup. These templates are managed by *component* classes, and the actual application logic is inside the *services*. Together they

are boxed in *modules*. Every Angular App has a *bootstrapping* the *root module*, from where on Angular takes over the navigation, presentation and user interaction. This separation of different components follows strictly the model-view-controller pattern. Additionally Angular is providing most functions in its core dependencies and almost predefined workflow for the development. [@Larsen.2018]

Vue.js is the youngest of the three frameworks, introducing its the first version in February 2014 and the current version 2 in 2016. It is not developed by a big company, instead of by a small team of a dozen developers led by Evan You. Even though large companies like Alibaba or Baidu are using it. Vue.js has borrowed most of its concepts from its successors like Angular, react or Ember.js. Like React, Vue.js utilizes a virtual DOM and provide reactive and composable view components. In addition, it integrates frequently used features like routing and global state management in the core library. In contrast to Angular or React it forces the user not to learn superset of JavaScript, like JSX or TypeScript. The top priority of Vue.js is to provide a lightweight and easy to learn library for user views. [@Vue.2018]

Comparison of the UI frameworks

All three libraries have their eligibility for different domains. The evaluation will choose the best match based on the criteria defined in the chapter before. All libraries are using the MIT license, so the whacked criteria of being used without any license cost are fulfilled. [@Andrushko.2017, Neuhaus.2017, @DA14.2017]

Regarding programming speed, Vue.js is the best framework. According to several articles, it has the fastest learning curve, together with a very simple syntax that requires no additional technology like JSX or TypeScript. Not so easy to learn but also fast is React since it is only a simple JS library to implement with great flexibility in the workflow. Angular has a harder learning curve since it is very complex, but TypeScript (which is also available for React) improves the readability of the code. In large projects, the scalability of the development time is better with more developers [@McDaniel.2016]. All libraries are providing reliable documentation to learn the framework.

In point of integration into the application environment, React, and Vue.js are equally straightforward. The reason behind this is that both are only JS libraries, which can easily be imported in standard JS files without the need extra build dependencies like angular requires it.

The scalability can only be rated by way of deploying an application. Angular brings its strengths in large Single Page Applications (SPA) because it was built for them [@Kan.2018]. Because of the strong enforcement for modularity, large projects are better maintainable and suitable for larger pools of developers. Angular is also easily testable and therefore better scalable. React, and Vue.js can more easily be used for microservices since they require only pure imports of JS libraries. This enables better scaling for large environments. React is even better for microservices since every functional component of React is a single module. Because of the huge amount of libraries React is also better for

complex projects.

As shown in the js-framework-benchmark the frameworks cannot compete with the fastest framework available. There compared performance is almost one level. Vue.js is slightly faster, but the benchmark from mid-2017 is also not using the newest versions aka React 16 and Angular 5. Especially in React 16 with React Fiber [@Clark.2016] has made huge performance improvements under the surface. Important for the speed of the initial page loading, can also be the library size. Since Vue.js is the most lightweight, it is in general faster, whereas Angular is the slowest one.

React, and Angular have the backing by two of the most influential companies, on financial aspects as well as in their skill in the area. Vue.js, on the other hand, is founded via Patreon even tough it get used by Baidu and Alibaba. Based on the npm downloads React is the most used framework of them three. React is also the most popular one based on the stateofjs survey [@Greif.2017] and the GitHub stars [@Qian.2017]. Even though the community of Vue.js is growing faster at the moment.

React is the winner in this comparison based on the huge environment supporting it and its high popularity. The workflow of React is very flexible so that it is easily integrated into the current application environment. Vue.js stands out with a simple and lightweight framework, which would make development, even more, easier than React. Even though its only one of many uprising technologies. Vue.js may success React [@Chartrand.2017], but at the moment React is a safer choice of tool. React is backed by a big company and the most popular framework for now. The framework straightforward to implement and fulfills the requirements. Together these points make React the best choice as the framework for the more complex multipage sites.

Styling the User Interface

The decision on the styling components has no huge effect on the application. The styling components can be changed easily changed, and aesthetics are a matter of taste. The main requirements for the styling components are that they can be used with different UI-Frameworks. The support of different UI-Frameworks makes it easier to substitute the underlying UI-Framework layer. The small reports may also only use jQuery or even no library.

If IBM provides a suitable framework for the application, this would be the most consistent style component choice. At the moment internal design framework for the internal w3 design is not publicly available. IBM provides only the Carbon Framework [@IBM.2018] used by IBM Cloud. This framework has not been chosen, because it provides not enough form components for the user interaction. According to the GitHub statistics Bootstrap, Semantic-UI, Foundation, Materialize and Material UI are the five most popular UI-Frameworks. The frameworks have an equal feature-set in most categories. All are licensed under MIT. Some frameworks are more modular and performant than other, but the differences are to slight to make a significant distinction. All styling components have a responsive

design and can be used together with CSS preprocessors.
All frameworks are suitable for complex user interactions, except for the Foundation framework. This framework has not enough form elements for the required filtering components.

For the Material Design framework, Bootstrap and the Semantic UI components exist implementation for all discussed UI-frameworks.
This makes the choice of the framework an aesthetic choice. I like the Material Design by Google the most. Because of that I choose the Material-UI for the React Framework and Materialize for jQuery. [@Brillout.2018, @Metnew.2017, @Arsenault.2018]

Visualizing the data

This chapter evaluates the libraries for the core feature of this work, the visualization. Since there is a huge amount of options, this chapter will firstly filter the visualization libraries before comparing them to the given criteria's.

Filtering visualization libraries

Despite the user interactions on the dashboard, the data visualization is the main focus of the dashboard. For efficiently creating a huge an amount of plots for different views and datasets the use of a library cannot be avoided. There many visualization libraries for JS. To get a glue of the numbers, GitHub is listing about 850 JavaScript projects (07.02.2018) under the term visualization [@GitHub.2018]. This chapter can focus only on a subset of the available libraries. Since this work relies on some mandatory requirements, libraries which not match the following criteria will not be considered. The library must be open source and **free for commercial use**. This disallows libraries like Highcharts, FusionCharts, and ZingChart. Since the analyzed data is confidential, the visualization has to be done **locally** without leaving the Intranet. This prohibits API solutions like Google Charts even if there are free or commercial use. To leave the possibility open to switching the UI framework, the visualization library should not be **depended on frameworks** ones like Ember Charts, Victory, ReCharts or jqPlot. The reports are requiring a huge amount of different charts which should ideally implement with only one library unless there is an individual use case which requires a particular library. Therefore specialized libraries like Timesheet.js, VivaGraph, Sigma.js, Leaflet or heatmap.js are unsuitable. Only charting libraries are considered.

The focus report of this report is not a complete comparison of all JS charting libraries. Only the most important libraries will be evaluated. The distinction is done by GitHub star ranking and npm download statistics. Only projects with over 1k stars and npm downloads per day will be considered.

This condition limits the list to D3.js, Chart.js, C3, Raphael, NVD3, Chartist, ECharts, plotly.js, Vega, Vega-Lite and Vis.js [@Wang.2018b]. In both rankings, D3.js and Chart.js are completely outstanding. D3 is complete outstanding by even doubling both numbers of Chart.js. In the further course of the chapter, the ten different libraries will be shortly introduced with there main features and then compared based on the criteria defined in chapter [@sec:usecase-result].

Introduction of possible visualization libraries

The most popular library for data visualization is D3.js [@Bostock.2011] also known as D3 or Data Driven Documents. The library is based on Protovis and has 2 actively maintained versions v3 and v4. This evaluation will refer to the newer (and better) version [@Bostock.2017]. One of the reasons for the popularity is the flexibility of D3 [@Meeks.2017]. It can manipulate the DOM and integrates the existing web technologies, like Scalable Vector Graphics (SVG) and Canvas to create the visualizations. Any visualization project can use the library, because of its BSD 3-Clause, which allows commercial use as well. Using D3 for different projects becomes even easier since D3 has a comprehensive API documentation [@Bostock.2018] and an own platform for example visualizations with Bl.ocks [@Bostock.2010]. If a type of visualization not supported directly by D3, there is a high chance of an already written plugin to solve for this visualization type, This list is proving an overview about the D3 plugins [**?**]. The visualization by D3 itself is fast, and D3 can handle huge amounts of data compared to other libraries. The actual speed of the rendering performance depends on the underlying technology. Canvas is much faster the SVG graphics as shown in this performance benchmark [@UA.2012].

Many visualization libraries are using d3 and abstracting its complexity and low-level interface to an easier to use interface. From the current selection C3, NVD3, plotly.js, Vega and Vega-Lite are based on D3. Each of the four libraries has different levels of abstraction [@McGrail.2015].
NVD3.js has the highest level of abstraction of the four. The library is developed by Novus Partners and distributed with the Apache 2.0 license. The syntax is close to the original d3 library, but only rudimental documented and with a few possibilities. The library also supports only the basic types of diagrams: bar, bullet, line and pie charts. Each of the charts is provisioned with animations.
C3 indifference has a more extensive visualization API. The library is developed since 2013 by Masayuki Tanaka under MIT license. The visualization for C3 is done by a model definition in JSON. Compared to NVD3 it has less visual effects. The API parameter, on the other hand, is highly customizable and well documented. In contrast to NVD3, there is also no more coding in d3 required. This makes it a lot easier to use, but also loses the customization possibilities of d3.
Another D3 based library is called plotly.js. It is the JavaScript implementation of the declarative Plotly API. The visualization description exists for Python, Matlab, and R and is developed and distributed by the Plotly Inc. under MIT license.The framework is built on top of d3.js and stack.gl and provides over 20 different chart types. The documentation side provides an extensive API description together with

many examples.

Like C3 or plotly.js, Vega [@Satyanarayan.2016] and Vega-Lite [@Satyanarayan.2017] are also declarative visualization languages based on a JSON syntax. The languages are build by the Stanford Visualization Group, now called the University of Washington Interactive Data Lab. This is the same group that also developed D3. Vega is not intended as a replacement for D3 [@Heer.2017]. Like D3 there are both licensed under the BSD-3 clause. D3 is meant to be a low-level visualization framework, whereas Vega provides an abstract layer for representing and reasoning visualizations. The idea behind the project is to "promote an ecosystem of usable and interoperable tools, supporting use cases ranging from exploratory data analysis to effective communication via custom visualization design". The abstraction layer also has various improvements for Browser compatibility since there are translated in runtime to D3 code based on the environment. Vega-Lite is based on Vega and provides a higher level grammar to enable fast specification of interactive data visualizations. Both libraries provide an enormous amount of different charts and a very extensive documentation.

The main competitor of D3 according to usage stats is Chart.js. Like C3 or Vega, the charts are described with a JSON Model. The library supports eight highly customizable chart types, which cover almost all use cases. Chart.js provides multiple examples for these charts and the API is well documented. Chart.js is actively maintained by seven different developers and published as Open Source with an MIT license. Chart.js has good plotting performance using HTML5 Canvas. It also supports responsive rerendering. Like D3 it also can be extended by plugins.

Raphael is a lightweight library for drawing vector graphics. It is developed by Dmitry Baranovskiy under MIT license. In contrast to the other libraries, its the oldest release is a bit older than a year. The library is easy to use for visualizations, but it is not scoped for charts. This makes plotting charts harder to implement than in D3. Especially for older devices or a great compatibility Raphael is a good choice because of its unique support of the Vector Markup Language (VML), SVG and Canvas objects.

Chartist is a lightweight charting library for customizable responsive charts. Gion Kunz is the main developer of the library and licensed it under MIT. Chartist can create three different kinds of plots bar, pie and line charts. There are plotted into SVG and can be animated with the Synchronized Multimedia Integration Language (SMIL). To implement the charts, Chartist.js provides detailed examples and an API documentation. Chartist is also extendable by plugins.

The vis.js visualization library [@Almende.2017] initially developed by Almende B.V. and dual licensed under both Apache 2.0 and MIT. The library designed to be easily usable and to handle large amounts of dynamic data. The library consists of five main components: DataSet, Graph2d, Graph3d, Network, Timeline. For the use case important is the Graph2d component which plots data into bar or line charts. Pie charts are not possible which makes the library unsuitable for the visualizations purpose.

ECharts is a powerful charting library developed and maintained by Baidu. ECharts provides a large number of possible visualizations in two and three dimensional. The library is based on zrender, which

uses Canvas for the plotting, but ECharts is also available as a WebGL version. It can also be integrated with several UI Frameworks or GIS applications and has implementations in many other programming languages. The JavaScript implementation is licensed under BSD 3-Clause. The website provides many complex visualization implementations. The API is documented in English and Chinese, even though the main language in almost all user forums is Chinese.

Decision on the best charting library

The previous chapter has given a short overview of the most popular JavaScript visualization libraries. This has only a small subset fitting for the visualization requirements. The decision on the visualization depends mainly on the balancing between simplicity and performance of the implementation. If simplicity is more important Vega or even Vega-Lite would be a considerable choice. If performance and maximal customization are important D3 is the best choice. In the following, the decision on D3 will be explained on the established criteria.

The library should be compatible with a wide range of devices. All libraries provide a robust compatibility. The high-level frameworks, like Vega, are abstracting compatibilities between browsers automatically, so they are a bit better concerning compatibility. The best compatibility has Raphael. This is based on the support for old browsers by VML.
The chart implementation with Raphael requires a very low-level programming of the graphic because the library is not developed for plotting charts. Libraries like D3 are there more effective regarding simple implementation. Vega-Lite is providing here the best compromise between implantation speed and features.
Nevertheless, a big feature-set can only be reached with a complex low-level visualization framework. Besides the performance, the comprehensive range of features is the essential criteria. The libraries that provide the biggest feature-sets are D3 and ECharts. ECharts has the problem that the support is only available in the Chinese language, whereas D3 has a larger English community. The D3 community provides example implementations for almost any use case. D3 is also highly customizable and provides almost any kind of chart either natively or with a plugin. Additionally, it has an excellent performance for a huge amount of data, which makes D3 the best choice. [@DashBouquet.2017, @Baaj.2017]

Rendering Graphviz files

In addition to the charting libraries which are dynamically creating data-driven visualizations, the graph reports will require a simple and fast rendering library for graphviz files. Graphviz is an old and on the web very uncommon graph format. There exist only a few libraries that port the graphviz format into the web: livizjs, graphviz.js, graphlibDot, and viz.js. Only two of them are still maintained:

graphlibDot or viz.js.
graphlibDot has been written from scratch in JS, whereas viz.js is built on the original graphviz source by using emscripten. emscripten is a toolchain for converting c and c++ in WebAssembly. WebAssembly (WA) is a low-level assembly-like language that can run in the browser with almost native performance. If the browser does not support WebAssembly asm.js is used. asm.js is not as fast as WA, but also highly optimized [@Zakai.2017]. This makes the viz.js library much faster than graphlibDot.
Both libraries are not providing any extra features for parsing dot files. Actually, there are no existing JS libraries for parsing the graphviz files at all. This makes viz.js the best choice even if it can only render the dot files.
The parsing will be implemented to be manually in the frontend.

6 Implementation of the Dashboard

This chapter will present an implementation overview based on the architecture and technology decisions presented before. In the first part, the architecture will be explained in detail. The second part will present the key implementations of back and frontend. The implementation samples for the frontend will cover the REST API and the data pre-processing. The samples for the frontend will include the graph plotting with D3, the visualization of graphviz files and the page routing with React.

Dashboard architecture

The application is designed with a two-tier client-server architecture [p. 43ff, @FitzGerald.2012]. The advantage of this implementation that the two layers are strictly separated and can be developed independently. The layers are solely communicating over the REST interface. The following graphic presents the components of the architecture. The Web Layer contains the complete frontend logic separately provisioned into the different report modules shown as elements. The Python backend handles all web request from clients which are requesting the report modules. In also handles the REST requests from the report modules working at the client. In the following chapters explain the internal structure of the layers as well as their logic and execution flow.

Dashboard Architecture Overview

Application Frontend - Web Layer

The frontend environment of the application is a mixture of frameworks to build the application, the report modules itself and the tools to deploy the modules. The current solution has four reporting modules for the four kinds of files described in chapter [@sec:usecase-data].
Each reporting module is a single page app that has its application stack. The application stack contains the three core libraries for UI, style, and visualization together with the deployment tools. All modules share the same basic deployment tools. NPM scripts for automatic building, the ESLint linter and webpack for bundling, and minifying the JS code. The React based applications are including also other tools created by the create-react-app boilerplate. The React stack also contains the Jest testing tool and the babel transpiler.

Each report includes a different combination of core libraries. The Graph report is visualizing the profiling data. The report is static. It requests only the dot file from the server an makes some basic manipulation on them like shrinking the function signatures. The viz.js libraries are rendering these dot files. The other two core libraries come from the minimized stack. The log-file report is implementing the same core libraries, but it has a more sophisticated visualization of the data. Therefore the D3.js library is used. The sysstat report implements complex visualizations as well but also has multiple views. The rendering and routing between of multiple pages is faster when handles by the UI framework. To accomplish these task React is used in the backend. The CPI report has the same complexity. So this report is using React as well. In contrast to sysstat, the CPI data report is not visualizing the data with plots. It is only presenting them in tables.

The minimized stack is handling the REST request with the `XMLHttpRequest`. The react stack uses `axios` for the JSON queries. If the report imports the D3.js library, it is using the `d3.json` implementation directly.

Application Backend

As discussed in chapter [@sec:backend-tech] the backend is written in Python 3.6. The project has several dependencies.
Flask is handling all web requests, the REST interface as well as the backend rendering and the provision of the web pages. For each report module, the web provisioning has to be extended with the new routing rules and the data-processing functions. Each report view also needs one or more REST requests. Each REST Call is bind to the processing functions.
Since each report has different file types and requirements, a common layer is not possible. The only common layer is the data access, which contains only a few functions.
The data preparation has many different tasks. Some performance reports like the sysstat or the CPI data do contain multiple datasets. The “header functions” are iterating over these files to find the line range of the datasets. The client can request these datasets separately. Since the backend is RESTful, each request will execute an independent data preparation function. If the data-set is tabular, the pandas library will handle the call. Otherwise custom log analyzer functions are processing the request. The pandas library will only load the requested file-part into memory to transform or filter the dataset for the client’s demands. The log functions are working similarly.
The handling of profiling data is an exception to the described RESTful workflow. Profiling data, e.g., perf data files, is most cases binary data and cannot be analyzed directly. At first, the data has to be processed by the profiling tool itself. Like in this example for perf: `perf script | c++filt | gprof2dot.py -f perf > perf.data.dot`.
Therefore the backend will execute a bash sub-process. After the serialization by the profiling tool, the output is directly piped into gprof2dot. Gprof2Dot is written José Fonseca and creates a graphviz file

with the corresponding call graph. This dot file is then delivered to the client.

Dashboard Implementation

The second part will present the key implementation in the back and frontend. The implementation samples for the frontend will cover the REST API and the data pre-processing. The samples for the frontend will cover the graph plotting with D3, the visualization of graphviz files and the page routing with React.

Implementation of the backend engine

Data preprocessing in Python 3

REST implementation with Flask

Implementation of visualization and user interface

Page routing with React v16

Visualisation of Graphviz files

Data plotting with D3.js v4

7 Conclusions and future work

::: What have been the requirements ::::::
Time
::: Architeture conclusion ::::::
Client Server Application

::: Technology conclusion ::::::
Best web framework = Flask is used because it is already used -> consider switching the web framework in python to wheezy or falcon
speak about Python vs. Go…???
Best UI-Framework -> react, vue may be a competitor
CSS Styling not important
Best Visualisation Framework -> D3

::: Implementation conclusion ::::::
- four reports, with difernet coverage and aspects -> further work have to be done -> more reports, more plots -> …

::: Further Work ::::::

live stream data, while the report is running

single sign-on IBM

interconnect the reports

plugin structure

::::::::::::::::::