Examensarbete

HÖGSKOLAN
I SKÖVDE

A/B-testing for web design: A comparative study of response times between MySQL and PostgreSQL

Implementation of a web based tool for design comparisons with stored images

Examensarbete inom huvudområdet Informationsteknologi
Grundnivå 30 högskolepoäng
Vårtermin 2017

Tobias Lindberg

Handledare: Mikael Berndtsson
Examinator: Henrik Gustavsson

Abstract

Web development is a challenging task and it's easy to neglect feedback from users in development stages. That's why the aim is to create a tool which would help the communication between developers and users by using A/B-testing. The idea is to let developers release two choices containing images, which would be the intended design changes. By letting the users vote for the preferred option, they will be able to provide some feedback for the developers.

Response times becomes a critical factor for the tool's overall success. Therefore, this study compares MySQL and PostgreSQL through a technical experiment to see which database would be the better option regarding the image processing.
The experiment indicated that PostgreSQL would the better alternative regarding the subject, as it had the most responsive processing of images. This prototype provides a good foundation for a potentially useful system that could be implemented in future work.

Key words: [A/B-testing, Response times, MySQL, PostgreSQL]

Contents

1 Introduction 1

2 Background 2

2.1 Web analytics 2

2.1.1 A/B-Test 4

2.2 Database management 5

2.2.1 MySQL 5

2.2.2 PostgreSQL 5

2.2.3 PHP 6

2.2.4 BLOB-format 7

3 Problem definition 8

3.1 Field of research and hypothesis 8

3.2 Method description 10

3.2.1 Chosen method 11

3.2.2 Alternative method 13

3.3 Ethical aspects 14

4 Implementation 15

4.1 Research 15

4.2 Progression of implementation 17

4.2.1 Web page structure 17

4.2.2 Database structure 18

4.2.3 Add A/B tests 20

4.2.4 Select testID 21

4.2.5 Insert image 22

4.2.6 Search form 24

4.3 Pilot study 26

5 Evaluation 29

5.1 Presentation of research 29

5.2 Analysis 36

6 Conclusions 37

6.1 Summary 37

6.2 Conclusions of experiment 38

6.3 Discussion 38

6.3.1 Application 38

6.3.2 Experiment 40

6.4 Future work 42

References 43

1 Introduction

Designing and developing applications on the web is quite a challenging topic. One of the biggest aspects to consider is the user satisfaction of the product. If the users are not satisfied with their experience, the product itself will not be very useful, as they might consider other options instead. Therefore, the aim is to create a tool which could help developers with collecting user feedback about web design changes. This tool will apply web analytics to collect and analyze user preferences within web development. Web analytics is a scientific field used on the web to analyze lots of different data. Such data includes user statistics and their preferences, which will be the primary focus of this tool. Kumar, Singh, Kaur (2012) quoted the following about the usage of web analytics:

> "Web analytics is the art and science of improving website to increase their profitability by improving the customer's website experience."

The method of web analytics that will be performed in this tool is called A/B-testing, also known as **split-testing**, of which it will be referred as in this work. It is a test where two options will be measured against each other, and statistics from user input will be analyzed to see which option was the most successful. This tool itself will consist of a web application that contains images with suggestions of web designs and new implementations. The idea is to let users have a direct impact in the development process of software by having the option to see different suggestions of design changes within the application, and then vote on the preferred option. By using such a tool in web development, it would become easier to gather user feedback about preferred changes and additions for products.

The application itself will be connected to a database to store the image and user statistics. Database response times have a huge impact on the user experience, as it is one of the biggest reasons for delays while navigating web pages. When users experience slow response in web applications, their satisfaction will decrease and the experience might turn out so bad that the user stops using the product altogether (Hoxmeier & DiCesare 2000).

Therefore, the database of the system becomes a point of interest to see if it is possible to optimize the response times. To do this, two relational databases will be implemented and compared, the first as a MySQL database and the second as PostgreSQL. Experiments will be performed to measure them and see which one that can process the data faster than the other alternative. **Image processing** will be the most interesting variable, due to the applications primary usage of images for the split-tests. Image processing in this work is referred to the process of selecting an image from the database up until the image is visible in the application. By concluding the better option for the database regarding image processing, a better user experience in the system will be achieved, which would be a crucial part for the applications overall success.

A web application built with mainly HTML, JS and PHP would be implemented for this research, as well as instances of the two separate databases. Once everything is setup accordingly, a pilot study will be run to conclude if measurements can be performed. If possible, an experiment will be conducted where the image processing will be measured to see which database would perform faster. The tests of the experiment will measure response times and benchmark those results, where the results can be analyzed to conclude which database does perform image processing better.

2 Background

2.1 Web analytics

Web analytics is a scientific field that analyzes many kinds of data from the web. Some examples of approaches within Web Analytics are data visualization, analyzing web performance and user statistics. It holds a scientific value since it manages statistics, data mining techniques, report summaries, operations and a methodological process (Kumar, Singh, Kaur, 2012).

There is an increasing need to meet customer preferences and understanding their behavior, in which Web Analytics plays a big role (Kumar, et al. 2012). It becomes an excellent platform to gather their preferences and analyze user behavior. Kumar et al. (2012) also refers to web analytics as a very important component of many web based systems and helps in taking business decisions. They mention the three fundamental pillars in web analytics, data collection, storage and evaluation.

Web analytics has a very large range of applications. One of the most commonly used tools for web analytics is called Google analytics and is used to measure traffic in 50% of all websites (Cegan & Filip, 2017). Cegan and Filip (2017) also states that while this tool is mainly used for marketing, it can also be applied to analyzing faults in the user interfaces as well as the user experiences, which further confirms the diversity of usage with web analytics. In figures 1 and 2, we see examples of the interface of Google Analytics whilst tracking statistics about web pages. Figure 1 contains an example of statistics about user sessions and figure 2 displays statistics regarding e-commerce.

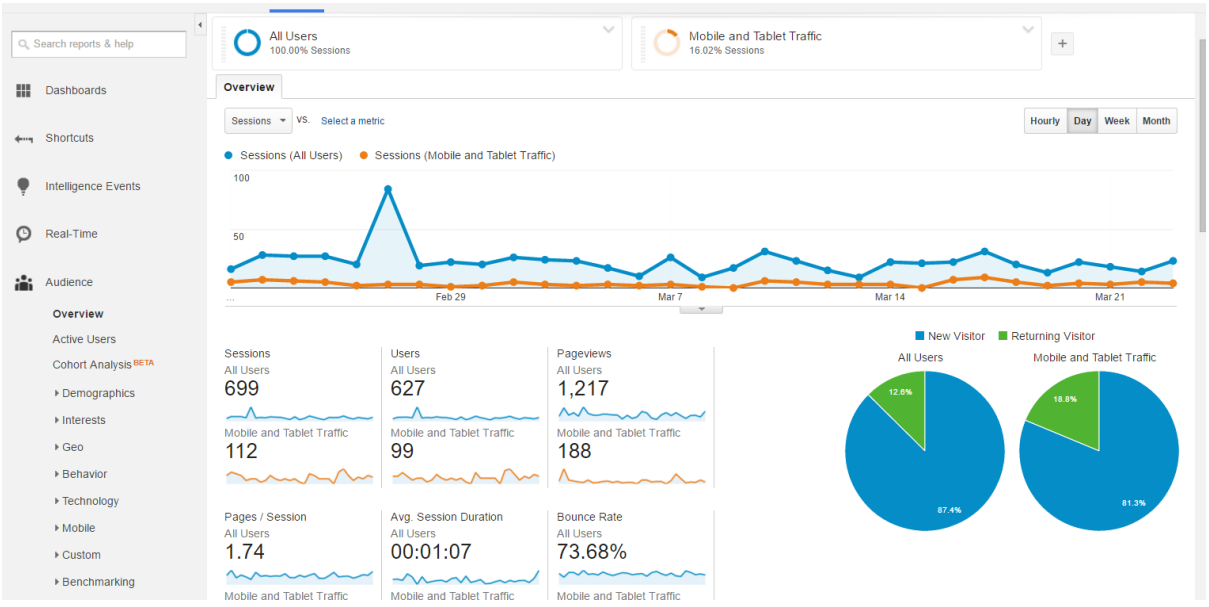

Figure 1 Statistics of user sessions in Google Analytics

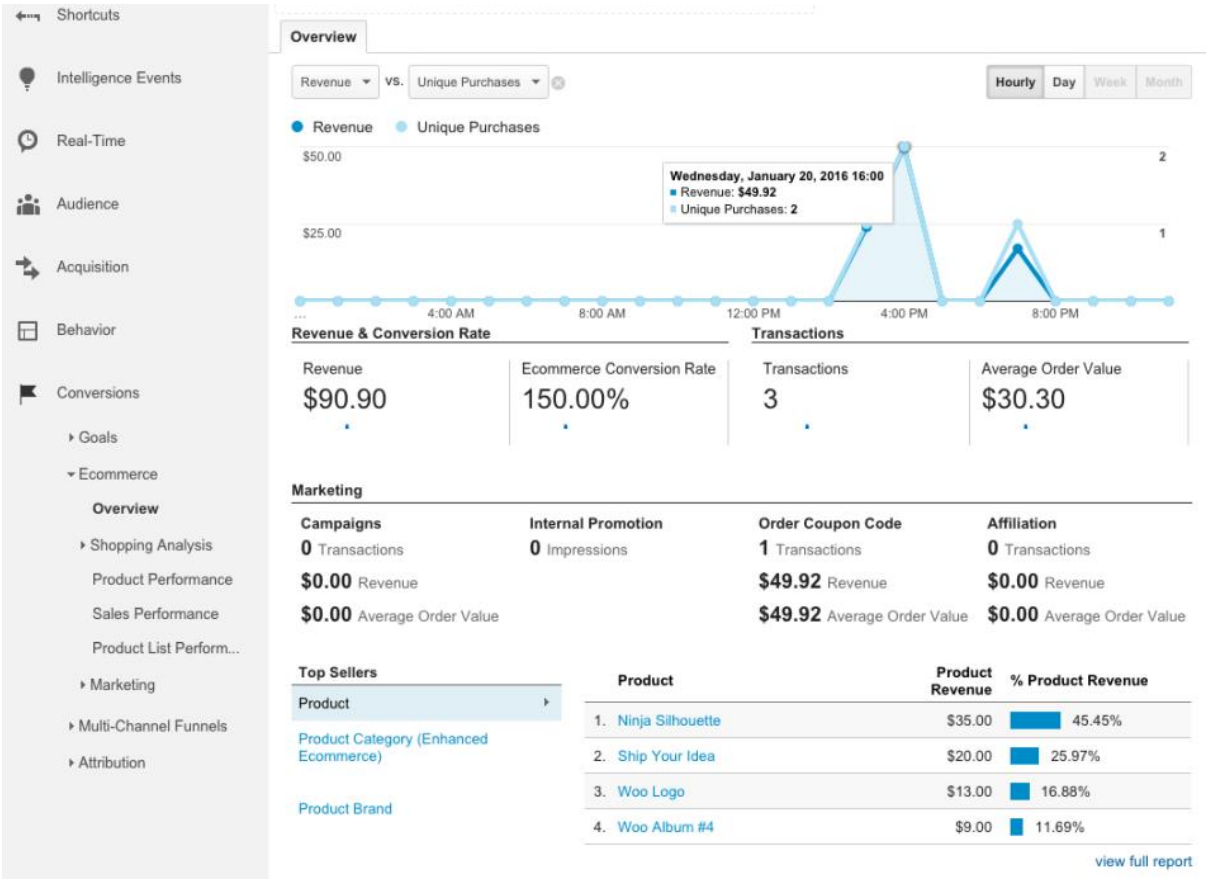

Figure 2 Statistics of e-commerce in Google Analytics

Though web analytics has a lot of benefits, it is worth noting that it does come with some limitations. An example is that we can see *how* a user is interacting with a website, but it becomes difficult to see *why* they are engaging in their specific behavior (Brown, Lush, Jansen, 2016). Brown et al. also refers to its accuracy issues, as there are typically error margins in a range between 5-10% within analyzes.

2.1.1 A/B-Test

A/B-testing, also referred to as split-testing, is a form of controlled experiment used within web analytics. It has a long history of usage in medicine, and is growing within other areas, such as marketing, e-commerce or other consumer-centric domains (Brodovsky & Rosset, 2011). Many big corporations such as Google and Microsoft take advantage of these experiments for their development. The tests mainly consist of a scenario where two designs proposals appear, where the result will be either one of those options. The term A/B-test refers to those options as A and B, hence the name A/B-test.

Split-testing occurs commonly in web development and is used to determine which result that will be the most successful in the implementation (Adinata & Liem, 2014). A commonly used example is using two different web pages for the same application and track the usage statistics about them. The data can be analyzed and then concludes which one that the users would prefer by checking the statistics of their usage on the pages.

The tests are a useful help to measure factors that define the overall success of the elements that are being compared (Adinata & Liem, 2014). In their research they applied split-testing in combination with experiments to compare two different design implementations of a mobile application. Their success led to the assumption that it will be a much viable choice of gathering user feedback.

2.2 Database management

2.2.1 MySQL

MySQL is an open source database management system that is based on the structure query language (SQL). It is commonly used with relational databases where SQL queries can be used to alter tables where data is stored. Insert, Delete and update are some of the commands used to alter the information in the database. Select is a command that is used to fetch the data from the database and can be used to retrieve information from the database to a web application. Figure 3 displays an example on how the SQL syntax can be used to fill a table with statistics in a MySQL database and is then retrieved with the select statement.

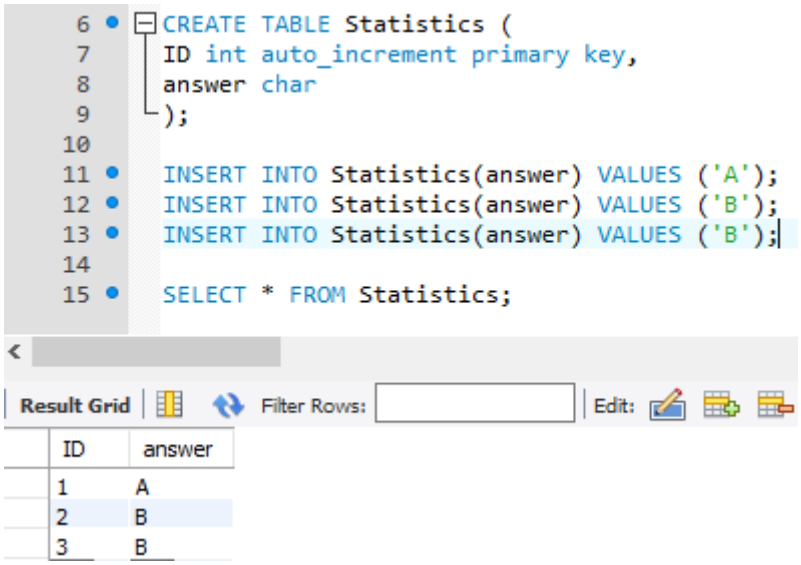

Figure 3 SQL queries

MySQL has been very successful in highly demanding production environments and its speed and security makes it a very viable choice to access databases from the web (Di Giacomo (2005). Due to its open source nature, it also becomes an excellent choice to use for database management because it is free to use.

2.2.2 PostgreSQL

PostgreSQL is also a relational database management system and is very similar to MySQL in many aspects. PostgreSQL is also open source and it runs on the Structure Query Language (SQL) as well. However, initially it was using QUEL instead of SQL as its language, and the system was simply called Postgres as well as having a commercial license. In 1996, a new project started where a new version of the database system, which built on the old Postgres was created, called Postgres95. The language was translated to SQL as well as the license was made open source, the new system was further developed and became PostgreSQL as it's known as today.

Figure 4 displays an example of an equivalent SQL syntax used for PostgreSQL to create the same table listed as in figure 3. Since both databases runs on SQL, the scripts are very similar. Only one noticeable difference appears for the primary key, where the datatype *bigserial* is used instead of *auto_increment*. However, they both have the exact same usage, where the ID value is added by 1 for each new row to provide a unique identifier for each row.

```
17  CREATE TABLE Statistics(
18  ID bigserial PRIMARY KEY,
19  answer char);
20
21  INSERT INTO Statistics(answer) VALUES ('B');
22  INSERT INTO Statistics(answer) VALUES ('A');
23  INSERT INTO Statistics(answer) VALUES ('B');
24
25  SELECT * FROM Statistics;
```

Data Output | Explain | Messages | Query History

	id bigint	answer character (1)
1	1	B
2	2	A
3	3	B

Figure 4 SQL Syntax in PostgreSQL

2.2.3 PHP

PHP is a well-known server side-scripting language that has been the choice that will be used for establishing the connection to the databases. PHP is one of the most popular programming languages for service-side scripting within application development. The language is highly dynamic, which provides developers with large amounts of flexibility (Hills, Klint, Vinju, 2013).

As dynamic web applications commonly use information that is stored within databases, a solution for the connection between frontend and backend must be implemented. Figure 5 displays an example of how a database connection could be written with the help of PDO in a PHP script, where variables determine how the connection is configured.

```
$PDO = new PDO('mysql:host='.$dbhost.'; dbname='.$dbname, $dbuser, $dbpass);
```

Figure 5 A PDO connection to a MySQL database through PHP

Another feature of PHP is that it can run SQL queries that calls to databases, which will be very useful tool for web applications to process data. In figure 6, we see an example on how SQL queries can be written directly in PHP to retrieve data from a SQL database, where a PHP variable is being used to control the condition attribute of the query.

```
$fetchQuery = $PDO->prepare("SELECT testID FROM SplitTest WHERE testName='$testName'");
$fetchQuery->execute();
```

Figure 6 SQL query in PHP

2.2.4 BLOB-format

BLOB (Binary Large Object) is a format consisting of binary data that is used to store some objects into a database. Some examples are images, audio or other multimedia files, even executable code can be stored as BLOB-data in a database. BLOB-data allows an efficient way of storing such data into databases, although they cannot be viewed or changed in the database workbench itself. Conversion from binary data to its source needs to be applied to view the BLOB-content. That is why an external application is required to access the BLOB-data. A web application is a very viable choice to achieve this, as PHP does provides conversion of files to binary and back. Both MySQL and PostgreSQL supports the BLOB format, which will be an essential component for this study. There are 4 kinds of BLOB types that can be used in MySQL databases. Smaller BLOB types takes less time to manage, but can at the same time not store as much data as the larger types. The most appropriate type to choose would the smallest one possible that is still large enough to store the amount of data required.

Format	**Storage**
Tinyblob	255 Bytes
Blob	~64KB
Mediumblob	~16MB
Longblob	~4GB

Table 1 – BLOB types

3 Problem definition

To implement a good web design can be a very challenging assignment. There are many conditions to consider when it comes to the user experiences and if you do not research into the preferences of the users, there is a risk that the usage of the product will decrease significantly. You must find a balance in the design by neither having a boring and confusing design nor apply too advanced content just to fill out the site with elements (Gershon, Czerwinski, Neale, Nielsen, Ragouzis, Siegel, 1998). Gershon et al. (1998) also refers to the key for implementing good design as adding value to the user experience. If a user feels like the application has been useful for them, they will become more consent with the overall experience of the product.

Hence, we can draw a conclusion that getting the feedback of the users can play a significant role in the overall success of an application. Therefore, the intent is to create a tool to help developers gather user feedback during development of design changes. However, there are many ways to gather user feedback. One of the methods to collect user feedback can be achieved through split-testing, which will be applied in this case. The aim with such tests is to create a tool that will bridge the gap in communication between developers and users, as the developers will receive a new platform to obtain user feedback.

3.1 Field of research and hypothesis

Split-testing will be a major component in this study, where a user will stay on a single web application while performing the A/B-tests. Those tests will include comparisons with two suggested web designs layouts displayed next to each other, where the user can choose the preferred one.
Images have been chosen to display the designs that will be used for the tests. This is mainly due to their simplicity to implement. It is relatively easy for a developer to apply a couple of images with the intended design changes, rather than writing code to display prototypes for the users with the design changes. When using images, a lot of time and resources can be saved, as images will generally not take very long time to produce compared to writing code to apply design changes. Another advantage of using images is that the developers can get a clear view of how their design should look like before starting implementing its structure with code.

It is currently quite difficult to find any research regarding the usage of images as the element within split-testing, as the tests are most often used in scenarios like Adinata and Liem (2014) constructed, comparing whole web pages. While this makes it challenging to find information regarding the efficiency of images in split-testing, it becomes an interesting topic as there is not much research that has been performed with such applications.

By storing and retrieving the images for the split-tests, a conflict appears where the image processing through the databases needs to work efficiently. Response times becomes a new issue, as they will remarkably impact the user experience of the tool. When long response times appear, the user satisfaction and productivity will decrease significantly. This can result in a drop of usage and the user might even quit using an application completely (Hoxmeier & DiCesare, 2000). A conclusion can be made that response times have a great impact of the overall user satisfaction of an application.
That's why the response times will be a key component to research for successfully implementing the tool with split-testing. To measure response times, the database systems MySQL and PostgreSQL will be compared in an experiment to see which one is more efficient at processing images, thus lowering response times.

If one of the databases would process the data of the tests significantly worse than the other, it becomes bad choice for this type of application, due to its restricted user experience as the response times will be slower (Hoxmeier & DiCesare, 2000). It would also be quite contradicting to launch an application with the intent of aiding user satisfaction within web development, and not directing any focus of the issue to the application itself.

Stancu-Mara & Baumann (2008) performed research where they compared MySQL and PostgreSQL on how they performed when large objects was used, including data with BLOB-format. They concluded that MySQL was a better choice compared to PostgreSQL, due to its ability to process the data faster. Their research had some similarities to the experiments that will be performed in this research in some regards, but in this case the focus will turn to image processing as the specific point of interest. This will generate less generic results of the research, as most measurements between those databases includes data processing overall. It has also passed 10 years since Stancu-Mara and Baumann published the results of their research. In that time a lot has happened with the development of the database managers and the results may possibly differ quite a bit, which is another reason why it would be essential to perform new experiments to measure the databases.
Based on the results of the research of Stancu-Mara & Baumann (2008), a reflection can be made that MySQL will most likely perform better when measuring the databases in this study as well, which is the leads to the following thesis:
MySQL will perform image processing more efficiently than PostgreSQL.

This thesis builds a foundation which can be used to setup an experiment, reviewing the most suitable database for this type of tool, as well as processing of images in relational databases overall. Image processing in this research refer to the steps as a split-test is being searched for and a SQL query is executed, the images are converted from binary data to its original form, and then displayed in the application. In other words, images being transferred from a database to being shown in the application.

3.2 Method description

In this research, a web application will be created which contains split-testing that is applied to collect user preferences in development of web design. The web application itself will be created with programming languages such as HTML, CSS and JavaScript. The split-tests will consist of two images that contains examples of web design, where a user will be able to choose the preferred one by clicking one of the pictures. An instance of MySQL and PostgreSQL will be implemented to supplement the web application with database to store the images and statistics. The connections between the web application and the databases will be handled with PHP. Figure 7 shows a flow on how the user statistics of the tests could be stored to the database.

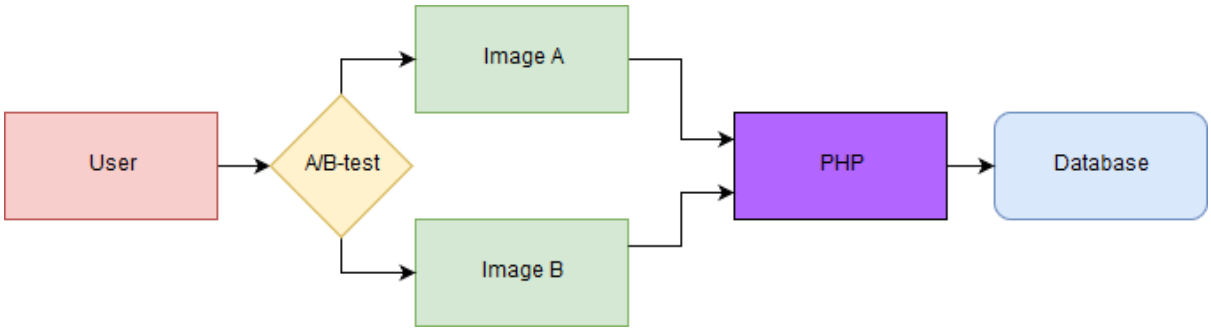

Figure 7 The flow of the split-tests

By using such an application, developers will have an easier time to gather feedback from users and the users can feel a bit more influential with the development process as well. This results in better communication between the developers and users and there is a higher possibility that the users will be more satisfied with the result of the implementations.

What would also be interesting is to add text fields for the images, where a description can explain the changes in a more detailed manner. This makes the usage of the application slightly more interactive as the developers can explain what the changes would consist of. This also allows the split-tests to include things such as new implementations and functionality, as well as changes to design. The tests can consist of either be two new implementation changes or a case where the current design is compared to a new version that might be implemented. Figure 8 displays an example on how an old design and a new design could be compared with split-testing in the web application.

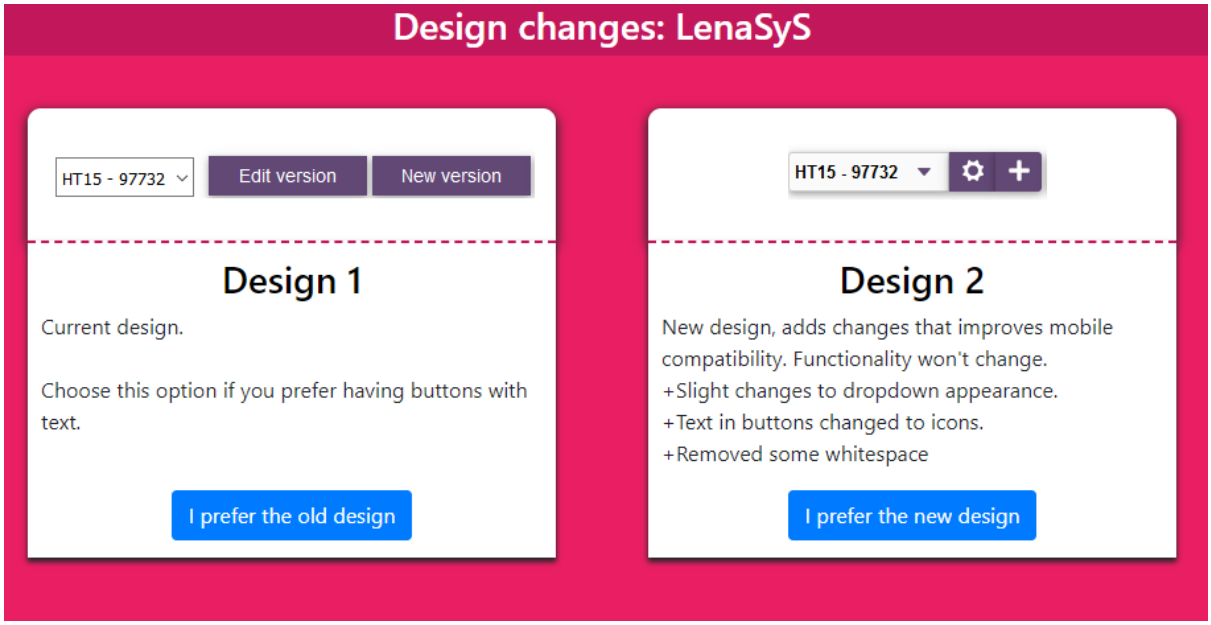

Figure 8 Layout example of the user view in a split-test

3.2.1 Chosen method

The method that is going to be used will experiments to compare the databases against each other to measure response times. Experiment has been the chosen method as they can be performed in controlled conditions, as well as organize events to simulate their usage in the real world (Wohlin, Runeson, Höst, Ohlsson, Regnell, Wesslén, 2012). Wohlin et al. (2012) also mentions two types of experiments that exists, human-oriented and technology-oriented. In this research, the focus will direct to the technology-oriented approach, as no user interaction would be required to measure the databases. By using the technology-oriented approach, the experiment usually applies different tools to objects (Wohlin et al. 2012). In this case, the tools for experiments would be the databases that are connected to the web application.

One of the biggest advantages by going with the technology-based approach is that the experiments would be more controlled and consistent, since humans can behave differently at different occasions, as well as having the results becoming less biased due to learning effects of users (Wohlin et al. 2012).

Although experiments provide a very good environment for controlled measurements, it is worth noting that they are not perfect. While they are a useful method for testing a hypothesis, there are many factors that plays a part while performing the experiments. A lot of individual factors could affect the results of the measurements. In this case, a couple of such issues could refer to internet connectivity, server downtime or processing power of the computer. With this said, we cannot prove that a hypothesis would be correct by performing experiments, but a positive outcome will support its claim (Berndtsson, Hansson, Olsson, Lundell, 2008).

Experiments has been a successful method for comparing databases in previous publications, which influenced the decision to apply the method to this work as well. It becomes easy to run many tests to measure the databases with experiments as the chosen method. Another example of such research is a comparison between MySQL and MongoDB published by Gyorödi, Gyorödi, Pecherle and Olah (2015), where they performed their measurements through experiments.

To efficiently compare MySQL against PostgreSQL, two similar instances of database structure will be implemented in their respective environments. The stored data will be as identical as possible for the different databases to minimize differences of the measurements and application structure. The experiment will first be performed on a baseline with MySQL, where the time it takes for data processing of the images will be measured. Then, the database manager will be switched to PostgreSQL and new measurements can be performed. After the measurements are done, we can analyze the results to see which database would be the better alternative. All the tests and the databases will be used with Windows 10 as the operating system.

To reach the two different databases, some changes will be added to the connection script in the PHP code to match the correct database. This could potentially also add a risk that some deviance will appear in the measurements, since the connection might differ a bit. That's why a pilot study will be performed to see how much these changes would impact the results of measurements. In this pilot study, very small amount of data will be used to avoid having the size of data as a critical variable for the results. With this pilot study, we can see if the databases would differ in processing power before running the main testing with images, as well as see if the connection string might influence the response times.

All measurements will be run through an external JavaScript that will measure the time it takes for the application to handle a database request and print the data. The main measurements that will be performed after the pilot study will show how long it takes for the application to fetch the images used for the split-tests from the database and then print them in the application. Measurements on how long it takes to add the images to the database might also be interesting to see, as it is also a critical part of the application. In the pilot study, the experiment will instead focus on how quick the database can add and store very low amounts of data, such as simple strings of text.

3.2.2 Alternative method

As experiments are not completely reliable, some other methods could be used to aid the research. Another method that was a candidate that potentially could be used to complement the experiments would be a user survey. Feedback from users could be collected and analyzed to see if they can tell which databases would be the fastest. While the surveys do not benchmark response times, a user would be able to distinguish a difference in loading times between databases if the response times would have a high enough deviation.

However, this method was discarded due to multiple reasons. Using user surveys purely for detect differences in response times would not be optimal, as it does not provide enough data for the research. Experiments already perform measurements at a much more precise level, with a higher capacity. At a smaller scale database, the response time of queries would most likely not even differ enough for a user to reliably tell the difference, which could potentially result in some wrong statistics as well.

The final issue that occurs regarding user studies is the design of the application. The databases would be initiated locally, which would prevent users from participate in such a study of their own computers. For them to participate in the study, the overall structure of the application would have to be setup on a web server, which would make a huge impact of both the benchmarking results and application structure.

3.3 Ethical aspects

As the plan is to not include any user studies, it does removes the concern of any potential breach of sensitive information about the users. The application itself will involve the statistics from the answers of the split-testing, but as this only includes preferences of design and no personal information, there should not occur any ethical problems. The split-testing in the tool will be completely anonymous, as the chosen options in the split-tests is the only data that is a point of interest.

Beyond that, the probability is big that these statistics will be automated during the research, instead of having real user inputs. The reasoning why automated statistics could be applied is that the user inputs of split-tests are not the point of interest in this research. Instead, the focus lies within measuring relational databases and how they perform with image processing.

The images that will be used in the application will consist of either found images or existing web pages, where some changes have been made to their design. Those changes will be performed by tweaking the CSS code of the web pages through the console in the Firefox browser. After those tweaks are made within the console, a print from the site is saved where the appearance has been changed. For example, the design of Google's index page was changed through the console to add a new color scheme, as seen in figure 9.

Figure 9 Example of a design change for Google

One last important thing to consider when performing the measurements is that technical experiments are not 100% accurate. There might be some variables that could affect the results a bit. However, as those issues would most likely affect the databases in the same way, it should not cause any major concerns. The critical result of this research is which database operates best in this scenario, and not specifically how fast they would do it.

4 Implementation

This chapter summarizes the process of implementing the application and databases, as well as walking through the pilot study that was setup to see if the experiment was possible to perform. It consists of three subchapters, *research*, *progression* and *pilot study*.

The research chapter involves the knowledge that was required to implement the tool and where such information was found. The progression chapter walks through the process that was followed to implement everything for the application, as well as some design changes that had to be made during the process. The pilot study presents the first test that occurred to see if the chosen method was possible to use to measure the databases, as well as presenting the results it provided.

4.1 Research

Most of the knowledge on how to implement this tool was learned at the University of Skövde. The course *Webbprogrammering* included lots of information on how to use implement a single-page application, as well as some essential knowledge with JavaScript, jQuery and AJAX. The courses *Databassystem* and *Databaskonstruktion* provided all the necessary knowledge on how to create a MySQL database and connect it to a PHP page to access and insert data through forms.

The PostgreSQL database that was installed for this application needed some research to implement, as it was an assignment which required some information that was not known initially. Specifically, how to setup the server and connect the chosen workbench to it. To find the required configuration, an installation guide was followed. This guide was a part of the PostgreSQL documentation (2018) found on the official web page for PostgreSQL, which was followed to some extent to setup the PostgreSQL database.

Regarding the databases, some important considerations had to be considered. Based on the statements of several experienced developers on internet forums, such as Stack Overflow, image storage in databases is not really a preferred method to choose generally. This is mostly because as the images will take up a lot of storage space in the databases.
Instead, some users recommended storing the image URL's and link them to a disc, such as a hard drive, mostly to save database storage (*Brandon Wood & giorgi,* Stack Overflow, 2009). While this project's chosen approach for this application might not be the optimal way to go, it does however make the research a whole lot more interesting at the same time, since not many people have been performing experiments with images as binary data in databases. This research will as a result provide some interesting information regarding binary image storage and how well the databases can process them.

The storage space issue the users refer to will not be of any major concerns in this research, as the databases used will contain relatively small amounts of data. Those who oppose storing images in databases mostly refer to scalability issues when mentioning image management, as large databases would be filled extremely quickly.

This would not be a relevant problem in this research, as the range of data that the implemented databases would require for the study is relatively low. For example, in the pilot study, the databases would only require a single split-test containing two images to perform the first experiments to measure the processing. The main measurements performed at a later stage will contain some scalability of stored data to see how big of an impact in overall data processing a larger database with images would make.
To successfully store the images in the databases, some instructions that was found online was followed. The initial setup for the application form and inspiration for the database management came from a written guide online (Jain, M., Talkerscode, 2018), which walked through the basic process, as well as describing the difference between BLOB datatypes. Based on the information found, *mediumblob* was chosen, as it was deemed the most appropriate type to use for the images.

4.2 Progression of implementation

This chapter will go through the process that was used when implementing the application and databases, where the subchapters will describe the different components that was required to setup for the pilot study, as well as the measurements in the main experiment. Hashes marked with # in this chapter are the name of commits from the GitHub repository that are being referred to when changes in the code are being mentioned. The repository itself is found at https://github.com/a15tobli/Exjobb.

4.2.1 Web page structure

The first thing that was implemented was the web page of the application and its layout, #c8b7929. The application required different views for the content where navigation is being used. An important choice was made at this point, whether the application should become single- or multipaged. The choice was made to make the application single page, where a single index PHP file is used to navigate through the content, and making the visibility of the content dynamic with the help of jQuery in conjunction with navigation links. Instead of redirecting the user, the links will instead manage visibility of content of the page, #486ee0f.

The biggest reason to why a single-page application was chosen over multi-page is to reduce loading times when navigating through the application. This becomes especially important as jQuery is used within the project, which includes a huge library that takes quite some time to load. Appendix A and B contains the code used for the application structure and styling.

Figure 10 contains some of the code from the Navigation script, appendix G, used to manage the visibility of page content with JavaScript and jQuery. When a link is clicked, the function *setContent* is run, which first clears the shown content, then a string of the active link is sent as a parameter and is used in a switch to display the correct page content based on the click. The case “Home” is the default view, which is seen initially when the page is opened.

```
//Displays content based on link clicked
function setContent(clickedLink) {
    //Hide contents
    $("a").removeClass("active");
    $(".indexContent").hide();
    $(".splittestContent").hide();
    $(".formContent").hide();

    //Display correct content based on active link
    switch(clickedLink){
        case "Home":
            $("#homeLink").addClass("active");
            $(".indexContent").show();
            document.getElementById("title").innerHTML="Home";
            break;
        case "AnswerTest":
            $("#answerLink").addClass("active");
            $(".splittestContent").show();
            document.getElementById("title").innerHTML="Design comparison";
            break;
```

Figure 10 Navigation with JS

The outcome of the single-page design choice is quite relevant as it will define how the forms that sends data to the databases will function. Normally, posting a form would send a user to another page created by a PHP script that runs the code used in the post, or possibly reloading the current page. As this application is single-paged, those options would not work. This is because the application would be changed back to the home view as soon as the page is reloaded, instead of staying at the current view containing the displayed form. To avoid this, AJAX was added to make the forms send data to different PHP-scripts to perform functions and still has the user staying on the same page.

4.2.2 Database structure

The structure of the tables was an interesting topic to consider while implementing the databases. The big issue was how to store a split-test and its corresponding images with the insert form. The first option was to use a single table, containing all the information. The second option was to use two separate tables which have a relation, *SplitTest* and *ImageEntry*,. As seen in appendix C and D, the final decision was to split the data into the two different tables in the databases, the initial commit with the final structure in MySQL is found in #5c81023.

If a single table would have been used, some of the written code for the application would not have been necessary, as there would be no need to insert a test and retrieve its ID before inserting images. Overall, the structure would probably be smoother and a bit easier to execute by choosing this option. However, a big factor played a role on this decision: the thesis of the research. The thesis states that the measurements would be performed on image processing. This is where the single-table structure contradicted with the thesis, as then the measurements would be applied to an entire split-test instead of just the image entries themselves. That is why the solution was to use one table for images and one for tests as implemented instead, as it was considered more relevant for the topic of the research.

As both databases runs SQL (Structured Query Language), the syntax of their structure is very similar. One noticeable difference is the *image* column of the *ImageEntry* table. This column is the most essential one of them all, this is where the images are stored as BLOB-format in their respective DB environments. MySQL uses *mediumblob* as the variable, where PostgreSQL instead uses the *bytea* variable. Both are quite similar in the regards of storing BLOB-data, yet requires some different encoding to successfully be stored and retrieved. Figure 11 and 12 shows a comparison in the SQL used to create the tables for the databases. The implementation of the PostgreSQL tables is found in #62cbc96.

MySQL

```
CREATE TABLE SplitTest(
testID int AUTO_INCREMENT PRIMARY KEY,
testName varchar(25) UNIQUE NOT NULL
)engine=innodb;

CREATE TABLE ImageEntry(
ID int AUTO_INCREMENT PRIMARY KEY,
image mediumblob,
caption varchar(100),
testID int,
FOREIGN KEY (testID) REFERENCES SplitTest(testID)
)engine=innodb;
```

Figure 11 Tables in MySQL

PostgreSQL

```
 --Create tables
CREATE TABLE SplitTest
(
    testID SERIAL PRIMARY KEY,
    testName varchar(25) UNIQUE NOT NULL
);
CREATE TABLE ImageEntry(
    ID SERIAL PRIMARY KEY,
    image bytea,
    caption varchar(100),
    testID int,
    FOREIGN KEY (testID) REFERENCES SplitTest(testID)
);
```

Figure 12 Tables in PostgreSQL

The initial setup of PostgreSQL was a bit tricky, mostly because the server connection refused to work properly with the database. Different workbenches were used to see which was the easiest to set up the database in. The workbenches that was tried were DBeaver, SQL workbench and PGadmin. PGadmin was used initially, but it seemed a bit inconvenient to work with at first. That is why DBeaver and the SQL workbench, which seemed easier to work with, was attempted to use instead. However, the connection to the PostgreSQL server was never made to work in any of those applications. This part of the implementation halted the workflow by having some time spent attempting to find the error, without any result. Eventually, the decision to keep using PGadmin was made, as the connection worked flawlessly without any further complications. Besides the initial connectivity, no issues were stumbled upon with PostgreSQL, since the process was very similar to implementing the MySQL database.

4.2.3 Add A/B tests

Due to how the database was structured, some additional steps while working with the queries was added to make the insertion of data work, #0ae04bd includes the implementations for the insertions. As seen in figures 11 and 12 above, the tables for images requires a *testID* to be added. The ID is incremental, and is automatically signed a value for every entry that has been inserted in the *SplitTest* table. Because of this, a test must always be created before image inserts. In the application itself, a single form is used to add all the required data for every test. As seen in figure 13, a name for the test, captions and images are sent with the form. Figure 14 displays how this data is assigned to different variables in PHP after being retrieved from the AJAX post of the form. The AJAX also sends an additional variable, *activeDB,* based on the active database connection to control the correct flow of the PHP code.

The PHP code for handling database interactivity is divided into several files. Two classes were created, one for insertions and one for selections from the database. *Viewdata.php*, appendix L, handles all the fetch queries while *insertData.php*, appendix I, was used to insert data. Another file included in appendix J, *insertTest.php*, is the main file used to call on the functions involved in adding new tests. The following subchapters in 4.2.3 further explain the occurring process of insertions.

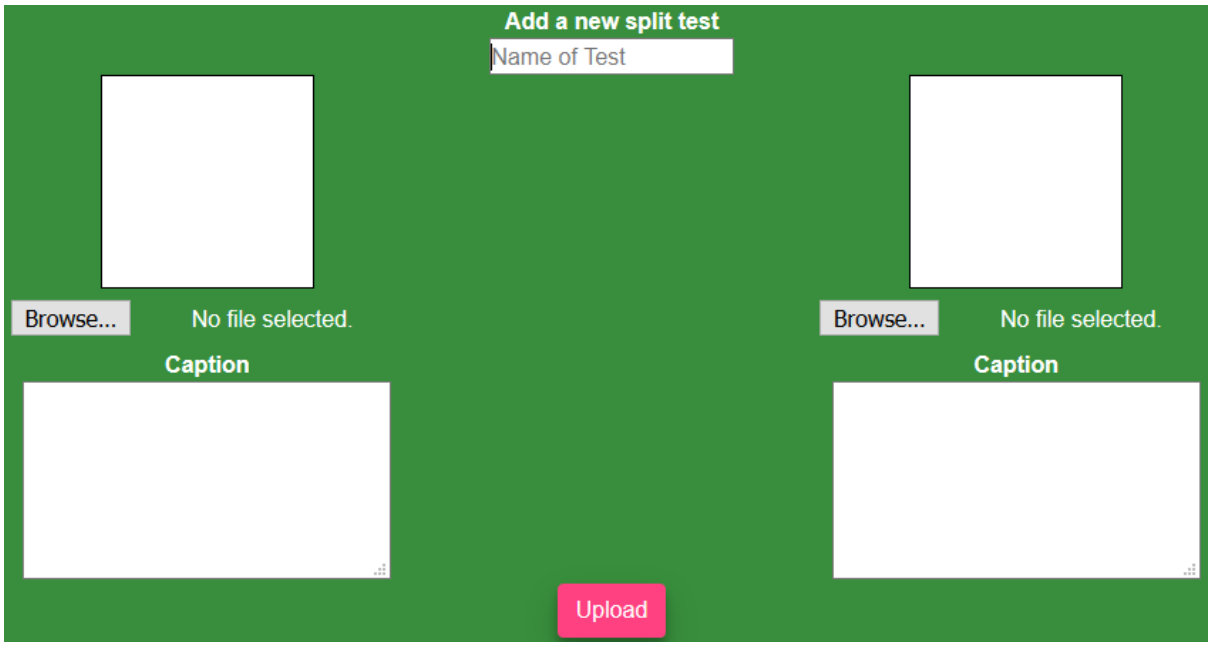

Figure 13 Insert form of the application

```
//Form data retrieved from Ajax
    $testName = $_POST['testName'];
    $img1blob = $_FILES['image1']['tmp_name'];
    $img2blob = $_FILES['image2']['tmp_name'];
    $caption1 = $_POST['caption1'];
    $caption2 = $_POST['caption2'];
    //Connection type
    $activeDB = $_POST['DBcon'];
```

Figure 14 Form data in PHP

4.2.4 Select testID

As soon as the form is submitted, the required fields are validated. If a test name and both images have been set, the *insertTest* function is run, where a new Split-test is inserted into the database, see figure 15. A returned *boolean* value is used to determine a test name already exists. The boolean halts the rest of the insertion code for the form from running of the test was not successfully inserted into the database, i.e. when the returned Boolean becomes false.

```
$sql = "INSERT INTO SplitTest(testName) VALUES ('$testName')";
$insertQuery = $PDO->prepare($sql);

//If test name already exist in database
if(!$insertQuery->execute()){
    return false;
}
return true;
```

Figure 15 Insertion of a Split test

If the insertion was successful, the PHP code jumps to the next function, *getTestID*. This is where the ID of the split-test is fetched and will later be used for insertion of images. The *testName* variable of the inserted test is used to fetch the correct ID in the database, displayed in figure 16. An interesting issue turned up at this point. While retrieving data from PostgreSQL, all the characters had to be lowercase for the column name, even though the column is known as '*testID*' where 'ID' is uppercase. Because of this, a selection had to be added that converts the column name in the code to lower case to successfully fetch the data from the PostgreSQL database.

```
$fetchQuery = $PDO->prepare("SELECT testID FROM SplitTest WHERE testName='$testName'");
$fetchQuery->execute();
$row = $fetchQuery->fetch();

//Postgres requires small chars
if($activeDB == 'mySQL'){
    $ID =$row['testID'];
}else if($activeDB == 'pgSQL'){
    $ID = $row['testid'];
}

return $ID;
```

Figure 16 Select testID from database

4.2.5 Insert image

When a testID has been retrieved, the focus turns to the images. As images cannot be stored as they are into databases, a function that converts them to BLOB-format was added to encode them correctly first. As the databases have different variables that determines how BLOB-data is stored, a database selection also appear in this code, *addslashes* is used for MySQL, and *pg_escape_bytea* is used for PostgreSQL, as seen in figure 17.

The solution for the file reading and conversion to binary data for the databases in the *convertImage* function received inspiration in a response on a Stack Overflow thread (*SomeKittens,* Stack Overflow, 2012). The function parameter *tmpimage* in the function is a variable that retrieves the images from AJAX data that the form sends.

```
//Read image and return correct data to store into DB
function convertImage($tmpimage, $activeDB){

    $fp = fopen($tmpimage, 'r');
    $data = fread($fp, filesize($tmpimage));

    //Different file handling for db's
    if($activeDB == 'mySQL'){
        $file = addslashes($data);
    }else if ($activeDB == 'pgSQL'){
        $file  = pg_escape_bytea($data);
    }

    fclose($fp);

    return $file;
}
```

Figure 17 Image conversion

As soon as the image is converted correctly, the *insertImage* function can be called upon. The converted image, text caption and *testID* is inserted as parameters, and the query for inserting images is executed, where everything is stored into a row.

```
//Query for inserting images into database
function insertImage($tmpimg, $tmpcaption, $tmpID, $activeDB){
    if($activeDB == 'mySQL'){
        require "MySQLcon.php";
    }else{
        require "PostgreSQLcon.php";
    }

    $query = "INSERT INTO ImageEntry(image, caption, testID) VALUES ('$tmpimg', '$tmpcaption',
    '$tmpID')";
    $insertQuery = $PDO->prepare($query);

    if(!$insertQuery->execute()){
        echo ("Error while inserting images!");
    }
}
```

Figure 18 Function *insertImage*

To call on the insertion efficiently, another function, *submitForm,* was implemented. This allows two images to be inserted at the same time while avoiding redundant code for the queries. As figure 19 displays, the variables that are being sent into the function as parameters are still the same that is retrieved from the post, shown in figure 14.

```
//Adds images and captions linked to a split-test to the database
function submitForm($testID, $img1, $caption1, $img2, $caption2, $activeDB){
    self::insertImage($img1, $caption1, $testID, $activeDB);
    self::insertImage($img2, $caption2, $testID, $activeDB);
}
```

Figure 19 Image submission

To summarize the process of insertions, first the form is filled in the application. Then, validation occurs and a row in the Splittest table is created, where a testID is generated. After that, the chosen images are converted to binary form and finally inserted into the databases as two separate image entries in the database, linked to the testID.

4.2.6 Search form

Once the split tests could be inserted into the databases, the second form of the application was implemented. This form is where the searches for the split-tests are performed, and it is the form used where the experiments are conducted. The form contains a text field, *searchData,* and a submit button. When the button is clicked the function in figure 20 runs and an AJAX request is sent, #32eda2d. The database connection and value of the search field is sent into the *searchTest.php* script, included in appendix K.

The PHP script returns an array with the images if a search was successful, as seen in the success function of the AJAX post. Two image elements of the application (*img1*, *img2*) will have their sources set as the images that was retrieved from the database. The *startTime* and *timeDiff* variables will benchmark the time that the test took to retrieve from the database, counted in milliseconds. The time difference is then sent into the *sendBenchmark* function.

```
//Search form for finding tests
$("#searchBtn").click(function(){

    //Benchmarking variable
    var startTime = (new Date).getTime();

    $.ajax({
        type: 'POST',
        url: "./PHP/searchTest.php",
        cache: false,
        data: {
            value: $("#searchData").val(),
            activeConnection: DBtype
        },
        success: function(data){
            //If search results were found
            if (data[0] !== undefined && data[1] !== undefined) {

                //Returns images based on search result
                $("#img1").attr("src", data[0]);
                $("#cap1").html(data[1]);
                $("#img2").attr("src", data[2]);
                $("#cap2").html(data[3]);

                //Save testID to use when sending answers
                activeTestID = data[4];

                //Control visibility of displayed content
                $(".captionRow").show();

                //Benchmarking result
                var timeDiff = (new Date).getTime() - startTime;
                sendBenchmark(timeDiff, DBtype);
            }
```

Figure 20 Handling search form data with JavaScript

The time difference will be written into a separate text file for each database with the help of PHP, where the *sendBenchmark* function is used to successfully send the benchmark value to the PHP script. To do this, another AJAX request is being used, where the *timeDiff* variable and database connection type is sent as data to the PHP script *writeResponsetime.php*, attached in appendix N.

```
//Send benchmark timers to DB
function sendBenchmark(responseTime, DBtype){
    $.ajax({
        type: 'POST',
        url: "./PHP/writeResponsetime.php",
        cache: false,
        data: {
            benchmark: responseTime,
            activeConnection: DBtype
        },
        success: function(data){
            console.log("Timer saved: " + data + "ms");
        },
        error: function(exception){
            console.log(exception.responseText);
        }
    });
}
```

Figure 21 Sending benchmark variable to PHP

The PHP file will then fetch the variables from the second AJAX post, as seen in figure 22, and select the correct file for logging benchmarking values based on the database connection. Then the script loads the content of the file and then writes the new benchmark value in milliseconds from the latest search result as a new row.

```
<?php
$time = $_POST['benchmark'];
$activeDB = $_POST['activeConnection'];
if($activeDB == 'mySQL'){
    $file = "../Benchmark/mySQLresults.txt";
}else{
    $file = "../Benchmark/pgSQLresults.txt";
}

//Read file
$content = file_get_contents($file);
//Write new times to file
$content .= $time . "\n";
file_put_contents($file, $content);

echo $time;
?>
```

Figure 22 writeResponsetime.php script

4.3 Pilot study

Once the implementation of the search form and database entries was finished, the next step was to perform the initial test searches. Two JavaScripts was written and used with the *Greasemonkey* Firefox plugin, which was downloaded from the official site (Greasespot, 2018) to simulate image searches in the application while it is open in Firefox, appendix M and N contains the scripts used for this purpose.

These scripts would handle data generation for the form for adding new split-tests, as well as an automated search script to find them. To make sure that the tests would be as consistent as possible, the databases was both wiped clean prior to this and both contained a single split-test, named *Test1*, where identical images was inserted for both the databases.

The data generation script could not fill all the required data. Instead, the text fields were filled and an incremented name was generated in the fields when the link leading to the form was clicked. This made the data generation a bit simpler, as only the images were needed to be manually selected for every test that was being added.

The search script was designed to automatically perform searches for the split-test based on a set integer as soon as the link leading to the search view was clicked. For the pilot study, the integer was set to 10. The *fillForm* function was tthen iterated 10 times with intervals to send a single search request at a time. Without a delay between the searches, the function could affect the result of the searches while trying to make multiple searches simultaneously. That is why a 1-3 second delay was set to perform a single iteration at a time. As the searches only took numbers of milliseconds to perform, a single second as interval usually was more than enough time to perform a search. But as a search could potentially take a bit longer time to perform, an extra margin of up to additional 2 seconds was added for some cases to only perform a single search at a time.

All the implementation for image retrieval, mentioned in chapter 4.2.4, was run as soon as the script performed a search. This resulted in 10 entries of search times stored into the log file for the current database. Just to make sure that the results would generate similar times, a second iteration of the search script was run, resulting in 20 benchmark values. Then the script was run twice again for the PostgreSQL database, which filled the other log file with 20 values.

Figure 23 displays the script for automated searches in the measurements, where the *counter* variable iterates the code in the function *fillForm* if it is below 10. The *setTimeout* function is where the 3 second delay is added to prevent multiple searches at the same time. The value of the *searchData* element is where the name of the active split-test to find is specified.

```
$(document).ready(function(){

    console.log("Script successfully loaded");

    //Initialize script by clicking on the link
    $("#answerLink").click(function(){
        var counter = 0;
        fillForm(counter);
    });
});

function fillForm(counter){
    //Run 10 searches with 3 second intervals
    if(counter < 10){
        setTimeout(function(){
            counter++;
            console.log(counter);
            $("#searchData").val("Test1");
            $("#searchBtn").click();

            fillForm(counter);
        }, 3000);
    }
    else{
        console.log("Measurements are finished");
    }
}
```

Figure 23 Search script with Greasemonkey

Based on the results that was stored in the log files, some statistics were retrieved in Excel by using analysis tools such as ANOVA. The average response times was produced and displayed in the chart in figure 24. What was interesting to see was the PostgreSQL was performing a lot faster when searching for images compared to MySQL. MySQL retrieved the images of the searched split-test in ~789ms at an average, with a deviation of 58ms, while PostgreSQL has benchmarks as low as ~124ms average, with a deviation of 50ms. All the search results were quite consistent, with a relatively low variation, as no spikes in the results occurred during benchmarking.

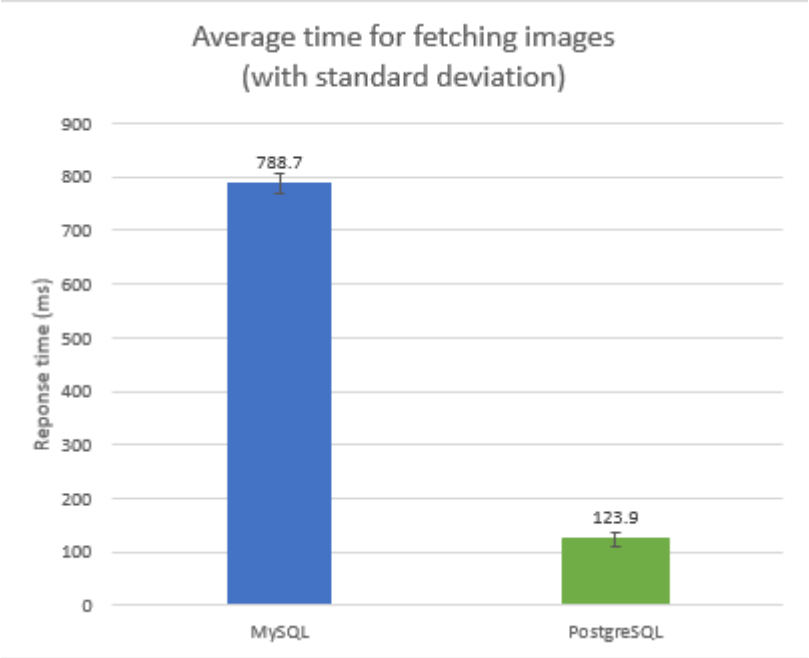

Figure 24 Benchmark result of pilot study

The result of the pilot study claims that the hypothesis, which stated that MySQL would be the better option for image processing, was wrong. However, it is not possible to conclude that as a fact purely based of this pilot study. Additional measurements need to be performed with different data sets, such as changes to file sizes or database variables to see if the results would differ due to those circumstances. This study will be further expanded into a larger experiment to try out more variables, which will be conducted in the next chapter.

5 Evaluation

The results of the pilot study implied that PostgreSQL would be the better option compared to MySQL for image processing in databases. However, the databases only contained a single test with 2 different JPG images. The next step in the research is to insert more entries in the databases, as well as using different types of data, such as file formats, file sizes or resolutions for the stored images. This would provide more accurate results compared to the pilot study, as several factors can be tested to see if the databases can process some variables faster than others.

After all the necessary data has been inserted into the databases, the main measurements can begin. Those measurements would cover more variables and being run at a much larger scale than the initial 20 searches from the pilot study. This would result in much more precise benchmarking results for the image processing in the application, as well as covering different variables concerning the stored data.

5.1 Presentation of research

The research was performed with a Windows PC and was chosen over a work laptop, to provide as accurate readings as possible, since the components would be a lot more efficient. Some hardware and the required software used in the measurements are listed in the table below.

Hardware	**Specification**
CPU	Intel Core i5 6600k @ 3.50GHz
RAM	8GB @ 1199MHz
Storage	500GB Samsung SSD
Motherboard	ASUSTeK COMPUTER INC. Z170-P
Software	**Version**
MySQL Workbench	6.3
PGadmin	4
XAMPP	3.2.2
Mozilla Firefox	59.0.3

Table 2 Specifications of required components

A variety of test cases was created for the experiment. Each case had some unique variables where 5 split-tests of the type was stored into the databases. 5 Different test cases were constructed to use, as seen in the table below.

Test case	**Information**
1: JPG – Small Images	Small images at ~60kb Resolution: 1920x1080px
2: JPG – Large Images	Larger images at ~1.7mb Resolution: 1920x1080px
3: PNG – Large Images	Larger images at ~1.6mb Resolution: 1920x1080px
4: PNG – Doubled Data	Same images as test case 3. Doubled the contents of the databases.
5: HTML	Very small HTML file at ~6kb

Table 3 List of test cases

First, two cases with JPG files were created to see if file sizes would determine benchmarking results when comparing the databases. The cases are referred to as small and large images, due to their differences in file sizes. These two cases were created to see any differences in response times with varying file sizes of the same image format.

Then, a test case with large PNG images was added to see how the databases would handle a different file format with similar size to the large JPG files. These images were as close to the large JPG images as possible in size to provide a consistent comparison of image processing between the different file formats.

The fourth case was created to test the amount of data stored in the database. The test case itself has the same variables as in case 3, with the exception that the amount of data in the databases was doubled, from 5 stored A/B-tests for every case to 10.

The final test case was determined as a quite interesting topic to measure the processing of; HTML files. This case would provide two interesting variables to measure: very small file sizes, as well as a stored file format that was not an image. The results of HTML processing could also provide some unique data that could be used in conjunction with future work, such as logging changes through stored HTML files. The amount of data in the databases were the same as in case 1-3, i.e. 5 split-tests stored per test case in the experiment.

Each test case was run 1000 times for each database where the first stored A/B-test for every case was being used when performing searches for the benchmarks, where the stored A/B-tests contained two identical pictures to provide the same processing times within the test. Performed experimentations before the main measurements showed that iterating a single chosen split-test did not affect the benchmarking results when compared to differing between different tests within the same case for the searches.

Test case 1: JPG – Small Images

The initial test was performed to see how the databases would compare when using relatively small files in JPG format. 1000 Tests were run where a specific A/B-test with the set variables of the case was searched for.

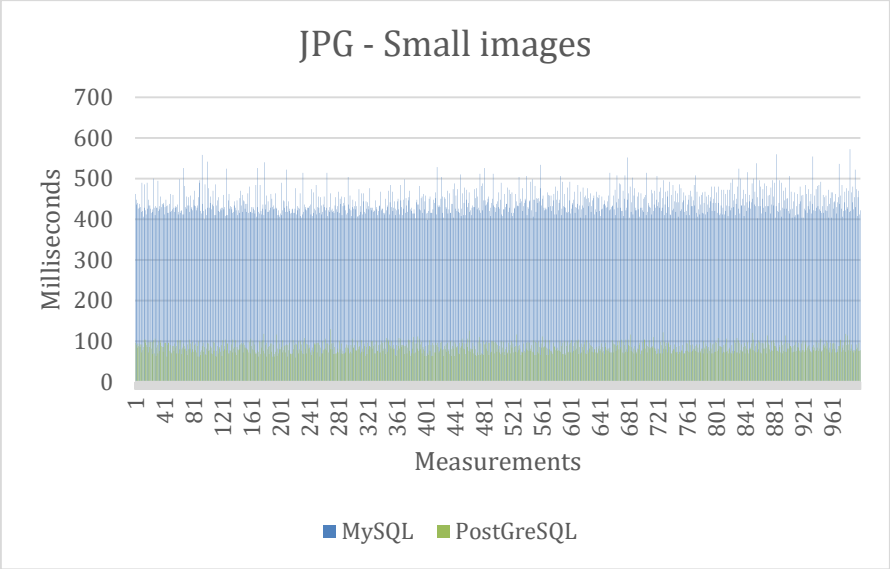

Figure 25 Case 1: Benchmark values

As seen in figure 25, the timers were quite consistent. Some minor spikes occurred with the MySQL searches, but they were not significant enough to affect the results. Figure 26 contains the average benchmarks with standard deviation for the first 1000 tests in each database.

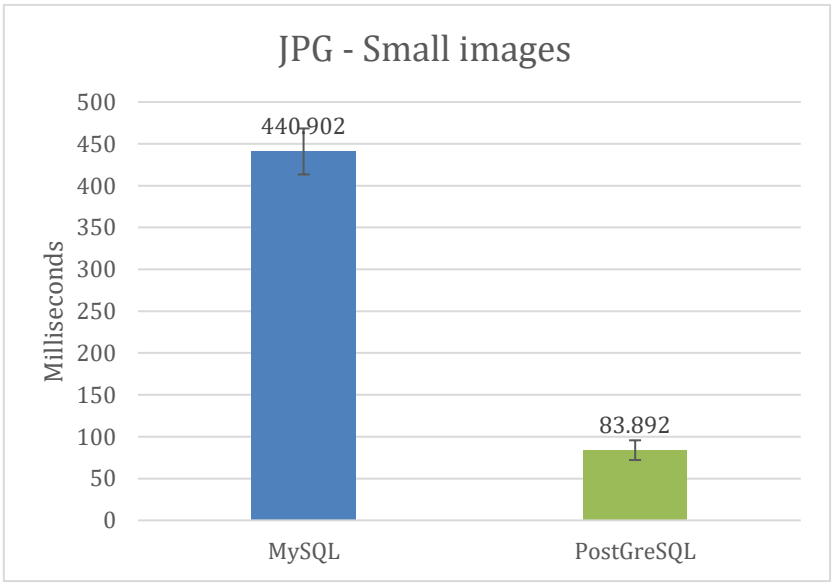

Figure 26 Case 1: Average response times

Test case 2: JPG – Large Images

The second test case was very similar to the first one, with the only changed variable is the file size of the images that was used in the searches. To see if any differences in the response times would occur with larger files, JPG images with the size of 1.7mb were used in the searches.

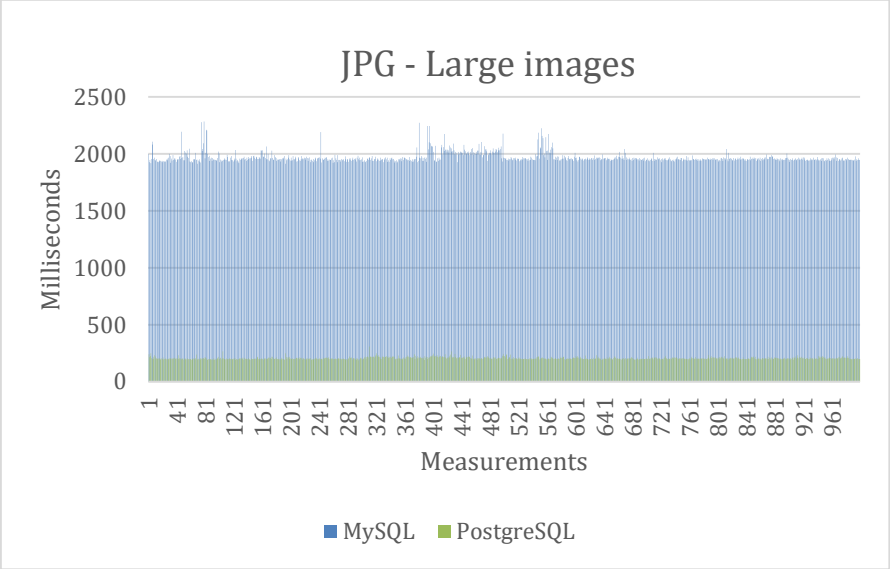

Figure 27 Case 2: Benchmark values

Just as in the first case, PostgreSQL performed a lot faster processing when retrieving the images. Compared to the first test case, PostgreSQL had the response times just a bit above doubled, from ~84ms to ~207ms. MySQL received a delay of about 4 times the initial tests however, increasing the average time from ~441ms to ~1968ms, as seen in figure 28.

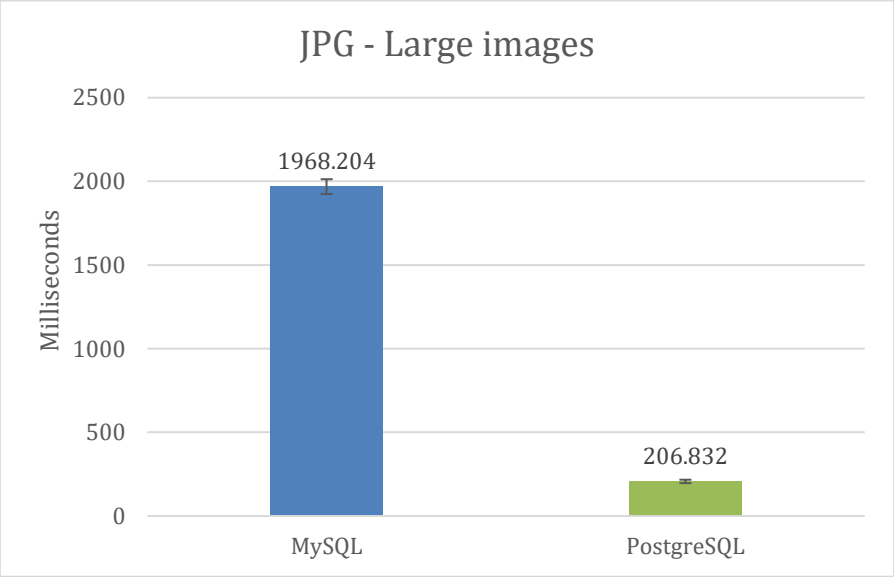

Figure 28 Case 2: Average response times

Test case 3: PNG – Large images

The next case measures how the databases would process another file format, PNG. Images that were as close to the second test case in sizes was chosen to provide as consistent variables as possible. The used PNG files was only about 0.1mb bigger compared to the large PNG images, but at was deemed as close enough in file sizing to provide relatively consistent results when compared to large JPG.

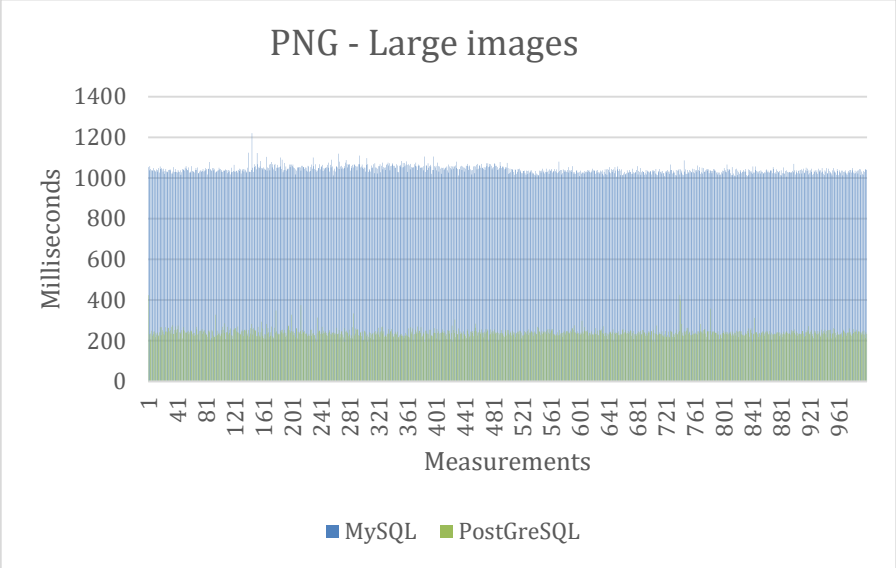

Figure 27 Case 3: Benchmark values

When the image type was changed to PNG, some interesting differences occurred compared to test case 2. PostgreSQL had a ~20ms increase on average, which was a negligible amount because of the increase in file size compared to the large JPG image that was being used in previous case. MySQL did however see a significant increase in performance with PNG files, lowering response times with ~900ms on average when compared to large JPG images.

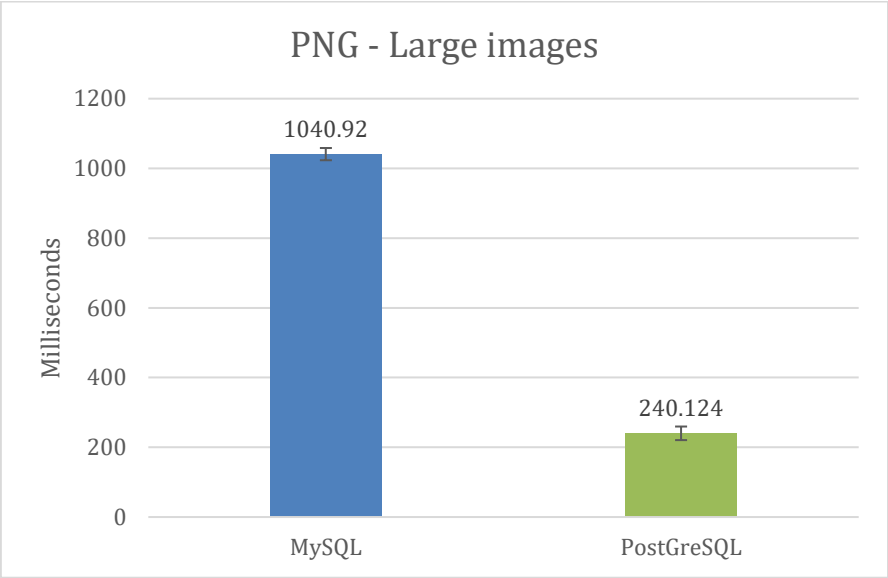

Figure 28 Case 3: Average response times

Test case 4: PNG – Doubled data

For this case, a new variable was measured: the stored data in the databases. The number of split-tests stored in each database was doubled to see if it would impact the results of image processing with large PNG images.

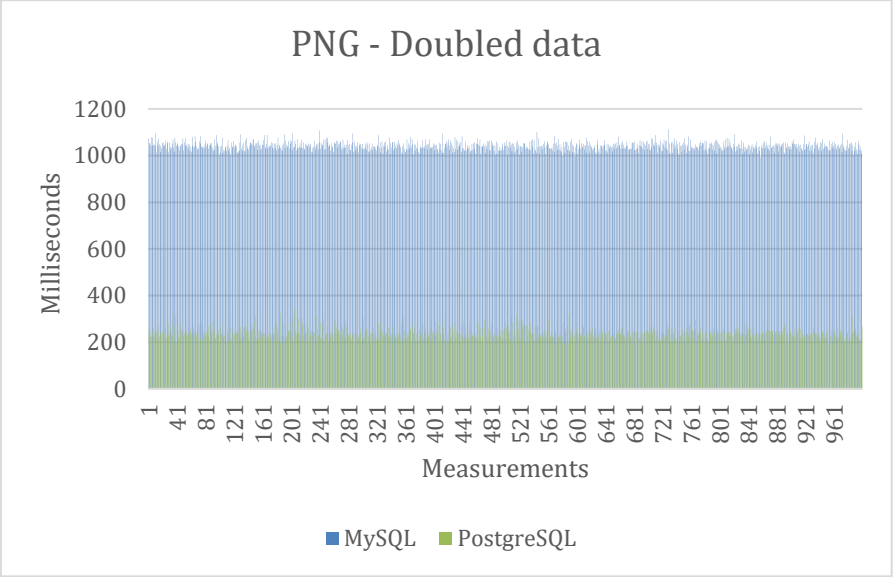

Figure 29 Case 4: Benchmark values

The result of the case was that there is no significant difference in the benchmarks. Compared to test case 3, the results became very similar, with just a few milliseconds in difference. The differences of each database were not consistent, which implies that the amount of data was not a factor that affected the results. MySQL performed a single millisecond worse, while PostgreSQL had a 4ms improvement. As the differences were such low, we can see that the results were not directly affected by the amount of stored data.

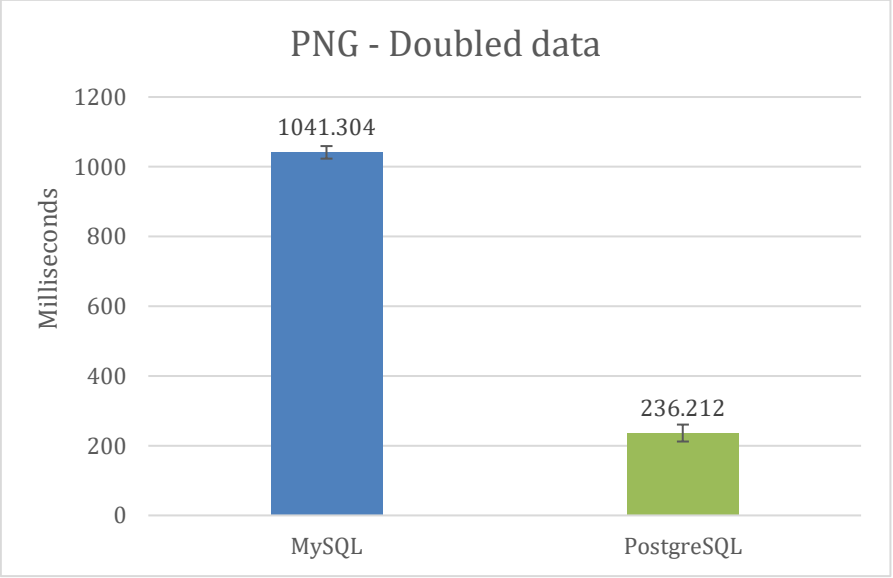

Figure 30 Case 4: Average response times

Test case 5: HTML

This test case was the most unique one. It introduced two new variables for the experiments: very small file sizes (~6kb) as well as using HTML as the file format. This adds some new factors compared to the previous tests, which could provide some unique results of the benchmarks.

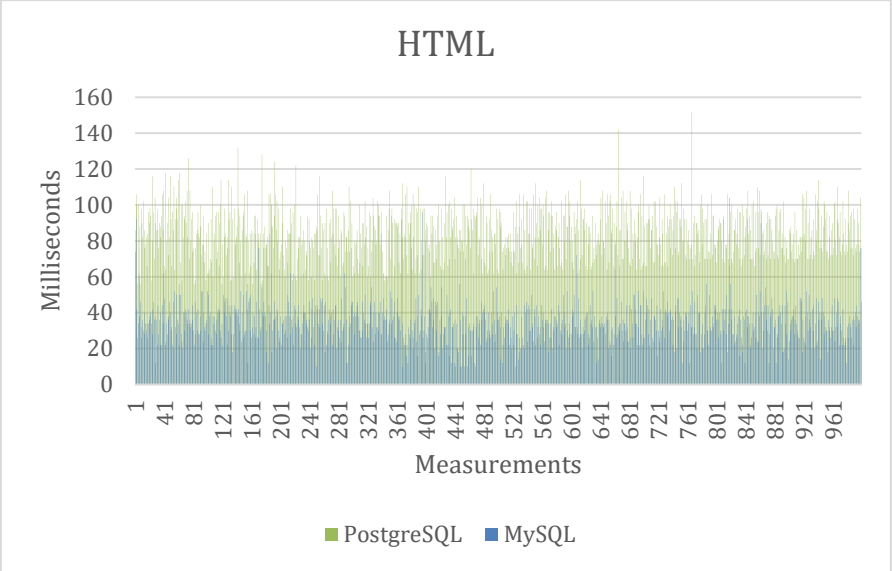

Figure 31 Case 5: Benchmark values

Both databases were very efficient at processing this format. What was interesting to see in the results is that MySQL actually did perform a whole lot faster than PostgreSQL in this case. This was the only case where MySQL had better response times when compared to PostgreSQL, as shown in figure 32.

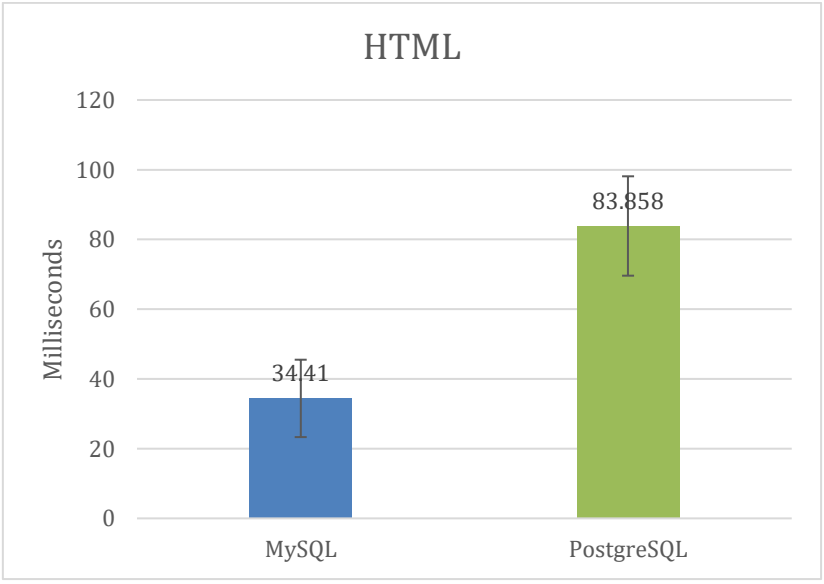

Figure 32 Case 5: Average response times

5.2 Analysis

All the average benchmarks have been composed into a single graph for each database. With the diagrams, we can clearly see that PostgreSQL performs at a much faster for the cases where the images have been searched for. The fifth test case where HTML is compared is the only exception, but since it does not reflect research based on the thesis, it does not provide relevant information for this analysis. Instead, a discussion of possible future interactions based on the HTML experiment will be mentioned in chapter 6.3.

In the research by Stancu-Mara and Baumann (2008) they concluded that PostgreSQL is only marginally affected by file sizing and overall shows a pretty regular behaviour. This statement reflects very well with the performed experiment, as the PostgreSQL shows relatively consistent response times throughout all the test cases. One of the most interesting factors was the change from large JPG to large PNG, test cases 2 and 3, where almost no difference in response times occurred with PostgreSQL. At the same time, MySQL included a drastic change of a second for the image processing, as seen in figure 33.

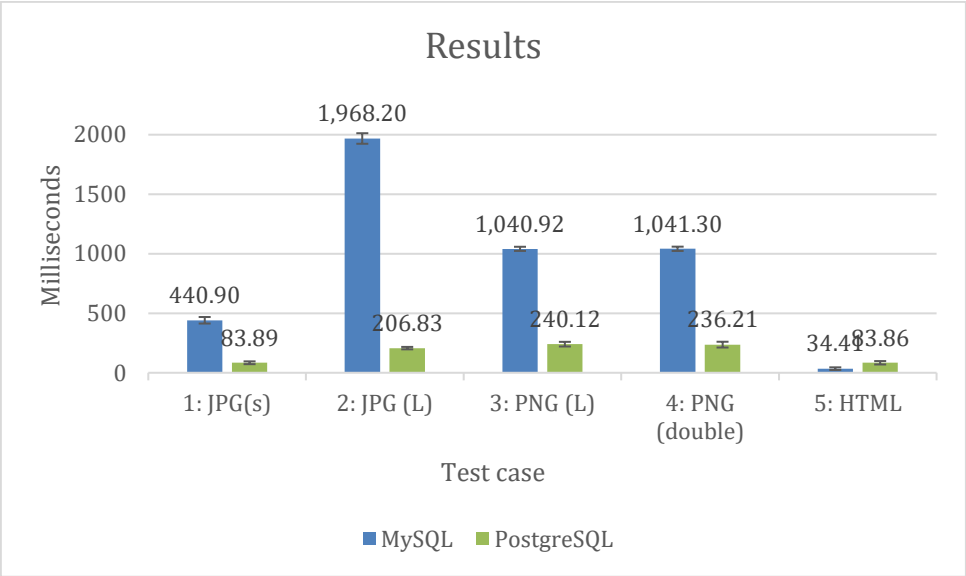

Figure 33 Average response times

Overall, PostgreSQL was shown to be the most consistent database, as well as the faster option. When looking back at the hypothesis, *MySQL will perform image processing faster than PostgreSQL,* it is apparent that it was proven wrong. 3 of the cases showed that PostgreSQL was about 4 times faster than MySQL, which proves it to be much more successful at processing the images when compared to MySQL. This shows especially well for the case with large JPG files, where MySQL took more than 2 seconds to load a single image, which is not regarded as sustainable in the long run, due to the limited user experience with slower response times. MySQL performed almost 10 times slower in that case when compared to PostgreSQL.

There is not much certainty of exactly why PostgreSQL performs much faster when fetching images compared to MySQL, as the subject has a lack of previous research. One speculation is that how the different BLOB files are converted plays a big role. As the databases requires slightly different encoding to retrieve the files, it does have an impact of the response times. It is possible that the method used for MySQL converts at a much lower rate compared to the PostgreSQL encoding, which affects the results of the experiment.

6 Conclusions

This final chapter provides a summary which walks through this research and the results. Its purpose is to provide a quick overview of what was conducted in the study. This chapter also discusses the conclusions of this study, which will bring light to some of the important things to consider regarding this tool and the conducted experiment. Some faults are mentioned as well as some possible ways to deal with them. The chapter ends with a discussion of possible future work that could be used to either develop this application further or perform more research into.

6.1 Summary

A comparison between the two relational databases MySQL and PostgreSQL were conducted to see which database system would be most efficient to use for the images used in the split-tests. This led to the implementation of two separate local databases which was connected to the application with the help of PHP through XAMPP to fully establish a connection to the web application. A PHP form was added to the site to insert the data for split-tests, and the files were converted to BLOB-format when posted to successfully be stored into the databases as binary data. PHP was also used for implementing file conversion which would turn the images into BLOB-format and reverse it whenever an image was to be shown in the application.

Once the tool had been implemented, a pilot study was created which would determine if it was possible to measure the databases and use the results to setup the main experiment. The pilot study concluded that it was possible to measure the databases against each other, and that PostgreSQL would seem to be the better option regarding response times when performing searches of A/B-tests. In the study, PostgreSQL was more than 6 times faster when compared to MySQL. As the pilot study was concluded, some variables for the main measurements was chosen to test further and see if they would affect the results.

The main measurements were then conducted as a part of a technical experiment, which would benchmark the response times of SQL queries and image conversion and retrieval to the application. This experiment included three variables that was deemed interesting to investigate; file size, file formats and database storage. Based of those three variables, 5 separate test cases were formed where each case consisted of 1000 searches of a split-test. A total of 5000 tests for each database was run to see the average benchmarking values for every case. Each test measured the time it took from the point where a split-test was searched for, to when it was visible as a result in the application.

The conclusion of the experiment was that PostgreSQL would be the better option when looking at processing of images stored as binary data, as it had much lower response times when performing searches of A/B-tests compared to MySQL. This disproved the thesis that MySQL would be faster at image processing. While MySQL turned out to perform between 4-10 times slower than PostgreSQL regarding images, it did however perform better in the last test case where HTML files were used. That case, as well as previous research of BLOB-data (Stancu-Mara & Baumann, 2008) led to the assumption that MySQL would be a better option when using other file formats than images.

6.2 Conclusions of experiment

As the experiment was built upon the previous research of Stancu-Mara and Baumann (2008), a technical experiment was conducted to compare the databases much like how they performed theirs. What this study provided was a unique variable which had not previously been reviewed, the processing of stored images. What Stancu-Mara and Baumann concluded was that MySQL was faster at processing BLOB-data than PostgreSQL, which was not the case in this study when it came to using images as the stored binary data. This provides some interesting results, as the specified thesis for this study, which was based upon their results, was proven wrong. In the test case with HTML files, MySQL was faster, but as soon as images were used for measurements, the opposite occurred. As the thesis stated, image processing was the interesting point of view in this study. The results of the experiment showed that PostgreSQL was approximately 4 times faster than MySQL during 3 of the cases, and 10 times faster during the remaining case regarding image processing. Based on the previous research and the HTML test case, MySQL instead appears to be the faster option for other types of stored binary data. HTML files had more responsive results in MySQL, but since they are not relevant to the appliance of scientific research regarding the databases, that conclusion does mostly provide a base for possible future work.
What this study could show is that PostgreSQL is in fact superior to MySQL in terms of image processing, when images are being stored as binary data in databases. If the tool were to be further developed with a single database, PostgreSQL appears to be the better option to choose to provide a more stable database manager with faster processing times.

The standard deviations of the measurements in the experiment do not overlap, which tells us that the results would be quite consistent even if bigger measurements in a new experiment would be conducted. However, more measurements would still be a good thing to do to further confirm the results of the performed experiments and is something that could be further investigated in the future. Especially scalability of image storage in databases is an interesting subject that could use some more research in future work.

6.3 Discussion

This discussion revolves around two separate topics. First, the application itself will be reviewed and some ethical aspects regarding its usage is being brought up and some discussion on how they could be prevented is mentioned. The second part of the discussion is about the experiment itself, and some factors that are very important to consider when looking at the results.

6.3.1 Application

Overall, this application provides a good foundation that could be further developed into an efficient tool for bridging the gap of communication between developers and users. Users of a product would have more impact in the development if the developers would be using this tool to suggest design changes to the users. The users could then leave some feedback of preferred options, which could become very useful in the development of the applications. Previous research, such as the article published by Adinata & Liem (2014) concluded that A/B-tests has been proven to be quite successful when gathering user feedback. A/B-testing, which is a is a part of the web analytics concept, does hold scientific value (Kumar, et al. 2012), which further adds credibility to its usage in the tool.

Some ethical aspects to consider would be how the images are being stored and displayed in the application. It is possible that most companies do not wish to reveal the intended changes of their product designs before releasing the updates. This could lead to a limited usage of the application, as those companies will not be using the tool at all so that they can protect sensitive data.
Another thing to consider is how the stored data would be managed. For how long would these images be stored, and what could a potentially data breach lead to? This tool would require to be very transparent for the developers who would use it, clearly defining how the images would be saved, and most likely also include a set time limit for how long they would be stored in the databases. If a data breach would occur, there would rise lots of difficult problems as potentially thousands of documented design changes through images and logs could be compromised. Lots of security concerns would need to be reviewed and be dealt with to store the data in a good manner in a fully scaled application.

The users themselves would not issue many ethical conflicts as the application stands right now. They are anonymous and the only saved data from them would be the answer of split-tests, a single character A or B, stored into the database. Another important topic arises here instead: *How the results of the split-tests are being saved.*
Optionally, a person would only be able to save a single answer for every split-test. As the application is implemented at this stage, there is neither validation to see if a person or a script is entering the statistics, nor is there any limit on how many answers can be sent into the database. This leaves the application very vulnerable to flood attacks, where a single script can be written to enter unlimited data and eventually crash the system. This also removes the validity of the entered answers, as someone could easily abuse the system and entering false statistics for the split-tests. The issue with flood attacks would also apply to the form where new A/B-tests are being added. That form also lacks validation to stop making new inserts if a single person were to abuse the system, which could crash the server completely.

One of the most concrete solutions to those problems would be to add user accounts, which would be required to view the content and have some restrictions on how they can act on the site. For instance, a user could be restricted to only answering a single time for every split-test, and only being able to upload a single new test in a limited time frame, e.g. once per week. However, by adding user accounts, the information of the users could instead be compromised if a data breach would occur, which leads back to the ethical difficulties of security concerns of sensitive information. If the application would be further implemented for production, security concerns would require high priority.

Storing images in databases also proved to be a good method to choose, despite the topics that the users on the Stack Overflow (2009) mentioned against it. The scalability issues they referred to were not a problem at all in this study, and would rather be something that could be further investigated in future research. However, databases provide a safe storage for the images. It adds a lot of security when compared to other storage methods, such as web hosting, as well as having good control of the access of the users, determining which data they are authorized to view. Databases are also very useful when looking at backup options, which would be essential for a successful launch of such an application. One final useful thing that could be used in conjunction with databases would be possibility to encrypt the data, which would also add more security against potential breaches.

Despite all the ethical difficulties regarding the application, the study showed that it is highly possible to process images within databases without having the users dissatisfied by slow response times. According to the research of Hoxmeier and DiCesare (2000), users normally stay at a site when the response times are below 12 seconds at an average. As the response times never even went over 2 seconds during searches, it showed that users would not be too negatively affected by the loading times, especially as a user would most likely not perform repeated searches once their desired split-test has been found.

One final note about the application is that the concept of image comparisons can be used in a much wider perspective than just for A/B-testing. While A/B-testing is proven to be a great method to gather user statistics, the implementation of the image processing could be applied to other types of areas as well. As an example, it could be used within gaming, where a quiz application would be a good example of when images could be relevant. The image comparison could then instead be used to measure a correct answer, instead of user preferences. It could also be used for educational purposes, for example different tests online. Those two options could also be combined into an application with gamification that has the purpose to aid learning by creating a fun, interactive learning platform.

6.3.2 Experiment

Some factors can have a great impact in the benchmarking results of experiments and needs to be considered. It is a known fact that technical measurements are not completely reliable as a source of research results (Berndtsson et al. 2008). There are factors that can affect the measurements, such as computer hardware or internet connectivity. Usually, measurements also come with a small deviation which will affect the results a small bit. The databases were initiated locally to let connectivity issues have as little impact as possible during the experiment. If the application would be taken into real production, this would not be the case, and new results of the image processing would most likely occur. This would most likely not change the outcome of the databases, but the response times would most likely see a negative impact in such case. As the study of Hoxmeier and DiCisare (2000) stated, response times plays a huge role in how successful an application would be, and this could potentially become an issue of user experience if the impact would be great enough.

Another thing to consider is how the images are saved as their binary form. Specifically, regarding MySQL, there are different sizes used for BLOB-data. *Mediumblob* was the chosen format, as it was most appropriate since it could store data up to 16MB. An ordinary *blob* type would not be enough to contain the images as they are limited to 64KB and *longblob* would negatively affect the response times of the databases, which made the choice to use *mediumblob* the best option. What could be implemented here would be a validation of file sizing, which always stores the images as the smallest type of BLOB possible by checking which types could store the image. That would provide faster response times for the images as the smallest BLOB would always be chosen and take less time to load. However, this would require additional implementations, most likely several tables or columns in the database and would take a significant time to implement to the database.

The difference between the variables *bytea* in PostgreSQL and the *mediumblob* of MySQL could also have an impact when comparing the processing of binary data between the databases. There is a possibility that this could play a major role in why PostgreSQL is performing much more responsive compared to MySQL at retrieving images from the database. However, this is just a mere speculation and might not be the case at all, as the

HTML files were in fact faster in MySQL compared to PostgreSQL, despite them also being stored as binary data as *mediumblob*.

The research performed in this study can be repeated for future experiments. All the required code for the application and databases are listed in the appendix. What's missing is the setup progression on how to initiate local databases, however the PostgreSQL guide that was followed is being referred to in this report. Such information is however relatively easy to find on the internet. Regarding the used data for the A/B-tests, the same images of this research are not essential to perform future studies. Any kind of images could be used when comparing the response times between the databases, as long as those images are being used for both databases in the experiments.

6.4 Future work

As mentioned in the discussion, A/B-testing is an excellent method of gathering user input in the development process. This application has a lot of potential if it were to be further developed. Some new implementations would be possible, one potential implementation would be to include a text input linked tied to the answers of the split-tests. This would allow users to have an even greater impact in the development process by having the opportunity to add some feedback in free text instead of just choosing a preferred option.

The test case with HTML files in the experiment also opens a lot of possibilities for future work. The results showed that different file types can also be stored as BLOB-data in the databases quite successfully. The tool could eventually become a great platform to saving different changes and logs during web development if some further functionality would be added for handling this specific usage of the files. If this application would be further developed, the security issues would be one of the biggest concerns and would require quite a bit of effort to tackle successfully.

What could be further researched into is the usage of different blob types in MySQL. Small files could be compared as *tinyblob* or *blob* instead of *mediumblob* to see how it would affect the benchmarking results. HTML files could be a great choice to try different blob types with, as those files are very small and would fit most kinds of blob. The exception is *tinyblob*, which has a maximum length of 255 bytes, which would not be enough to store a file with the size of several kilobytes. *Longblob* could be viable option to research more into, to allow comparison of very large files spanning up to 4GB. This would allow the application to compare big files such as 4K photos, which could provide some interesting results.

Database engines could also be something worth investigating in the future. *InnoDB* was the default engine of MySQL, which was used for every test case. Additional experiments with other database engines could be performed to see if it would be possible to optimize the search results with response times. Scalability of the databases is also an interesting topic to research in future work, as this study only contained a case where the amount of data was doubled. What could be interesting to investigate is how the databases would perform is several hundreds or thousands of images and see how that would impact the response times. There is a slight possibility that MySQL could perform better when being filled with huge amount of data, which would make it a more appropriate database if that would be the case.

As a final note, this research provides some foundation which could aid more support for image storage in databases in the future. This is something that could see lots of work, as the technology can always be optimized to perform better. Improvements of BLOB storage, or implementation of new formats designed entirely for images are things that could possibly be created, and the aim is to have this research as a potential reference that could help, due to the lack of research in the topic as it stands today.

References

Adinata, M., & Liem, I. (2014). A/B test tools of native mobile application. *Proceedings of 2014 International Conference on Data and Software Engineering*. ICODSE 2014, 1-6.

Berndtsson, M., Hansson, J., Olsson,B., Lundell, B. (2008). Thesis projects: A guide for students in computer science and information systems. Springer, London. ISBN: 978-1-84800-008-7.

Brodovsky, S., & Rosset, S. (2011). A/B testing at SweetIM: *The importance of proper statistical analysis*. Proceedings - IEEE International Conference on Data Mining. ICDM, 733–740.

Brown, A., Lush, B., & Jansen, B. J. (2016). Pixel efficiency analysis: A quantitative web analytics approach. *ASIST '16. Proceeding of the 79th ASIS&T Annual Meeting: Creating Knowledge, Enhancing Lives through Information & Technology*. Silver Springs, MD, USA. Article 40.

Cegan, L., & Filip, P. (2017). Webalyt: Open web analytics platform. *2017 27th International Conference Radioelektronika, RADIOELEKTRONIKA 2017*. IEEE.

Di Giacomo, M. (2005). MySQL: Lessons learned on a digital library. *IEEE Software*, *22*(3). IEEE, 10–13.

Gershon, N., Nielsen, J., Czerwinski, M., Ragouzis, N., Siegel, D., Neale, W. (1998). Good web design: Essential ingredient! *Proceeding CHI 98 Conference Summary on Human Factors in Computing Systems*. ACM, New York, 90-91.

Györödi, C., Györödi, R., Pecherle, G., Olah, A. (2015). A comparative study: MongoDB vs. MySQL. *Engineering of Modern Electric Systems (EMES) 2015 13th International Conference on*. IEE, Oradea, Romania, 1-6.

Hills, M., Klint, P., & Vinju, J. (2013). An Empirical Study of PHP Feature Usage. *ISSTA 2013 Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 325–335.

Hoxmeier, J., & DiCesare, C. (2000). System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications. *Proceedings of the Association of Information*. AiSeL, 1–26.

Kumar, L., Singh, H., Kaur, R., 2012. Web analytics and metrics: a survey. *ICACCI '12 Proceedings of the International Conference on Advances in Computing, Communications and Informatics*. ACM, Chennai, India, 966-971.

Stancu-mara, S., Baumann, P., & Marinov, V. (2008). A Comparative Benchmark of Large Objects in Relational Databases. *IDEAS '08 Proceedings of the 2008 International Symposium on Database Engineering & Applications*. ACM, 277–284.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. *Experimentation in Software Engineering* (Vol. 9783642290442). Springer, Berlin, Heidelberg.

Greasemonkey's homepage [Viewed March 2018]
https://www.greasespot.net/

Postgres installation documentation [Viewed March 2018]
https://www.postgresql.org/docs/current/static/tutorial-install.html

Jain, M. (Talkerscode, 2018) *Upload Image to Database and Server using HTML, PHP and MySQL* [Viewed April 2018]
http://talkerscode.com/webtricks/upload%20image%20to%20database%20and%20server%20using%20HTML,PHP%20and%20MySQL.php

[Usernames] *Brandon Wood* & *giorgian*. (Stack Overflow, 2009) *Saving images: files or blobs?* [Viewed March 2018]
https://stackoverflow.com/questions/1347461/saving-images-files-or-blobs

[Username] *SomeKittens* (Stack Overflow, 2012). *Html FileReader: convert blob to image file?* [Viewed March 2018]
https://stackoverflow.com/questions/13023231/html-filereader-convert-blob-to-image-file

Appendix A - Index.php

```
<!DOCTYPE HTML>

<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1">
        <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css
" rel="stylesheet" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

        <!--External scripts-->
        <script src="./JS/jquery-3.3.1.js"></script>
        <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min
.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
        <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
        <script src="./JS/navigation.js"></script>
        <script src="./JS/formHandler.js"></script>

        <link rel="stylesheet" type="text/css" href="style.css">

        <title>Design application</title>

    </head>
    <body>
        <!--Navigation menu-->
        <nav class="navbar navbar-toggleable-md navbar-dark">
            <button class="navbar-toggler btn btn-basic" type="button"
data-toggle="collapse" data-target="#navContent" aria-controls="navContent"
aria-expanded="false" aria-label="Toggle navigation" id="hamburgerBtn">
                <span class="navbar-toggler-icon"></span>
            </button>
            <!--Collapsed navbar-->
            <div class="collapse navbar-collapse"  id="navContent">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link active" id="homeLink" href="#"
onclick="setContent('Home');">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" id="answerLink" href="#"
onclick="setContent('AnswerTest');">Answer test</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" id="addLink" href="#"
onclick="setContent('AddTest');">Add test</a>
                    </li>
                    <li class="nav-item">
```

```
                        <a class="nav-link" id="resultLink" href="#"
onclick="setContent('ViewResult');">View results</a>
                    </li>
                    <!--Database toggle-->
                    <li class="nav-item">
                        <a class="nav-link" id="toggleDB" href="#">Switch
DB</a>
                    </li>
                </ul>
            </div>
        </nav>


        <!--Page content-->
        <div class="container-fluid">
            <div class="col-1"></div>
            <div class="col-10" id="content">
                <div class="row" id="titleBar"><span id="titleSpan"><h2
id="title">Home</h2></span></div>

                <!--Main page content, dynamic visibility-->
                <div class="row shadow2" id="content1">

                    <!--Index page (default view)-->
                    <div class="col-12 addedMargin indexContent" >
                        <p id="indexText">
                            This is an application used for measuring web
designs with A/B-testing.
                            The links in the left menu is used to navigate
through the site.</p><br/>

                        <table id="indexTable">
                            <tr>
                                <th>Answer Test</th><td>View a split-test
(A/B-test).</td>
                            </tr>
                            <tr>
                                <th>Add Test</th><td>Make a new entry to a
database.</td>
                            </tr>
                            <tr>
                                <th>View Results</th><td>View the
statistics from the split-tests.</td>
                            </tr>
                            <tr>
                                <th>Switch DB</th><td>Toggle the active
database. (Displayed at the bottom)</td>
                            </tr>
                        </table>
                    </div>

                    <!--Design choice/Split-test (answer test)-->
                    <div class="col-12 splittestContent addedMargin">
                        <!-- Search form -->
                        <div class="row">
                            <div class="col-4"></div>
                            <div class="col-4">
                                <div class="input-group">
                                    <input placeholder="Enter test name"
id="searchData" name="searchField" autocomplete="off"/>
                                    <span class="input-group-btn">
```

```
                                        <input class="btn btn-success"
id="searchBtn" type="submit" name="searchTests" value="Search"/>
                                    </span>
                                </div>
                            </div>
                            <div class="col-4"></div>
                        </div>
                        <!--Test output-->
                        <div class="row">
                            <div class="col-5 imageContainer shadow2"><img
alt="Design A" id="img1"/></div>
                            <div class="col-2"><p id="testing"></p></div>
                            <div class="col-5 imageContainer shadow2"><img
src="viewData.php" alt="Design B" id="img2"/></div>
                        </div>
                        <div class="row captionRow">
                            <div class="col-5 whiteBG shadow2 caption"
id="caption1Container">
                                <p id="cap1"></p>
                                <button id="btnA" class="btn btn-success
designBtn">Design A</button>
                            </div>
                            <div class="col-2"></div>
                            <div class="col-5 whiteBG shadow2 caption"
id="caption2Container">
                                <p id="cap2"></p>
                                <button id="btnB" class="btn btn-success
designBtn">Design B</button>
                            </div>
                        </div>
                    </div>


                    <!--Imageform (add new tests)-->
                    <div class="col-12 formContent addedMargin">
                        <div class="col-10 shadow5" id="formDiv">
                            <form enctype="multipart/form-data"
method="POST" id="addTestForm" class="sendForm">
                                <table>
                                    <thead>
                                        <tr><th colspan="3">Add a new split
test</th></tr>
                                    </thead>
                                    <tbody>
                                        <tr>
                                            <th></th>
                                            <th><input type="text"
name="testName" id="testName" placeholder="Name of Test"
maxlength="25"></th>
                                            <th></th>
                                        </tr>
                                        <tr>
                                            <td><img id="output_image1"
class="previewImage"/></td>
                                            <td></td>
                                            <td><img id="output_image2"
class="previewImage"/></td>
                                        </tr>
                                        <tr>
```

```
                                                <td><input class="btn formBtn"
type="file" accept="image/*" name="image1" onchange="preview_image(event,
1);"></td>
                                                <td></td>
                                                <td><input class="btn formBtn"
type="file" accept="image/*" name="image2" onchange="preview_image(event,
2);"></td>
                                            </tr>
                                            <tr>
                                                <th>Caption</th>
                                                <th></th>
                                                <th>Caption</th>
                                            </tr>
                                            <tr>
                                                <td><textarea id="caption1"
name="caption1" rows="5" cols="30"></textarea></td>
                                                <td></td>
                                                <td><textarea name="caption2"
id="caption2" rows="5" cols="30"></textarea></td>
                                            </tr>
                                            <tr>
                                                <td colspan="3"><input
class="btn bottom shadow3" id="sendBtn" type="submit" name="submit-test"
value="Upload"/></td>
                                            </tr>
                                        </tbody>

                                    </table>
                                </form>
                            </div>
                        </div>

                        <!--Add more content here-->

                    </div>

                    <!--Supplementing content-->
                    <div class="row shadow1" id="content2">
                    </div>
                </div>
                <div class="col-1"></div>
            </div>
            <footer id="pageFooter">Database: MySQL</footer>
        </body>
</html>
```

Appendix B – Style.css

```
/*Styling for the application*/
*{
    margin: 0;
    padding: 0;
    font-family: arial;
}
body{
    /*#388E3C*/
    height: 100vh;
    background: #374046;
}

/* --- Navigation styling --- */
#navContent{
    position: absolute;
    z-index: 100;
    -webkit-transition: all 0.2s ease-out;
    -o-transition: all 0.2s ease-out;
    transition: all 0.2s ease-out;
    top: 50px;
}
li a{ color: white;}

/* --- Page title --- */
#titleBar{
    background: #388E3C;
    display: block;
    border-top-left-radius: 4px;
    border-top-right-radius: 4px;
    z-index: 3;
    height: 8%;
}
#titleSpan{
    width: 100%;
    height: 100%;
    display: table;
}
h2{
    color: white;
    display: table-cell;
    vertical-align: middle;
    text-align: center;
}

/* --- Page content --- */

/* Container for page content */
#content{
    margin: auto;
    height: 85vh;
    z-index: 1;
}
/* First content container */
#content1{
    margin-top: 0;
    width: 100%;
    height: 82%;
    background-color: #4CAF50;
```

```
    z-index: 3;
    position: absolute;
}
/* Second content container */
#content2{
    bottom: 0;
    width: 100%;
    background: white;
    height: 10%;
    z-index: 2;
    position: absolute;
    border-bottom-left-radius: 4px;
    border-bottom-right-radius: 4px;
}

/* --- Index -- */
#indexText{
    text-align: center;
    color: #eee;
}
#indexTable{
    color: white;
    margin: auto;
}
#indexTable td{ padding: 5px;}

/* --- View tests --- */
.splittestContent{ display: none;}
.input-group{ justify-content: center;}
#searchBtn{ background: #FF4081;}

/* Containers for images */
.imageContainer{
    background: #388E3C;
    height: 70%;
    vertical-align: middle;
    text-align: center;
    z-index: 3;
    margin: auto;
    color: white;
}
#img1, #img2{
    max-width: 300px;
    max-height: 300px;
    padding: 10px;
}
.caption{
    z-index: 3;
    text-align: center;
    width: 100%;
    height: 100%;
}
.designBtn{
    background: #FF4081;
    margin-bottom: 10px;
}

/*Add images*/
.formContent{ display: none;}
#formDiv{
    background: #388E3C;
```

```
    height: 100%;
    position: inherit;
    vertical-align: middle;
    text-align: center;
    z-index: 4;
    margin: auto;
    border-radius: 4px;
}
#formDiv table{
    width: 100%;
    color: white;
    text-align: center;
    display: table;
}
#formDiv td{ width: 33%;}

.previewImage
{
    background: white;
    width: 150px;
    height: 150px;
    background: white;
    border: 1px solid black;
    display: inline-block;
}
#sendBtn{
    background: #FF4081;
    color: white;
    z-index: 3;
    margin-bottom: 20px;
}
#pageFooter{
    color: white;
    text-align: center;
}

/* --- General styling classes --- */
.addedMargin{ margin-top: 50px;}
.whiteBG{ background: white;}

.shadow1{ box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px
rgba(0,0,0,0.24);}
.shadow2{ box-shadow: 0 3px 6px rgba(0,0,0,0.16), 0 3px 6px
rgba(0,0,0,0.23);}
.shadow3{ box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px
rgba(0,0,0,0.23);}
.shadow4{ box-shadow: 0 14px 28px rgba(0,0,0,0.25), 0 10px 10px
rgba(0,0,0,0.22);}
.shadow5{ box-shadow: 0 19px 38px rgba(0,0,0,0.30), 0 15px 12px
rgba(0,0,0,0.22);}
```

Appendix C – MySQLscript.sql

```
/*DROP DATABASE IF EXISTS MySQL_DB;

CREATE DATABASE MySQL_DB;*/
USE MySQL_DB;

CREATE TABLE SplitTest(
testID int AUTO_INCREMENT PRIMARY KEY,
testName varchar(25) UNIQUE NOT NULL
)engine=innodb;

CREATE TABLE ImageEntry(
ID int AUTO_INCREMENT PRIMARY KEY,
image mediumblob,
caption varchar(100),
testID int,
FOREIGN KEY (testID) REFERENCES SplitTest(testID)
)engine=innodb;

CREATE TABLE Statistics(
ID int AUTO_INCREMENT PRIMARY KEY,
answer char,
testID int,
FOREIGN KEY (testID) REFERENCES SplitTest(testID)
)engine=innodb;

-- View data
SELECT * FROM SplitTest;
SELECT * FROM ImageEntry;
SELECT * FROM Statistics;
```

Appendix D – PostgreSQLscript.sql

```
--Wipe data
 DROP TABLE IF EXISTS ImageEntry;
 DROP TABLE IF EXISTS SplitTest;

 --Create tables
CREATE TABLE SplitTest
(
      testID SERIAL PRIMARY KEY,
      testName varchar(25) UNIQUE NOT NULL
);
CREATE TABLE ImageEntry(
      ID SERIAL PRIMARY KEY,
      image bytea,
      caption varchar(100),
      testID int,
      FOREIGN KEY (testID) REFERENCES SplitTest(testID)
);

CREATE TABLE Statistics(
      ID SERIAL PRIMARY KEY,
      answer char,
      testID int,
      FOREIGN KEY (testID) REFERENCES SplitTest(testID)
);

--Display data
SELECT * FROM ImageEntry;
SELECT * FROM SplitTest;
SELECT * FROM Statistics;
```

Appendix E – MySQLcon.php

```
<?php
$dbhost = "localhost";
$dbname = "MySQL_DB";
$dbuser = "root";
$dbpass = "qwe123";
try{
    $PDO = new PDO('mysql:host='.$dbhost.'; dbname='.$dbname, $dbuser,
$dbpass);
}
catch(PDOException $connectionError){
    echo "Failed to connect to database: ".$connectionError->getMessage();
}
?>
```

Appendix F – PostgreSQLcon.php

```
<?php
$dbhost = "localhost";
$dbname = "psql_database";
$dbuser = "postgres";
$dbpass = "qwe123";
try{
    $PDO = new PDO('pgsql:host='.$dbhost.'; dbname='.$dbname, $dbuser,
$dbpass);
}
catch(PDOException $connectionError){
    echo "Failed to connect to database: ".$connectionError->getMessage();
}
?>
```

Appendix G – Navigation.js

```
//Displays content based on link clicked
function setContent(clickedLink) {
    //Hide contents
    $("a").removeClass("active");
    $(".indexContent").hide();
    $(".splittestContent").hide();
    $(".formContent").hide();

    //Display correct content based on active link
    switch(clickedLink){
        case "Home":
            $("#homeLink").addClass("active");
            $(".indexContent").show();
            document.getElementById("title").innerHTML="Home";
            break;
        case "AnswerTest":
            $("#answerLink").addClass("active");
            $(".splittestContent").show();
            document.getElementById("title").innerHTML="Design comparison";
            break;
        case "AddTest":
            $("#addLink").addClass("active");
            $(".formContent").show();
            document.getElementById("title").innerHTML="Add images for new
tests";
            break;
        case "ViewResult":
            $("#resultLink").addClass("active");
            document.getElementById("title").innerHTML="View test results";
    }
}
```

Appendix H – formHandler.js

```
$(document).ready(function(){
    //Hide image captions
    $(".captionRow").hide();

    //Add function that determines DB
    var DBtype = 'mySQL';
    var activeTestID;

    $("#toggleDB").click(function(){
        var DBname;
        //Toggle active connection
        if(DBtype == 'mySQL'){
            DBtype = 'pgSQL';
            DBname = "PostgreSQL";
        }else{
            DBtype = 'mySQL';
            DBname = "MySQL";
        }
        alert("Database changed to " + DBname + "!");
        document.getElementById("pageFooter").innerHTML = "Database: " +
DBname;
    });

    //Send submitted form data with AJAX when adding tests
    $("#addTestForm").submit(function (e){
        e.preventDefault();
        var formData = new FormData(this);
        //Add connection type
        formData.append('DBcon', DBtype);

        $.ajax({
            type: 'POST',
            url: "./PHP/insertTest.php",
            data: formData,
            contentType: false,
            processData: false,
            dataType: 'json',
            success: function(data){
                alert(data['output']);

                //Reset form if submit was successful
                if(data['success'] == true){
                    $("#addTestForm")[0].reset();
                    $("#output_image1").removeAttr('src');
                    $("#output_image2").removeAttr('src');
                }
            },
            error: function(exception){
                console.log(exception.responseText);
            }
        });
    });
```

```
    //Search form for finding tests
    $("#searchBtn").click(function(){

        //Benchmarking variable
        var startTime = (new Date).getTime();

        $.ajax({
            type: 'POST',
            url: "./PHP/searchTest.php",
            cache: false,
            data: {
                value: $("#searchData").val(),
                activeConnection: DBtype
            },
            success: function(data){
                //If search results were found
                if (data[0] !== undefined && data[1] !== undefined) {

                    //Returns images based on search result
                    $("#img1").attr("src", data[0]);
                    $("#cap1").html(data[1]);
                    $("#img2").attr("src", data[2]);
                    $("#cap2").html(data[3]);

                    //Save testID to use when sending answers
                    activeTestID = data[4];

                    //Control visibility of displayed content
                    $(".captionRow").show();

                    //Benchmarking result
                    var timeDiff = (new Date).getTime() - startTime;
                    //sendBenchmark(timeDiff, DBtype);
                }
                else{
                    alert("No search results found!");
                }
            },
            dataType: "json",
            error: function(exception){
                console.log(exception.responseText);
            }
        });
        $("#searchData").val("");
    });

    //Call method for sending A/B-test answers
    $("#btnA").click(function (){
        sendAnswer(activeTestID, 'A', DBtype);
    });
    $("#btnB").click(function (){
        sendAnswer(activeTestID, 'B', DBtype);
    });

});

//Send answers of A/B-testing to database
function sendAnswer(ID, value, DBtype){
    console.log("TestID: " + ID + ", Answer: " + value);
```

```
$.ajax({
        type: 'POST',
        url: "./PHP/insertStatistics.php",
        cache: false,
        data: {
            testID: ID,
            answer: value,
            activeConnection: DBtype
        },
        success: function(data){
            alert(data);
        },
        error: function(exception){
            console.log(exception.responseText);
        }
    });
}

//Preview image when inserting tests
function preview_image(event, id) {
    var reader = new FileReader();
    reader.onload = function()
    {
        if(id==1){
            var output = document.getElementById('output_image1');
        }
        else{
            var output = document.getElementById('output_image2');
        }
        output.src = reader.result;
    }
    reader.readAsDataURL(event.target.files[0]);
}

//Send benchmark timers to DB
function sendBenchmark(responseTime, DBtype){
    $.ajax({
        type: 'POST',
        url: "./PHP/writeResponsetime.php",
        cache: false,
        data: {
            benchmark: responseTime,
            activeConnection: DBtype
        },
        success: function(data){
            console.log("Timer saved: " + data + "ms");
        },
        error: function(exception){
            console.log(exception.responseText);
        }
    });
}
```

Appendix I – insertData.php

```
<?php
class Insert{

    //Create a new test
    function insertTest($testName, $activeDB){
        if($activeDB == 'mySQL'){
            require "MySQLcon.php";
        }else if($activeDB == 'pgSQL'){
            require "PostgreSQLcon.php";
        }

        $sql = "INSERT INTO SplitTest(testName) VALUES ('$testName')";
        $insertQuery = $PDO->prepare($sql);

        //If test name already exist in database
        if(!$insertQuery->execute()){
            return false;
        }
        return true;
    }


    //Read image and return correct data to store into DB
    function convertImage($tmpimage, $activeDB){

        $fp = fopen($tmpimage, 'r');
        $data = fread($fp, filesize($tmpimage));

        //Different file handling for db's
        if($activeDB == 'mySQL'){
            $file = addslashes($data);
        }else if ($activeDB == 'pgSQL'){
            $file  = pg_escape_bytea($data);
        }

        fclose($fp);

        return $file;
    }


    //Query for inserting images into database
    function insertImage($tmpimg, $tmpcaption, $tmpID, $activeDB){
        if($activeDB == 'mySQL'){
            require "MySQLcon.php";
        }else{
            require "PostgreSQLcon.php";
        }

        $query = "INSERT INTO ImageEntry(image, caption, testID) VALUES
('$tmpimg', '$tmpcaption', '$tmpID')";
        $insertQuery = $PDO->prepare($query);

        if(!$insertQuery->execute()){
            echo ("Error while inserting images!");
        }
    }
```

```
    //Insert answers from A/B-tests
    function insertAnswers($testID, $answer, $activeDB){
        if($activeDB == 'mySQL'){
            require "MySQLcon.php";
        }else if($activeDB == 'pgSQL'){
            require "PostgreSQLcon.php";
        }

        $query = "INSERT INTO Statistics(answer, testID) VALUES ('$answer',
'$testID')";
        $insertQuery = $PDO->prepare($query);

        if(!$insertQuery->execute()){
            echo "Error while inserting statistics!";
        }else{
            echo "Answer was successfully inserted!";
        }
    }


    //Adds images and captions linked to a split-test to the database
    function submitForm($testID, $img1, $caption1, $img2, $caption2,
$activeDB){
        self::insertImage($img1, $caption1, $testID, $activeDB);
        self::insertImage($img2, $caption2, $testID, $activeDB);
    }
}
?>
```

Appendix J – insertTest.php

```
<?php
require "insertData.php";
require "viewData.php";

//Form data retrieved from Ajax
$testName = $_POST['testName'];
$img1blob = $_FILES['image1']['tmp_name'];
$img2blob = $_FILES['image2']['tmp_name'];
$caption1 = $_POST['caption1'];
$caption2 = $_POST['caption2'];
//Connection type
$activeDB = $_POST['DBcon'];

//Boolean for checking valid database submit
$insertSuccess = false;

//Handle form data from test inserts
if(isset($testName) && ($testName !== "")){

    //Check if both images are set
    if (isset($img1blob) && ($img1blob !== "") && isset($img2blob) &&
($img2blob !== "")){

        //Check if testname is unique
        if(Insert::insertTest($testName, $activeDB)){
            $testID = Retrieve::getTestID($testName, $activeDB);

            //Convert images to correct format
            $img1 = Insert::convertImage($img1blob, $activeDB);
            $img2 = Insert::convertImage($img2blob, $activeDB);

            //Submit data
            Insert::submitForm($testID, $img1, $caption1, $img2, $caption2,
$activeDB);
            $returnString = "Test was added! \n Name: ".$testName."\n ID:
".$testID;

            $insertSuccess = true;
        }
        else{
            $returnString = "A test with that name already exists!";
        }

    }
    else{
        $returnString = "Both images must be attached!";
    }

}
else{
    $returnString =  "No test name is selected!";
}

//Return data to Ajax
$returnList = array(
    "output" => $returnString,
    "success" => $insertSuccess
);
```

```
echo json_encode($returnList);

?>
```

Appendix K – searchTest.php

```
<?php
require "viewData.php";
$activeDB = $_POST['activeConnection'];

//Get testID by searched name on page
$testID = Retrieve::getTestID($_POST['value'], $activeDB);

//Return data based on testID
Retrieve::getImage($testID, $activeDB);

?>
```

Appendix L – viewData.php

```
<?php
class Retrieve{
    function getImage($testID, $activeDB){
        if($activeDB == 'mySQL'){
            require "MySQLcon.php";
        }else{
            require "PostgreSQLcon.php";
        }

        $outputArray = array();

        //Select correct images based in testID
        $fetchQuery = $PDO->prepare('SELECT image , caption FROM
ImageEntry, SplitTest WHERE ImageEntry.testID = SplitTest.testID AND
ImageEntry.testID = '.$testID);
        $fetchQuery->execute();

        //Fetches results from database
        while($row = $fetchQuery->fetch()){
            $img = $row['image'];
            $caption = $row['caption'];

            //Get content of image blob and encode it to png
            ob_start();

            //Different file management for DB's
            if($activeDB == 'mySQL'){
                $image = imagecreatefromstring($img);
                imagepng($image);
            }else if($activeDB == 'pgSQL'){
                fpassthru($img);
            }

            $data = ob_get_contents();
            ob_end_clean();
            $newData = " " . base64_encode($data);

            //Push image data into array that will be returned to JS
            array_push($outputArray, $newData);
            array_push($outputArray, $caption);
        }
        //Also adds testID to use when sending statistics later on
        array_push($outputArray, $testID);

        //Returns array to AJAX request
        echo json_encode($outputArray);
    }

    //Returns ID of inserted test
    function getTestID($testName, $activeDB){
        if($activeDB == 'mySQL'){
            require "MySQLcon.php";
        }else{
            require "PostgreSQLcon.php";
        }

        $fetchQuery = $PDO->prepare("SELECT testID FROM SplitTest WHERE
testName='$testName'");
```

```
        $fetchQuery->execute();
        $row = $fetchQuery->fetch();

        //Postgres requires small chars
        if($activeDB == 'mySQL'){
            $ID =$row['testID'];
        }else if($activeDB == 'pgSQL'){
            $ID = $row['testid'];
        }

        return $ID;
    }
}

?>
```

Appendix M – insertScript.js (Greasemonkey)

```
/* Search script used generating data */

// ==UserScript==
// @name     Exjobb InsertScript
// @version  1
// @grant    all
// @include http://localhost/exjobb/Application/*
// @require http://code.jquery.com/jquery-2.1.0.min.js
// ==/UserScript==

$(document).ready(function(){

    var ID  = 35;
    var name = 0;

    console.log("InsertScript successfully loaded");

    $("#addLink").click(function(){
      ID++;
      name++;
      addData(ID, name);
    });
});

function addData(counter, name){
  console.log(counter);
  $("#testName").val("H" + name);
  $("#caption1").val("First file. (HTML)  TestID " + counter);
  $("#caption2").val("Second file. (HTML) TestID " + counter);
}
```

Appendix N – searchScript.js (Greasemonkey)

```
/* Search script used for benchmarking with Tampermonkey */

// ==UserScript==
// @name     Exjobb SearchScript
// @version  1
// @grant    all
// @include http://localhost/exjobb/Application/*
// @require http://code.jquery.com/jquery-2.1.0.min.js
// ==/UserScript==
$(document).ready(function(){

    console.log("Script successfully loaded");

    //Initialize script by clicking on the link
    $("#answerLink").click(function(){
        var counter = 0;
        fillForm(counter);
    });
});

function fillForm(counter){
    //Run 10 searches with 3 second intervals
    if(counter < 10){
        setTimeout(function(){
            counter++;
            console.log(counter);
            $("#searchData").val("Test1");
            $("#searchBtn").click();

            fillForm(counter);
        }, 3000);
    }
    else{
        console.log("Measurements are finished");
    }
}
```

Appendix O – writeResponsetime.php

```
<?php
$time = $_POST['benchmark'];
$activeDB = $_POST['activeConnection'];
if($activeDB == 'mySQL'){
    $file = "../Benchmark/mySQLresults.txt";
}else{
    $file = "../Benchmark/pgSQLresults.txt";
}

//Read file
$content = file_get_contents($file);
//Write new times to file
$content .= $time . "\n";
file_put_contents($file, $content);

echo $time;
?>
```

Appendix P – insertStatistics.php

```
<?php
require "insertData.php";

$activeDB = $_POST['activeConnection'];
$testID = $_POST['testID'];
$answer = $_POST['answer'];

Insert::insertAnswers($testID, $answer, $activeDB);

?>
```