Hochschule Bremerhaven

Faculty II
Management and Information Systems
Informatics B. Sc.

Infrastruktur

Project Protocol

Team E

Chatbot ”Robbi”

Members:	Aliah Bahmann, Mat. No. 39185
	Muhammad Zahid, Mat. No. 38821
	Erfan Karimi, Mat. No. 39334
	Silas Rathgeber, Mat. No. 37584
Submitted on:	23 September 2022
Supervisor:	Prof. Dr. Oliver Radfelder

Contents

1 Preamble

As our document has been written by all the four team-members, in front of the heading of each chapter/section/subsection the name of the author of that particular section is mentioned.

2 General introduction

by Aliah Bahmann

Through the approx. fourteen lectures, we got to hear through this semester, we gained an understanding for requirements and essential components on a reliable, stable, web-based infrastructure. We have talked about the specific infrastructure at our college as well as about guidelines in general. This report contains our collected knowledge about different technologies, a description of our project and our experiences working together.

2.1 Considerations

by Silas Rathgeber

Our considerations for finding our project idea revolved around our future occupation as programmer. We wanted to learn how to implement classic web functions that offer added value for companies.

2.2 Idea

Our core idea was to create a chatbot, which can handle basic conversations, similar to what we find on a number of websites of larger corporations. These bots are created to prevent the customer from contacting a company employee directly to save resources. It only helps, of course, if the bot is actually able to answer the customer's questions appropriately. We have added a useful feature that provides the user with the opportunity to let the bot know which answer would have been correct, in case the bot could not find an appropriate answer to the asked question in our database. Then the asked question, along with the answer provided by the user, is added automatically to our databse. We also wanted to provide classic web functions such as a login, sign up or sessions.

3 Technologies

This sections' purpose is to present our research on all the technologies that were mentioned during class lectures. Specifically we divided into those, which were used to create Robbi. All descriptions are still unspecific and kept as general as possible.

3.1 Nohup

by Erfan Karimi

The name nohup stands for ”no hangup”. With Nohup we are able to run a process without it being attached to the terminal. What does this mean? Normally, when we start a process, by running a script, we can't run any other process or type a command till the current process is over. It is possible to make the terminal not wait for the process to be finished by adding the ”&” sign at the end of our command or press *Ctrl z*, while our process is still running and write *bg* (send to background). This allows us to start another process at the same time. Nonetheless, the problem still is, that if we close our terminal, every process of our current session will be terminated, since the hangup (HUP) signal gets sent to our process to inform that its user (we) is logged out, and the process has to be terminated. Using Nohup it will catch the ”hung up” signal and allow our process to continue running. Process output can be changed from the standard *nohup.log* to something else. [Hop21]

3.1.1 Common Usage

In the class assignments, we used *nohup* to start a TCP server with *ncat* and keep it in the background while we close the terminal. After that we were able to communicate with that TCP-server-port from another server. In the code below, we will start a process, redirect its output to a log file, close the terminal and open it again to check if there is any further output in the log-file. Finally terminate the process using the *process-id* with the command *kill*.

```bash
#file:      <<TestNohup.sh>>
#date:      2022-08-28_14:48:37
#Topic:    Testing example for nohup

while (true); do
  echo $(date '+%H:%M:%S')
  sleep 1;
done
```

Listing 1: bash: Nohup script example

In order to test Nohup, we wrote the above script that prints out the time every second. Using Nohup we execute the script in the background and redirect its output into a file.

```
infra-2022-e@hopper:~$ nohup ./TestNohup.sh 2>error.log >outPut.log &
[5] 1054878
infra-2022-e@hopper:~$ date '+%H:%M:%S'
21:20:41
infra-2022-e@hopper:~$ tail -f outPut.log
21:21:13
21:21:14
21:21:15
21:21:16
21:21:17
21:21:18
infra-2022-e@hopper:~$ ps -e | grep 1054878
1054878 pts/13   00:00:00 sh
infra-2022-e@hopper:~$ exit
Abgemeldet
erfkarimi@hopper:~$ sudo -i -u infra-2022-e
Hallo infra-2022-e
infra-2022-e@hopper:~$ ps -e | grep 1054878
1054878 pts/13   00:00:00 sh
infra-2022-e@hopper:~$ kill 1054878
```

Listing 2: bash: Nohup syntax example

As described in the first line, the command starts with the *nohup* followed by a command or script. The signs *2>* and > redirect the errors (*stderr*) and *stdout* to a given file. For a process to continue running in the background, a line has to end with an "&" sign. In our next step, we check the current time to track if there is something being sent to the output file by the script that is running. As we can see, after our *tail* command, the time change is being appended to the output file. We check if the process is in the list of processes and exit the terminal, then we sign in and check for it once again. We find out that it is still there, which shows the process is still running. At the end, if we like to end the process, we just kill it with the *kill* command using the *process ID*.

3.2 Redis

by Erfan Karimi

Redis is a database server, where the so called in-memory data structure store can be used. It is possible to save data in different data structures and run different atomic operation on them; such as, appending to a string, incrementing the value in a hash, pushing an element to a list, computing set intersection, union and difference, or getting the member with the highest ranking in a sorted set. [red22a]

3.2.1 Common Usage

During the lectures, in our class assignments, we used the command *redis-cli* to set, get and increment the value of a self defined variable. Some examples of how to use *Redis-cli* are shown in the following code. [red22b]

```
infra-2022-e@hopper:~$ redis-cli set tmp 4
OK
infra-2022-e@hopper:~$ redis-cli incr tmp
(integer) 5
infra-2022-e@hopper:~$ redis-cli get tmp
"5"
infra-2022-e@hopper:~$ redis-cli decr tmp
(integer) 4
infra-2022-e@hopper:~$ redis-cli decr tmp
(integer) 3
```

Listing 3: bash: Redis-cli used with set, get, incr and decr

In line one, we define and assign a value to the variable *tmp* of type integer with command *redis-cli* followed by *set*, which sets the value of *tmp* to 5. The next command *get* is used to get the value of *tmp* in *stdin* (standard in). The Commands *incr* (increase) and *decr* (decrease) are used to increase and decrease the value of the variable by one.

```
infra-2022-e@hopper:~$ redis-cli -r 5 -i 1 incr tmp
(integer) 9
(integer) 10
(integer) 11
(integer) 12
(integer) 13
infra-2022-e@hopper:~$
```

Listing 4: bash: Redis-cli with -r and -i option

As we can observe, it is also possible to execute a single command for a specified number of times with a *user-selected pause*. The *-r* option defines how many times to run a command, and the option *-i* defines a delay between every execution.

3.3 Ncat and TCP/IP

by Erfan Karimi

Ncat is, as our professor fondly calls it, the network *Swiss army knife*. Ncat is a terminal communication utility, which enables two or more network-participants to communicate over TCP or UDP protocol. With ncat it is possible to open a communication server on; e.g., device-user X with specific ncat-ports and communicate with this Host user (listener) over its ports.

3.3.1 Ncat Running Instructions

```bash
ncat -l [options] [host] [port]
ncat -l 2200
```

Listing 5: bash: Ncat with -l option

Using ncat with -l option, sets the device as listener. So it listens and responds to all in-coming data. In the code after this section, we will use the -e (execute) option to bind an executable script for an automatic behavior. For this to work, we also need to scan for all open ports, since not every port will be open.

```bash
ncat ip_address 2200
```

Listing 6: bash: Testing ncat on a client as an example

The command given above is used to communicate with the host-server or the *listener* from Client. By default, it is set to TCP protocol. In case we want to use UDP, we could add the -u option to the command. [NMA22; Rob22]

3.3.2 Ncat Working Example

In this part, we start an *ncat-listener* for message communication and bind it to an executable file for response communication. In the next step, we run the script in the background using *nohup*. Finally, we go back to the client server *hopper* and connect to *docker* for communication.

```bash
#!/bin/bash
#server:    docker
#file:      tcprunner.sh
#topic:     Start a tcp listener server.

ncat -t -k -l 0.0.0.0 2000 -e responsetcp.sh
```

Listing 7: bash: Script for setting up a ncat listener on port 2000

In this script, ncat uses the port 2000 to start a *listener* (option -l) *ncat-server*. The option -k is used to keep the communication open after we have sent our message. Those four zeros separated by a point in line 6 of the code means *ncat* should listen to every possible connection, just the like Wi-Fi or LAN connection-ports. The option -e defines a handler-script, which gives a response to every message that is received.

```

#!/bin/bash
#server:    docker
#file:      responsetcp.sh
#topic:     Handles communication with ncat client

echo "successfully connected"
while read command;do

  case "$command" in

  "next")
    redis-cli incr c
    ;;

  "test")
    echo "Test successful"
    ;;

  "ls")
    /bin/ls
    ;;

  "pwd")
    /bin/pwd
    ;;

  *)
    echo "No response for message \"$command\""
    ;;

esac
done
```

Listing 8: bash: Ncat response bash script

The above script is a response-bash-script for *ncat*. Every input is read over *stdin* with a *while loop* as the variable *$command*. Afterwards there are different response-cases based on the value of *command*.

```
#!/bin/bash
#server:      docker
#file:        starttcp.sh
#topic:       Run tcprunner.sh with nohup in background.
```

```bash

pkill -9 ncat #kill all associated PID
cd ~/infra/tcp/
nohup ./tcprunner.sh &> tcp.log &
id=$!
echo $id > tcp.pid

echo $id
```

Listing 9: bash: This file starts ncat in the background thread

The code given above describes, how *ncat* starts. We run *ncat* in the background thread, using *nohup*. Keeping in mind that every such Process needs to be terminated at some point and for that we need its process ID. We catch and write that in a file called *tcp.pid*. This way we can always recall and kill the process using its process ID.

```bash
erfkarimi@hopper:~$ ncat turing 101010
successfully connected
input> "pwd"
/home/docker-erfkarimi/infra/tcp
input> "test"
erfolgreich
input> "sdfsdfsdf"
No response for message "sdfsdfsdf"
input> ""
No response for message ""
input> "ls"
killncat.sh
log.log
ncat.txt
responsetcp.sh
starttcp.sh
tcp.log
tcp.pid
tcprunner.sh
```

Listing 10: bash: Starting ncat on a client

As described in the preceding code, *ncat* followed by *turing* and port (not real), starts a communication with our host. Turing is the name of our docker domain, by which its IP-Address is identified; alternatively we can also write a normal IP instead. As mentioned before, everything we type gets sent to *responsetcp.sh* on our docker server, and we receive a response based on what we wrote.

3.4 Git Version Control

by Erfan Karimi

Git is a tool for version control on big or small projects, with diversified number of team members. With git, we can create a so-called *bare repo* and request a copy of this repository with the *clone* option. A *cloned repository* represents our version of the project, where we add changes to a project-file and do our work. In order to add these changes to our main repository, we can create a commit of the changes by adding them to the *commit.* Now in order to make these changes accessible to all project members we can send our *commit* to the main project repository by using the *push* command. This way, other participants can load the *commit* using the *pull* command, which is a combination of *git fetch* followed by *rebase* or *merge.*

3.4.1 Bare Repository

In the sense of how *git* works, we would like to show some code-lines on how to use commands like *clone, add, commit, push* and *pull* on data.

```
infra-2022-e@hopper:~/bare_repos$ mkdir repoX
infra-2022-e@hopper:~/bare_repos/repoX$ git init --bare repoX
Leeres Git-Repository in /home/erfkarimi/bare_repos/repoX/ initialisiert
```

Listing 11: bash: Git create bare repo

As mentioned before, we normally clone a repository from its origin path. However, if we want to create a new origin; such as, a path, a server or a service like GitHub or GitLab, then we need to specifically create or mark a path as repository origin (upstream origin). This way it can be *cloned* by a user. The above given code shows, how a *git* repository can be created with *git* commands. First, we have to create a directory to mark as *bare repository* and then we use the command ”git init –bare «repo directory»” to mark it as the center for the code-sharing. In a bare repository, we cannot add any *commits* directly, rather we first have to *clone* the repository and then add the changes in the cloned repository to a *commit* and finally *push* the changes.

3.4.2 Elementary Operations

In this part, we show some elementary *git* commands, like creating and pushing a commit.

```
 1 infra-2022-e@hopper:~/bare_repos$ cd ~/repos
 2 infra-2022-e@hopper:~/repos$ git clone ~/bare_repos/repoX
 3 clone nach 'repoX'...
 4 warning: You seem to have cloned an empty repository.
 5 Finished.
 6 
 7 infra-2022-e@hopper:~/repos$ cd repoX/
 8 infra-2022-e@hopper:~/repos/repoX$ touch filex.txt
 9 infra-2022-e@hopper:~/repos/repoX$ git add filex.txt
10 infra-2022-e@hopper:~/repos/repoX$ git commit -m "initial commit"
11 [main (Root-Commit) d2e3e3f] initial commit
12  1 file changed, 0 insertions(+), 0 deletions(-)
13  create mode 100644 filex.txt
14 
15 infra-2022-e@hopper:~/repos/repoX$ git push
16 Objekte aufzählen: 3, fertig.
17 Zähle Objekte: 100% (3/3), fertig.
18 Schreibe Objekte: 100% (3/3), 224 Bytes | 74.00 KiB/s, fertig.
19 Gesamt 3 (Delta 0), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
20 To /home/erfkarimi/bare_repos/repoX
21  * [new branch]      main -> main
```

Listing 12: bash: Git clone bare repo and add changes

In order to clone a repository, we change our current directory to a directory other than the bare repository and use the command *git clone* followed by the path to the bare repository. This is how we clone a repository to our current path. In this repository, as it is shown on line 8, we create a file; e.g., fileX.txt and add this to a *commit* with *add* command. Now, before we are able to push this change we need to specify a *commit*, meaning we give it a subject and possibly a description. With the command *git commit* followed by the option ”-m”, we only add a subject and skip the commit description. It is important to note that this *commit* exists only locally at the moment and for it to be added to the main repository (bare repository), we use the *git push* command, as shown in line 15.

3.4.3 Branching

Branching, as it is known, is one of git’s ”Killer-features”. With every *commit*, we create a new branch. There is a pointer to the newest commit in the origin repository, called ”main (master)”, which always changes its pointing position automatically. By using the *checkout* command, we can change our branch to a previous one to see our previous working line.

Creating a Branch

That is not all that we could do with branching. As an example, let's say we have an idea on how our project could be realised differently, but it can't be implemented to the main branch, presumably because it is not directly adoptable to the project and may cause problems for other team members. In this circumstance, we can add a new branch as a copy of the main branch or some other branch to the project and carry out our alternative idea in that branch. The goal of this section is to show how this could be done. [McK21; git22]

```
infra-2022-e@hopper:~/repos/repoX$ git branch first_branch
infra-2022-e@hopper:~/repos/repoX$ git branch -a
   first_branch
* master
  remotes/origin/master
infra-2022-e@hopper:~/repos/repoX$ git checkout first_branch
Zu Branch 'first_branch' gewechselt
infra-2022-e@hopper:~/repos/repoX$ ls
fileX.txt
infra-2022-e@hopper:~/repos/repoX$ git branch -a
* first_branch
  master
  remotes/origin/master
```

Listing 13: bash: Creating a branch named firstbranch

In the above code, we create a branch called first_branch on line one. With the command *git branch -a*, we can display all of the existing branches. The "*" sign next to a branch name shows on which branch we currently are. Then we change the pointer to the branch with the command *git checkout first_branch* and can see that it is a copy of our current master (main) branch. At the end, we check for our current branch name, which is marked as, *first_branch*.

Adding changes to the Branch and Comparing

In the following code example, we add some changes to the *first_branch* and go back to the main branch to make sure, that those changes in the *first_branch* don't exist in the main branch.

```
infra-2022-e@hopper:~/repos/repoX$ touch file_first_branch.txt
infra-2022-e@hopper:~/repos/repoX$ git add file_first_branch.txt
```

```
3  infra-2022-e@hopper:~/repos/repoX$ git commit -m "Commit in the first branch"
4  [first_branch 9095121] Commit in the first branch
5   1 file changed, 0 insertions(+), 0 deletions(-)
6   create mode 100644 file_first_branch.txt
7
8  infra-2022-e@hopper:~/repos/repoX$ git push --set-upstream origin first_branch
9  Objekte aufzählen: 3, fertig.
10 Zähle Objekte: 100% (3/3), fertig.
11 Delta-Kompression verwendet bis zu 32 Threads.
12 Komprimiere Objekte: 100% (2/2), fertig.
13 Schreibe Objekte: 100% (2/2), 259 Bytes | 259.00 KiB/s, fertig.
14 Gesamt 2 (Delta 0), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
15 To /home/infra-2022-e/bare-repo/repoX
16  * [new branch]      first_branch -> first_branch
17 Branch 'first_branch' folgt nun 'origin/first_branch'.
18
19 infra-2022-e@hopper:~/repos/repoX~$ ls
20 file_first_branch.txt  fileX.txt
21
22 infra-2022-e@hopper:~/repos/repoX~$ git checkout master
23 Zu Branch 'master' gewechselt
24 Ihr Branch ist auf demselben Stand wie 'origin/master'.
25 infra-2022-e@hopper:~/repos/repoX~$ ls
26 fileX.txt
```

Listing 14: bash: Adding changes to "firstbranch"

In lines one to three, we *add* and *commit* a new change as usual, for *push*, however, we need to do a little more. Every branch such as *main* needs to be connected to a branch called *upstream* on the origin path; it's simply the path, where it is being saved. It is to be noted that since we just created a new branch, it has no upstream branch. Therefore, we also need to connect it to an Upstream branch during the first time we *push* and that's what *–set-upstream origin first_branch* followed by *git push* is for. Furthermore, from now on it will be connected to the *origin/first_branch*. Next time when we *push*, it won't be necessary to give an upstream path anymore. In the next step we change our branch to main (master), using the command *git checkout master* and, as it is shown above, its content varies from what we saw in the *first_branch*.

Creating a Branch with the Option -b

A simpler way to create and directly checkout to a new branch is by using the *-b* option with *checkout* command.

```
infra-2022-e@hopper:~/repos/repoX$ git branch -a
  first_branch
```

```
* master
  remotes/origin/first_branch
  remotes/origin/master
infra-2022-e@hopper:~/repos/repoX$ git checkout -b second_branch
Zu neuem Branch 'second_branch' gewechselt

infra-2022-e@hopper:~/repos/repoX$ ls
fileX.txt
```

Listing 15: bash: Create a branch with -b option

This way, we can easily create and change our branch to a branch called the *second_branch*.

Creating a Branch from another Branch

In this subsection, we intend to create another branch called the *third_branch* from our previous *first_branch*. We simply put their names after each other using the *git branch* command.

```
infra-2022-e@hopper:~/repos/repoX$ git branch third_branch first_branch
infra-2022-e@hopper:~/repos/repoX$ git checkout third_branch
Zu Branch 'third_branch' gewechselt
infra-2022-e@hopper:~/repos/repoX$ ls
file2.txt  file_first_branch.txt  fileX.txt
```

Listing 16: bash: Creating a branch from another one

As demonstrated, we now have another copy of the *first_branch*, that we name as *third_branch*.

Creating a Branch from a Commit

In this segment we will choose a *commit hash code* of the previous commits in the master branch and will create a branch from that *commit*.

```
infra-2022-e@hopper:~/repos/repoX$ ls
file_beta.txt  fileX.txt

infra-2022-e@hopper:~/repos/repoX$ git log --oneline
69e50f3 (HEAD -> master, fourth_branch) second commit
7d5d162 (origin/master, second_branch) initial commit

infra-2022-e@hopper:~/repos/repoX$ git branch fifth_branch 7d5d162
infra-2022-e@hopper:~/repos/repoX$ git checkout fifth_branch
Zu Branch 'fifth_branch' gewechselt
infra-2022-e@hopper:~/repos/repoX$ ls
```

```
fileX.txt
```

Listing 17: bash: Create a branch from a commit

As we can see, with the command *git log –oneline*, we can display all of the *commits* and their *hash code* with which they're linked. To create a copy of a specific *commit* as a new branch, we choose the *commit* with the hash code *7d5d162*. After it is created, we check in to this branch and can see that it contains only the content of that specific *commit*. This can be useful if we have some idea on how a previous file version can be changed.

3.5 DBMS, MySQL

by Muhammad Zahid

As Database Management System (DBMS), we used MySQL in our project. MySQL is a well-known, inexpensive DBMS. This database system is relational and can handle very large databases. It performs fast and is reliable. Furthermore, MySQL is compatible with standard SQL, is written in C and C++, and works on many different platforms.

There are several reasons as to why one would use MySQL or prefer it over some of the other DBMSs available. First and foremost, it is cheap, while other DBMSs, depending on user's requirements, can be quite costly. MySQL Can be installed locally and it provides multi-user access to a number of databases. It enables easy to use Shell (bash) for creating tables, querying tables, etc. Another advantage of using MySQL is that it is easy to use with Java JDBC (Java Database Connectivity).

Data security, on-demand scalability, high performance, round-the-clock pptime, comprehensive transactional support, complete workflow control, reduced total cost of ownership and the flexibility of Open Source are some of the other features that make MySQL an attractive and popular tool. For these reasons, MySQL is frequently used by PHP and Perl. The importance of MySQL can also be gauged by the fact that many of the world's largest and fastest-growing organizations; such as, Facebook, Twitter, Booking.com, and Verizon rely on MySQL to save time.

This is why we, in our project, have also used MySQL. Although we have kept our project simple, it has potential to be expanded and might have to use large data. In order to communicate with the consumer, our robot 'Robbi' might have to scan through large data.

3.5.1 Characteristics

MySQL works in *client/server systems*: a system that consists of a multithreaded SQL server that supports different back-ends, several different client programs and libraries, administrative tools, and a wide range of Application Programming Interfaces (APIs).[[dAm22b]]

MySQL also works in *embedded systems*: they provide MySQL Server with an embedded multithreaded library that can be linked to an application in order to get a smaller, faster and an easier-to-manage standalone product.

3.5.2 Architecture

The following diagram depicts the architecture of MySQL DBMS.

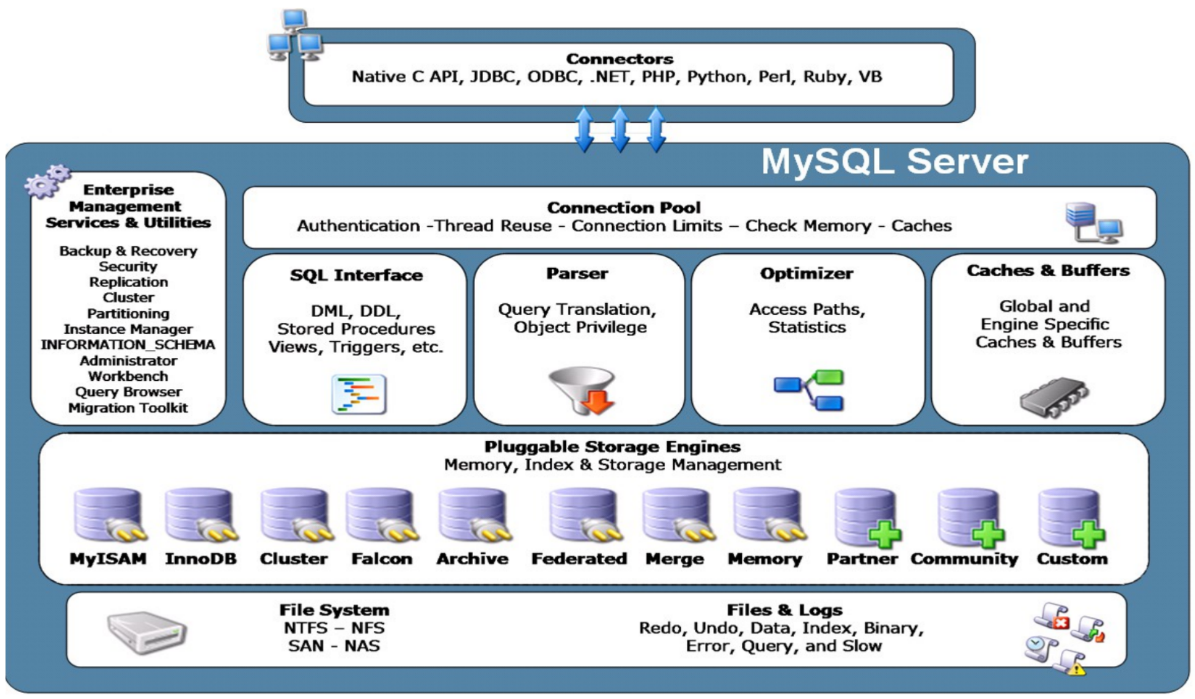

Connectors
Native C API, JDBC, ODBC, .NET, PHP, Python, Perl, Ruby, VB
MySQL Server
Enterprise Management Services & Utilities
Backup & Recovery
Security
Replication
Cluster
Partitioning
Instance Manager
INFORMATION_SCHEMA
Administrator
Workbench
Query Browser
Migration Toolkit
Connection Pool
Authentication -Thread Reuse - Connection Limits – Check Memory - Caches
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Query Translation, Object Privilege
Optimizer
Access Paths, Statistics
Caches & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
MyISAM
InnoDB
Cluster
Falcon
Archive
Federated
Merge
Memory
Partner
Community
Custom
File System
NTFS – NFS
SAN - NAS
Files & Logs
Redo, Undo, Data, Index, Binary, Error, Query, and Slow

Figure 1: MySQL Architecture. [dAm22a]

3.5.3 Workbench

MySQL Workbench provides a DBA, developer, or data architect with the tools to visually design, generate, operate and manage basically all types of databases; such as, Web, OLTP, and data warehouse databases. It contains all the mechanism a data-modeler needs for creating complex Entity-Relationship-Models (ER-Models). Along with this, it also delivers key features for performing complex change-management and documentation tasks that are usually time-consuming.

Database Documentation:
Documenting database designs can be a time-consuming process. MySQL Workbench includes *DBDoc* that enables a DBA or developer to deliver point-and-click database documentation. Models can be documented in either HTML or plain text format, and includes all the objects and models in a current MySQL Workbench session.

3.5.4 Connecting to MySQL

MySQL is an interactive program that enables us to connect to a MySQL-server, run queries and view the results. MySQL may also be used in batch mode: place your queries in a file beforehand, then tell MySQL to execute the contents of the file.
To display a list of options provided by MySQL, we could use the following command: mysql –help

How to enter and edit commands?
Prompt mysql>, issue a command, MySQL sends this command to the server for execution, displays the results and prints another mysql>. A command could span multiple lines, it normally consists of an SQL statement followed by a semicolon.

Basic Queries

Once logged in, we can try some simple queries; for instance,

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
+-----------+--------------+
| 3.23.49   | 2002-05-26   |
+-----------+--------------+
1 row in set (0.00 sec)
```

Most MySQL commands end with a semicolon (;). MySQL returns the total number of rows found, and the total time to execute the query. Keywords may be entered in any letter-case. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here's another query; it demonstrates that we can use MySQL as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-------------+---------+
|    0.707107 |      25 |
+-------------+---------+
```

We can also enter multiple statements on a single line. Just end each one with a semicolon.

```
mysql> SELECT VERSION(); SELECT NOW();
+--------------+
| VERSION()    |
+--------------+
| 3.22.20a-log |
+--------------+
+---------------------+
| NOW()               |
+---------------------+
| 2004 00:15:33 |
+---------------------+
```

MySQL determines where our statement ends by looking for the terminating semicolon, not by looking for the end of the input line. Here's a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+--------------------+--------------+
| USER()             | CURRENT_DATE |
+--------------------+--------------+
| joesmith@localhost | 1999-03-18   |
+--------------------+--------------+
```

Following is some of the basic syntax on how to create a table, insert, delete, replace entries and entering basic queries.

```sql
CREATE TABLE demo (
  name VARCHAR(256),
  age INTEGER
  );
INSERT INTO demo values ('me', 40);
INSERT INTO demo values ('you', 30);
```

```

SELECT * FROM demo
WHERE name = 'you' OR age = 40;

SELECT age, name
FROM demo;

UPDATE demo
SET name = 'YOU'
WHERE name = 'you';

DELETE FROM demo
WHERE name = 'YOU';

DROP TABLE demo;
```

Listing 18: MySQL Syntax; Tables

Information on databases and tables could be displayed with the help of the following commands:

```
Listing the databases on the MySQL server host
 mysql> show databases;

Access/change database
 mysql>Use [database_name]

Showing the current selected database
 mysql> select database();

Showing tables in the current database
 mysql>show tables;

Showing the structure of a table
 mysql> describe [table_name];
```

Listing 19: Information on databases and tables

3.6 PHP

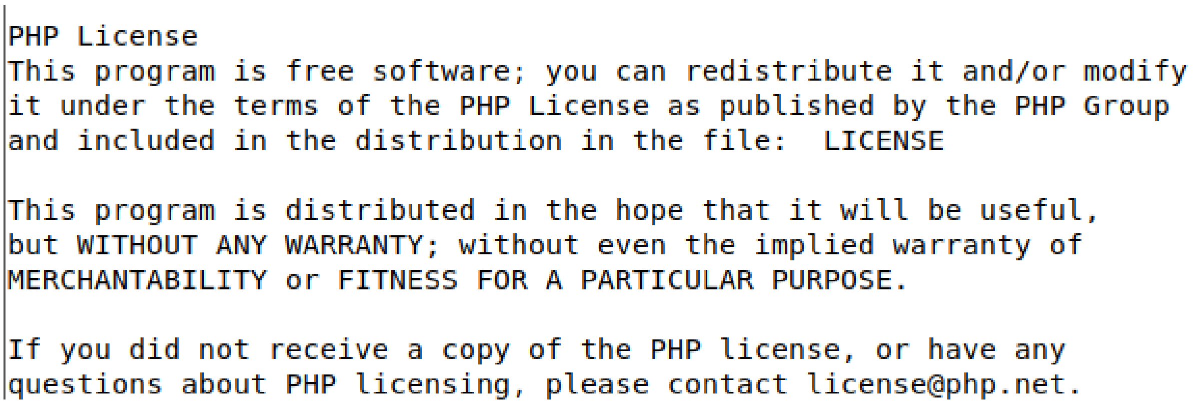
by Aliah Bahmann

PHP, firstly developed by Rasmus Lerdorf in 1994, is a recursive acronym for Hypertext Preprocessor. It is a server-sided scripting language, meaning that programs created in it execute on web servers and are not dependent on an online browser. The syntax of the PHP language is similar to that of C. It is regarded as an incredibly efficient technology that offers a simple development process with lots of additional resources to help it. Because it is platform-neutral, it can operate on any OS, including UNIX, Linux and Windows. Its greatest advantage is its performance: Particularly when built as an Apache module on the Unix side, PHP runs pleasantly quickly. Once it is running, the MySQL server performs even the most advanced queries with enormous result sets in a lightning-fast manner. [al22]

The main context we were given while learning the basics of PHP programming is for PHP to add dynamic content to a HTML-website, but its abilities do not stop there, it can also be used to

- create, open, read, write, delete and close files on the server
- collect form data
- send and receive cookies
- add, delete and modify data in a wide variety of databases
- control user-access
- encrypt data.

[W3Se]



```
PHP License
This program is free software; you can redistribute it and/or modify
it under the terms of the PHP License as published by the PHP Group
and included in the distribution in the file:  LICENSE

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

If you did not receive a copy of the PHP license, or have any
questions about PHP licensing, please contact license@php.net.
```

Figure 2: Information on PHPs License after using ”php -i” command (on hopper)

A valuable debugging tool since it contains all EGPCS (Environment, GET, POST, Cookie, Server)-data is ”phpinfo()”. It is commonly used to check configuration settings and for available predefined variables on a given system. Since its usage takes three steps (Creating an info.php file, writing ”phpinfo()” in it and opening ”path/info.php” via browser), I used ”php -i” on the command line instead to get the same information. [Gro]

3.6.1 Basic syntax-introduction

As in any other programming language there is a syntax to hold onto. PHPs syntax is, as mentioned above, similar to C language, but one still needs to get used to some particularities.

```php
<?php
$numa=78;
$numb=45;
$text="Text can be saved in a variable like this";
echo "$numa - $numb + $numa =".($numa-$numb+$numa);
//Results in "78-45+78=111" as output
?>
```

Listing 20: Example of a mathematical operation in PHP

As seen here variables are declared and defined in one line, which I find easiest, especially in regards to the fact that variables in PHP do not have to be defined as a certain type. PHP does support strings, integers, floats/doubles, booleans, arrays, objects, NULLs and resources as data types. [W3Sf] Variables are created in the moment one assigns a value to them. They are called by their names, the same way they were declared and defined. (Other programming languages that we learned, like Java, require a separate declaration and definition of a new variable.)

To output data on the screen PHP has the echo- and the print-statement, only differing in their return values (none for "echo", one for "print", making "print" usable in expressions) and their ability or inability to take multiple arguments: Even though such usage is rare, "echo" can take multiple arguments, while "print" can only take one. [W3Sa]

PHP has four types of loops:

- while-loop
  Used to iterate through a code block as long as a specified condition is true.
- do..while-loop
  Used to iterate through a code block once, to then repeat the loop as long as a specified condition is true.

```php
 <?php
$x = 28;
do {
echo "The number is: $x <br>";
$x++;
} while ($x <= 32);
//Output:
//The number is: 28
//The number is: 29
```

```
    //The number is: 30
    //The number is: 31
    //The number is: 32
    ?>
```

Listing 21: Exemplary representation of a do..while-loop in PHP

- for-loop
  Iterates through a code block a specified amount of times.
- foreach-loop
  Iterates through a code block for each element of an array.

```
    <?php
    $daysoftheweek = array("monday", "tuesday", "wednesday", "
       thursday");
    foreach ($daysoftheweek as $value) {
    echo "$value <br>";
    }
    //Output:
    //monday
    //tuesday
    //wednesday
    //thursday
    ?>
```

Listing 22: Exemplary representation of a foreach-loop in PHP

A sequence of characters, a string, can be edited and manipulated with PHPs string functions:

```
<?php
//Returning the length of a string:
echo strlen("Counting not the words but the letters");
//Output: 38
-----
//Counting the words in a string:
echo str_word_count("A string is a string is a string");
//Output: 8
-----
//Reversing a string:
echo strrev("Learning by doing is a great mantra");
//Output: artnam taerg a si gniod yb gninraeL
-----
//Searching for a text within a string:
echo strpos("999 998 997 996", "996");
//Output: 12
-----
//Replacing a text within a string:
echo str_replace("former", "new", "Mr former president");
//Output: Mr new president
```

```
?>
```

Listing 23: String functions in PHP

It is important to note, that, for the results to be displayed, the code must be embedded into HTML. Using include- and require-statements is a good option to do so, since they take all of the code, markup and text that exist in a specified file and copy it into the file that uses the statement. This is especially handy when you want to reuse components of a built webpage - like the footer or a menu bar. The only mayor difference between the include- and the require-statement is their behaviour upon failure, since the require-statement will produce a fatal error and stop the script while the include-statement only produces a warning and the script continues. Deriving from this information, ”require” is best used when the file is required by the application, ”include” is preferably used when the application is supposed to continue working even with the file missing. [W3Sd]

```
    <html>
    <head>
        <meta-charset='uft-8'>
        <title> example </title>
    </head>
    <body>
        <?php require 'predefinedvariables.php';
        echo "We still have $quantity of $thing in stock!";
        ?>

        <img src="folder/picture.png">

        <?php include 'predefineddescriptiontext.txt';
        echo readfile ("predefineddescriptiontext.txt");
        ?>
    </body>
    </html>
```

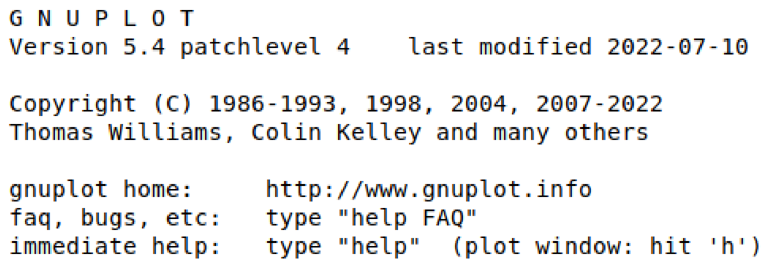
Listing 24: Integrating PHP code in a HTML website using include- and require-statements

Another interesting feature of PHP is, that it provides two methods through which the client (or browser) can send information to the server: The GET-method and the POST-method.

In order to submit form data to the server, the HTTP request methods GET and POST are utilized inside the form-tag. When using a browser as the client and an application operating on the computer system hosting your website as the server, HTTP protocol facilitates communication between the client and the server. The data from the HTML form is submitted using the GET method. The predefined GET variable is used to gather this data and process it. The URL will display all of the variable names and their values because the data supplied from an HTML form using the GET method is available to everyone in the browsers address bar. Also, only a limited amount of max. 2048 characters can be transmitted. The GET method has the benefit of allowing you to bookmark a website with a specific query string since the information you send is displayed in the URL. Additionally, GET requests can be cached and are always stored in the browsers history. Disadvantages of the GET method are the limited amount of data that can be sent, the unsuitability for transferring sensitive data as well as the inability to send binary data (such as images or word documents) to the server. The POST method, on the other hand, is also used to submit form data, but unlike the GET method, it does not have a limit on the amount of information to be sent nor is the transmitted data visible to the user. Thus, the POST-method is more suitable for sending sensitive information, even though the data security depends on the HTTP protocol, because the information sent using the POST method is passing the HTTP header. By using a secure protocol, data security is ensured. Next to POST methods advantages, it also has disadvantages: POST requests do not cache, they do not remain in the browser history since they are transmitted invisibly and it is not possible to bookmark the page because the variables are not displayed in URL. [jav]

3.7 Gnuplot

by Aliah Bahmann



```
G N U P L O T
Version 5.4 patchlevel 4    last modified 2022-07-10

Copyright (C) 1986-1993, 1998, 2004, 2007-2022
Thomas Williams, Colin Kelley and many others

gnuplot home:     http://www.gnuplot.info
faq, bugs, etc:   type "help FAQ"
immediate help:   type "help"  (plot window: hit 'h')
```

Figure 3: Gnuplot version installed on hopper

Gnuplot is a portable, command-line-driven graphing tool that runs on many different operating systems. The source code is copyrighted but it is freely available for download. Although it was initially developed to assist interactive visualization of mathematical functions and data by scientists and students, it has now expanded to cover various non-interactive purposes, such as web scripting. Gnuplot has received funding and has being actively developed since 1986. Its most recent version is 5.4.4. (July 2022). [22] Numerous plot types are supported by Gnuplot in both 2D and 3D. It can draw utilizing a variety of related text as well as lines, points, boxes, contours, vector fields, and surfaces. Additionally, it supports a number of specialty plot kinds and output formats, including direct output to pen plotters or contemporary printers, interactive screen terminals (with mouse and hotkey input), and output to numerous file formats (eps, emf, fig, jpeg, LaTeX, pdf, png, postscript, etc.). Its command language is case sensitive. [Kel21]

Taught modus operandi

- plot.gp
  The ending .gp standing for Gnuplot, this file will later be executed using the tool. In this file everything regarding the appearance of the later generated plot (via set-command), the file from which the data that is to be plotted originates (input.dat) as well as the desired output-file is declared (output.png).
- input.dat
  In this step the command line is used to generate as much data as needed, which one can do writing a loop that does two or more equations a number of times until a desired amount of rows and lines is accomplished. The results are lead into the input.dat file.
- output.png
  Lastly, typing ”gnuplot plot.gp” on the command-line will exert the plot and will either result in an error or in a new output.png in the working directory.

[Rad]

Syntax

I will use the following text passage to dig deeper into gnuplots syntax by decribing how the code in the listing affects the resulting plot:

```
set terminal pngcairo
#The set terminal has to be chosen based on the desired output-format.
set output "multiplot.png"
#The output file is named here and will be saved in the working directory.

set samples 1000
#Default sampling rate is 100.
#A higher number produces a more accurate plot, but takes longer.
set xzeroaxis
#The x axis may be drawn by setting this and removed by unsetting this.
set multiplot
#Gnuplot now knows that we plan on this to be a multiplot.
```

```

set origin 0,0
#Sets the plots origin - always relative.
set size 1,1
#Size and origin are used to position the plot.
set xrange [-15:15]
#The range portrayed on the x axis of this plot.
plot cos(x) * x**2, sin(x) * x**2
#Different approach than we got introduced to.
#The equation is directly put into the script, no separate .dat.

set nokey
#Enabling a key would place the legend on the plot,
#disabling that is best for multiplots.

set origin 0.3, 0.07
set size 0.5, 0.45
set xrange [-2:2]
set yrange [-4:2]
set xtics 1
set xtics 2
#Tics are setting the borders for the plot.
replot
#Commands the re-plotting of the equation using the newly sets attributes.

set origin 0.45, 0.14
set size 0.3, 0.2
set xrange [-0.1:0.1]
set yrange [-0.002:0.004]
set ytics 0.002
set xtics 0.1
replot

unset multiplot
#disables multiplots for the following scripts.
```

Listing 25: multiplot.gp

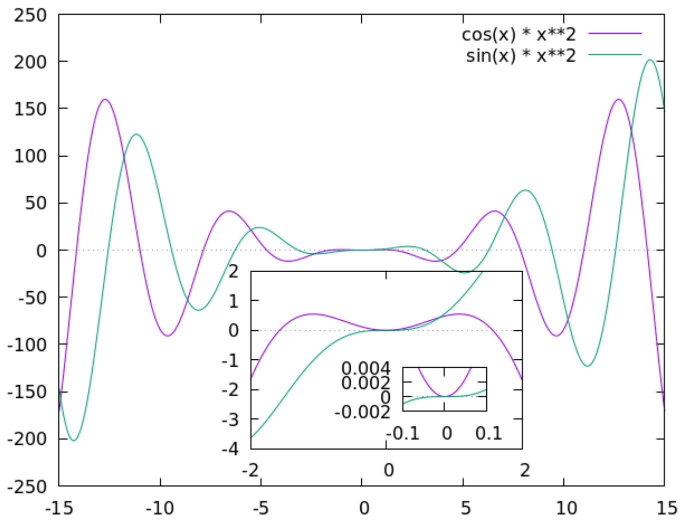


Figure 4: A multiplot with a zoomed in subplot made gnuplotting multiplot.gp

3.8 JavaScript

by Muhammad Zahid

Originally introduced as 'LiveWire', later on 'LiveScript' and created by 'Netscape', JavaScript is a *client-side* scripting language. The term 'Client-side' refers to the fact that it is executed in the client (software) that the viewer is using. In the case of JavaScript, the client is the browser. The scripting languages, such as, PHP and Python, are *server-side* languages and they run on the Web-server. JavaScript is interpreted *on-the-fly* by the client and each line is processed as it loads in the browser.
Though Similarly-named, Java and JavaScript are quite different types of languages. There are similarities in syntax because they both descend from C. Among other differences, JavaScript-programs being interpreted in the browser, while Java-programs being compiled and can be run as stand-alone applications, is an important one.

Even though PHP already allows us to create dynamic web-pages, there are several reasons to why we also use client-side (JavaScript) scripting. It provides *usability*, which means that it can modify a page without having to post back to the server (faster UI). Client-side scripting is *efficient* because it can make small, quick changes to a page without having to wait for the server. It is *event-driven*; i.e., it can respond to user actions, like clicks and key presses. The server-side programming (PHP), on the other hand, provides us with security, compatibility, power and can run back-ground processes.[Mar22]

Alongside all the advantages using JavaScript, there are some limitations; e.g., in order to run JavaScript, we must have a JavaScript-enabled browser. JavaScript requires a client who has enough trust in the server to run the code the server provides. Although JavaScript provides us with some protection, it is not fully secure and can cause security problems for clients. Not to forget that JavaScript was first developed in 1995 and web-browsers have come a long way since. [Mor22]

Most of the code and text in this chapter has been inspired by the tutorials of Prof. Dr. Radfelder [Rad22], Prof. Dr. Morin [Mor22], and by [Mar22].

3.8.1 JavaScript in HTML

There are two ways to add JavaScript to Web pages. Use the <script>…</script> tag or including the script in an external file. The following text explains how JavaScript can be directly inserted into the document by using the SCRIPT tag:

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World in JavaScript</title>
    </head>
    <body>
        <script type= "text/javascript">
            document.write("Hello World!");
        </script>
    </body>
</html>
```

Listing 26: JavaScript in HTML 1

Any number of Scripts can be placed in the HEAD or in the BODY. In the HEAD, scripts are run before the page is displayed, while in the BODY, scripts are run as the page is being displayed. To define functions, the right place is in the HEAD and variables, that are used by scripts, are defined within the BODY.

```html
<!DOCTYPE html>
<html>
    <head>
    <title>Hello World in JavaScript</title>
    <script type="text/javascript">
        function helloWorld() {
            document.write("Hello World!");
        }
    </script>
    </head>
    <body>
```

```
        <script type="text/javascript">
            helloWorld();
        </script>
    </body>
</html>
```

Listing 27: JavaScript in HTML 2

Often the scripts are long and complicated with a set of subroutines, which are being used in several different documents. For such scenarios, it is useful to write scripts in external files, which can be loaded into as a script-file with the help of the following simple command:

```
<script src="filename.js" type="text/javascript"></script>
```

3.8.2 Syntax

JavaScript **Variables** are declared with the optional keyword, var. Variables declared within a function are local to that function and those declared outside any function are global variables.

```
var myname = "Robbi Robotermann";
```

JavaScript is an **Event-driven** programming. Usually we start a program with a main method or implicit main, like in PHP, however, JavaScript programs wait for user-actions (*events*) and respond to them accordingly. Event-driven programming means, writing programs driven by user events.

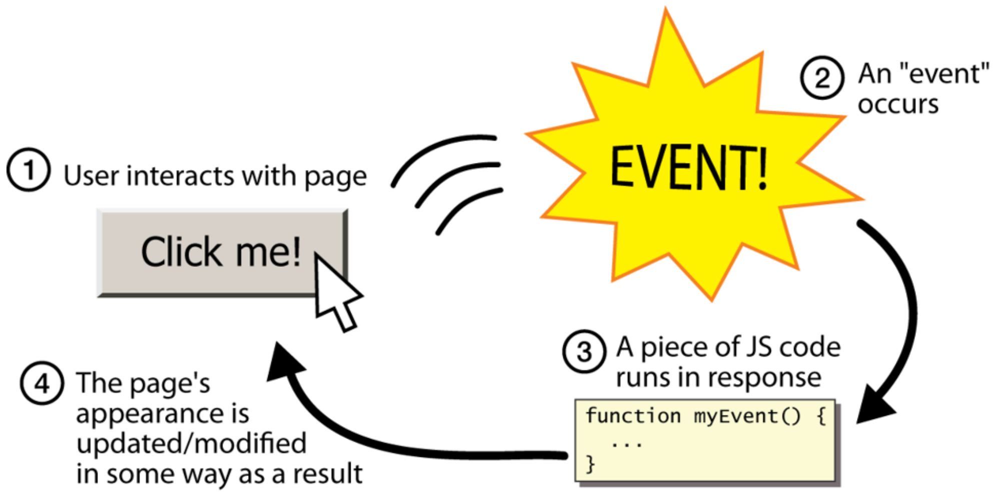

① User interacts with page
Click me!
EVENT!
② An "event" occurs
③ A piece of JS code runs in response
function myEvent() {
...
}
④ The page's appearance is updated/modified in some way as a result

Figure 5: Event-driven Programming [web22]

JavaScript Functions are defined using the keyword, *function*. Functions can return values using the keyword, *return*.

```
function square(x) {
  return x * x;
}
const demo = square(3);
// demo equals 9
}
```

JavaScript has **arrays** that are indexed starting at 0. In JavaScript, how a *for-loop* iterates over an array is shown in the following syntax.

```
<script type="text/javascript">
    var colors = new Array();
    colors[0] ="red"; colors[1] ="green";
    colors[2] ="blue"; colors[3] ="orange";
    colors[4] ="magenta"; colors[5] ="cyan";
    for (var i in colors) {
        document.write("<div style=\"background-color:"
                        + colors[i] + ";\">"
                        + colors[i] + "</div>\n");
    }
</script>
```

Listing 28: Arrays in JavaScript

3.8.3 DOM

DOM stands for 'Document Object Model' and is a programming API (Application Programming Interface) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. DOM is not specific to JavaScript only. There are several kinds of HTML DOM objects; **the environment objects**; e.g., Document, Screen, Window, Navigator, History, Location, while the **HTML objects** being: Password, Radio, Reset, Submit, Text, Link, Meta, Object, Anchor, Area, Base, Body, Button, Event, Form, Frame, Option, Select, Style, Table, TableCell, TableRow, TextArea, Frameset, Iframe, Image, Checkbox, FileUpload, Hidden. The **Document object** represents an HTML document and can be used to access all the documents in a page. A Document contains several collections; such as, anchors, forms, images, links. Some of the properties of the *document object* are: body, cookie, domain, lastModified, referrer, title, URL. The *Document object* has several useful methods: getElementById, getElementsByName, getElementsByTagName, write, writeln, open, close. Following is an *instance* of *Document object* called *document*:

```
function changeF() {
    var cText = document.getElementById("c");
    var fText = document.getElementById("f");
...
}

...

<input type="text" id="c" onchange="changeC()">C
<input type="text" id="f" onchange="changeF()">F
```

Listing 29: HTML DOM, Document

The other important DOM HTML objects are **Text**, **Form Elements** and the **Document Tree**. Having access to the *elements* and changing their properties gives us the freedom to perform simple manipulations; such as, form validation, data transfer etc. HTML DOM, however, enables us to do much more than just these simple tasks; we can create, delete, and modify parts of the HTML document. In order to understand that, we need the **Document Tree**.

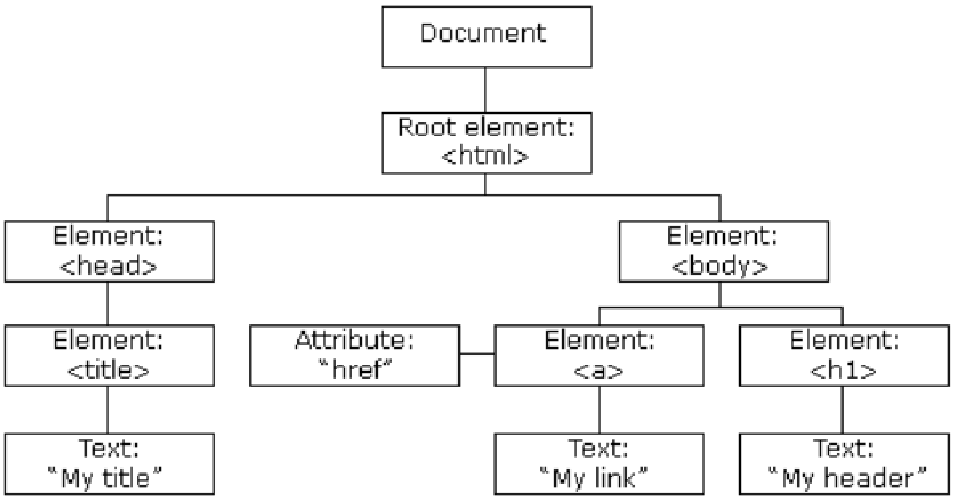


Figure 6: HTML DOM Document Tree [Mor22]

JavaScript enables us to **navigate** through the document tree with the help of the following elements:

```
document.getElementById(),

getElementsByName(), and

getElementsByTagName() returns nodes in the document tree.

The required information can be extracted using:

nodeName - The tag name

nodeValue - The the text of a text node

nodeType - The kind of node.
```

3.8.4 Ajax

by Aliah Bahmann

Ajax, acronomous to ”Asynchronous JavaScript and XML” is a client-sided technique that enables to load single parts of a website asychronously, when needed, to create a dynamically reacting webpage. (Its name is a little misleading though, in my opinion, since XML can be used to transport data, but plain or JSON text is equally common.) The shown contents can be manipulated, without loading the whole page anew. It is a non-standardized method of developing online applications that communicate with the web server using client-side scripting and therefore are interactively customizable. It allows for ressource-aware programming since the amount of retrieved data is controlled, therefore favours needful and frugal handling of resources since only requested contents are added and the rest of the page remains the same. [Gon]

As I mentioned it being a technique instead of a technology, it consists of several technologies that create Ajax’ functionality:

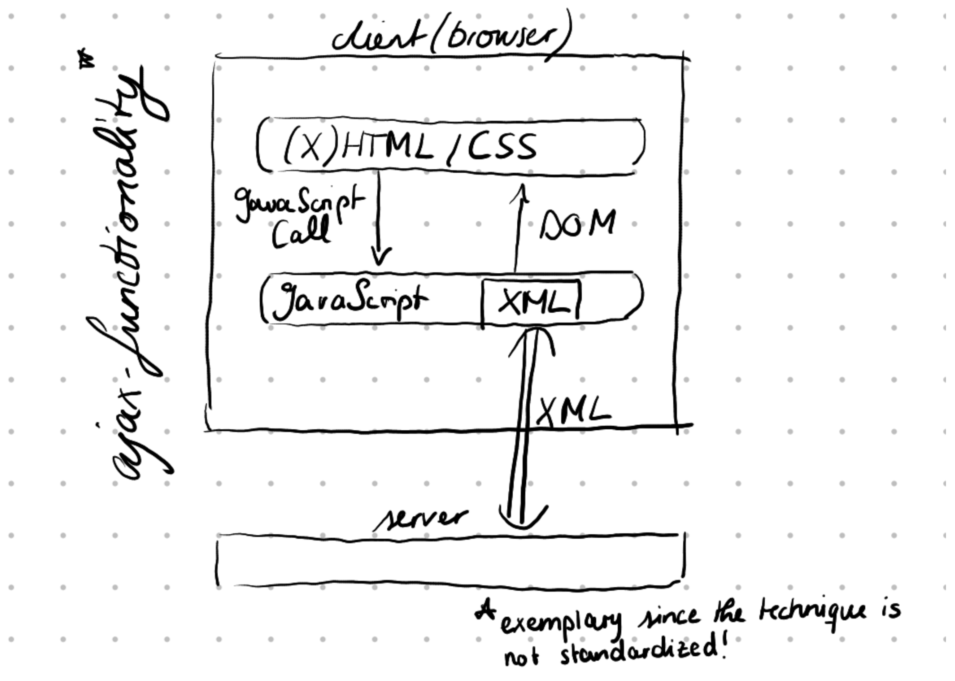


Figure 7: Ajax-components

- (X)HTML/CSS
  For presentation of content.
- JavaScript
  To modify the Document Object Model (DOM) and to access the XMLHttpRequest-API.
- XMLHttpRequest()
  For asynchronous connection between client and server.
- XML
  For exchange of information between client and server. [BW]

XHTML

Stands as an abbreviation for Extensible Hypertext Markup Language and is a stricter, more XML based version of HTML in regards of error-handling. It can be considered a family of XML markup languages, that mirrors or extends versions of the widely used HTML. XHTML uses a restrictive subset of XML that needs to be parsed with standard XML parsers.

As it is standardized by the W3C (World Wide Web Consortium), using it has certain benefits:

- Validation through standard XML tools.
- Easy to maintain, convert or edit the document over a larger span of time.
- Generates cleaner code because of stricter syntax.
- Less usage of bandwidth because of leaner format (even though only relevant for very large projects).
- Corresponds to the high quality standard of the W3C. [Gee]

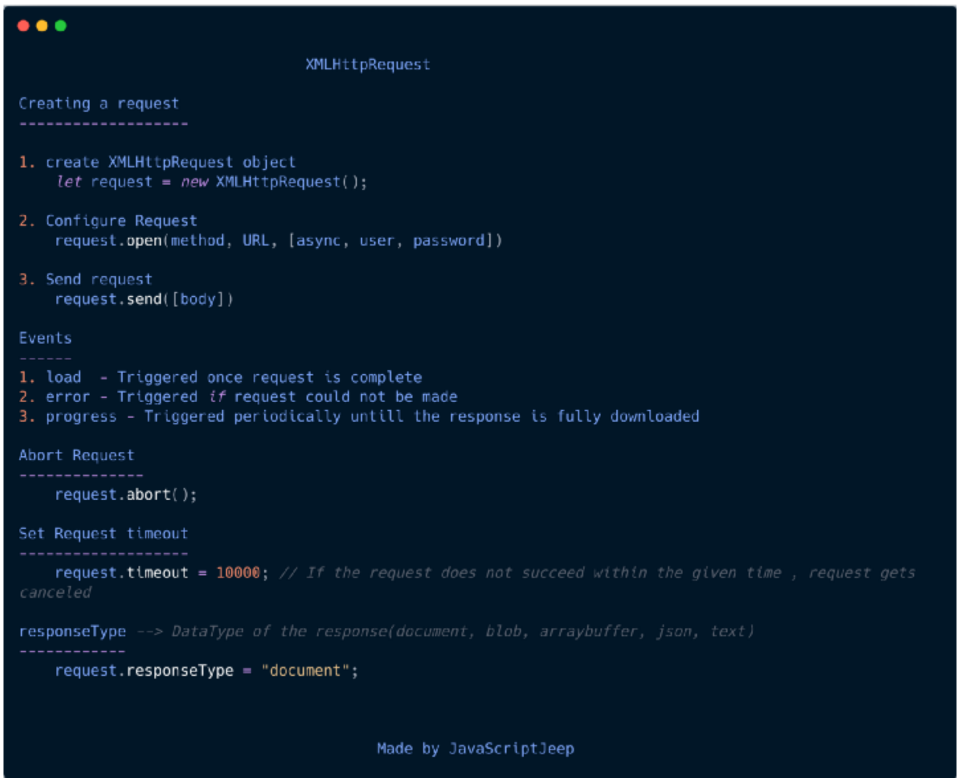
HTMLs syntax is resembled in XHTMLs syntax to a wide extent, XHTMLs syntax being a lot more meticulous concerning tag-closure, correct nesting of tags and mandatory elements. [W3Sb]

XML

Simple, text-based format for representing structured information like documents, data, configuration and books that is called Extensible Markup Language. It was derived from an older standard format named SGML (standardized through ISO 8879) and is designed to carry data and be self-descriptive without predefined tags, unlike HTML. XML is case-sensitive, which is important to know. [W3Sc] The flexibility of XML provides several advantages: It enables transfer of data between business databases and websites while preserving essential descriptive information. It allows to automatically alter the way data is presented. Additionally, it improves the efficiency of searches because search engines can sort through specific tags rather than lengthy text passages. [Roc]

XMLHttpRequest

Shortened to XHR, the XMLHttpRequest is a built-in object of JavaScript, that is used to retrieve data from an URL, either synchronously or asynchronously, without having to do a full page refresh. XHRs ability to do so is the most vital part of the Ajax-technique. [Jee] Even though it is possible to do both synchronous and asynchronous requests, asynchronous requests should be preferred, since synchronous requests block the further execution of code which causes the webpage to be unresponsive every time a request is made. Asynchronous requests receive a callback when the data has been received which lets the browser continue to work as usual while the request is being handled. [MDN]



```
                          XMLHttpRequest

Creating a request
-------------------

1. create XMLHttpRequest object
    let request = new XMLHttpRequest();

2. Configure Request
    request.open(method, URL, [async, user, password])

3. Send request
    request.send([body])

Events
------
1. load  - Triggered once request is complete
2. error - Triggered if request could not be made
3. progress - Triggered periodically untill the response is fully downloaded

Abort Request
--------------
    request.abort();

Set Request timeout
-------------------
    request.timeout = 10000; // If the request does not succeed within the given time , request gets
canceled

responseType --> DataType of the response(document, blob, arraybuffer, json, text)
------------
    request.responseType = "document";


                              Made by JavaScriptJeep
```

Figure 8: Using XMLHttpRequests

4 Prototype

Our prototype, affectionately called ”Robbi”, is a chat-bot, that we created using some of the technologies we learned this semester. This chapter is dedicated to Robbis development as a project.

URL	https://informatik.hs-bremerhaven.de/docker-infra-2022-e-web/robbi/

4.1 Planned Features

by Silas Rathgeber

After finding our project idea, we brainstormed about features that Robbi, our chat-bot could possibly have. This overview shows all the features that this meeting yielded and whether we've gotten around to implementing them yet.

Feature	Implemented	Possible Improvement
Chatbot which can handle basic conversations.	✓	
Login with opportunity to signup.	✓	
Function to reset the password per email.		✓
Sessions to check if the user is logged in or not.	✓	
Bot learns from the user if there is no suitable answer.	✓	
Bot can redirect to staff-chat.		✓
Function to save the database in the repository as MySQL-Dump.		✓
Function to correct Robbis answers.	✓	
No longer use the submit tag but JavaScript to send form data of login and signup.		✓
Two factor authentication by using email or QR-Code.		✓

Table 1: Planned vs. Implemented Functions

4.2 Basics of Infrastructure

by Aliah Bahmann

For designing application infrastructure, there is no universal paradigm. Instead, an applications designers create an infrastructure that can support the special capabilities and demands of the application, that have been specified by doing requirement-modelling beforehand. Servers, data storage, networking, application monitoring, logging, and application security services are a few examples of application infrastructure elements. A particular applications functionality and architectural design will determine the application infrastructure that is required to support it. Many programmers outline the many services inside an application and how they interact using a layered application architecture model, like we did in lecture multiple times:

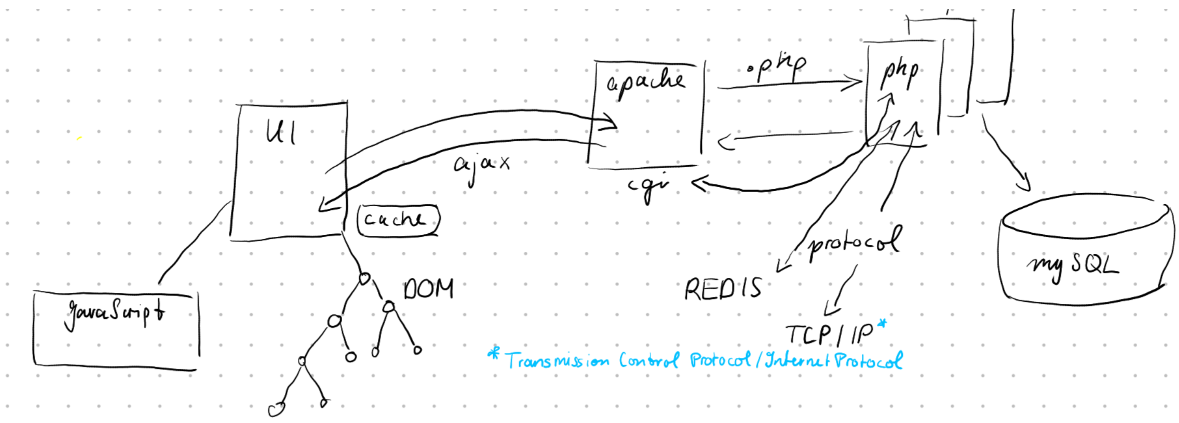


Figure 9: Depiction of connected web-architecture-components

Models like this, even though usually created using UML (Unified Modelling Language), help visualize the components of both front- and backend: The frontend, where the interaction between user and application via GUI (Graphical User Interface) takes place, is represented within this figure by JavaScript and partially DOM (Document Object Model), while the backend, that is only treated by the developer and connected to the UI (User Interface) via Ajax consists of a complex interaction of webserver, server-sided programming, transmission protocol and databases. To build an application infrastructure as depicted above, we learned to rely on LAMP:

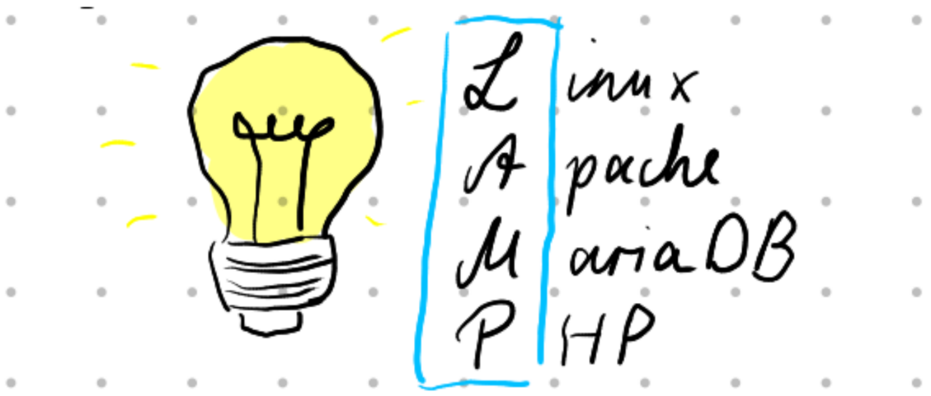


Figure 10: Visualization of ”LAMP”-model

- Linux
  Most of the servers across the world run on Linux, as its many distributions (or distros) can be fitted to a wide range of systems. It is an efficient OS, that is open-source, free, customizable and not only supports almost all of the major programming languages but also offers a great range of applications useful for programmers. [Das]

- Apache
  Apache runs on 67 percent of all websites in the world and is, therefore, the most widely used webserver software. It is open-source, free, fast, reliable and secure. By using extensions and modules it is also customizable. [Beg]
- MariaDB
  Is default in most Linux distros and one of the most popular open-source relational databases. Guaranteed to stay open-source and created by the developers of MySQL. [Fou]
- PHP
  Server-sided scripting language, meaning that programs created in it execute on web servers and are not dependent on an online browser. [Gro]

4.3 Comparison

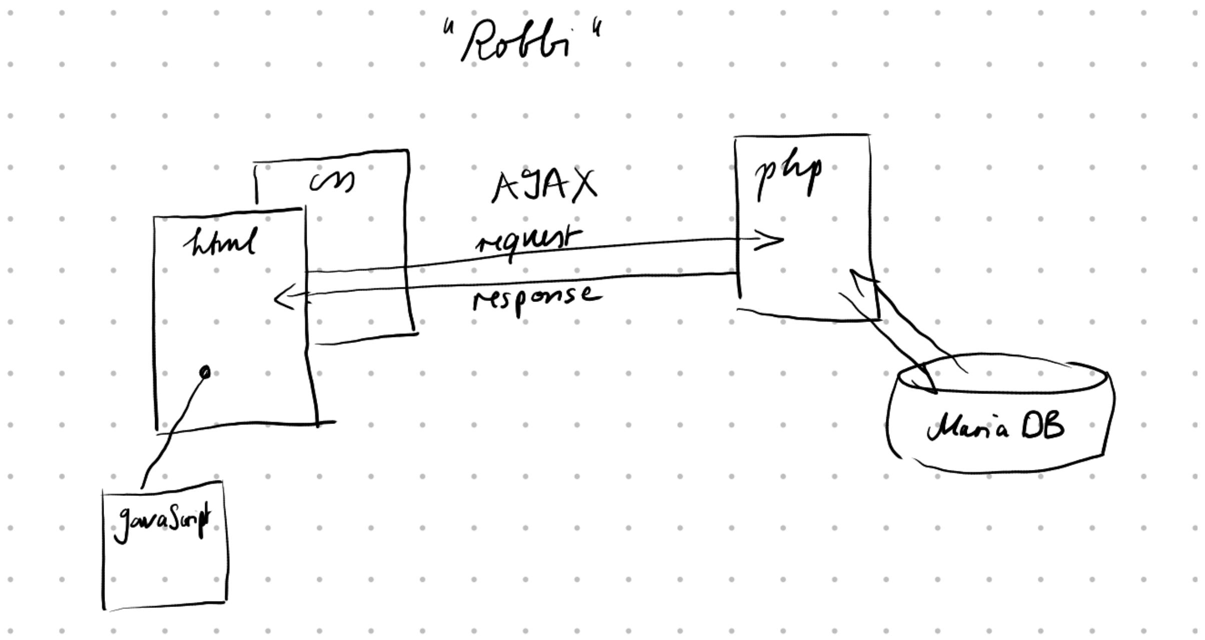


Figure 11: Graphical representation of Robbis components

As pictured in the graphic I created, Robbi actively includes six technologies, that we learned either this or last semester. His frontend consists of HTML, CSS and JavaScript, connected to the backend, made out of PHP and MariaDB, by Ajax. And alltough our project may not seem as complicated or extensive as others, since it does not include all of the technologies we got introduced to, we have done deep research on all technologies, regardless of whether or whether not they are included.

4.4 Scripts

by Silas Rathgeber

This section is dedicated to the coding we've done.

4.4.1 `createDB.sql`

This sql-script contains statements to initialize our MySQL database. Essentially we only need two tables; the first one (`bot_login`) is to save the user-data, which we collect by our sign up, while the second one (`bot_qa`) is for storing answers and questions in pairs. This is where *Robbi* searches for the suitable answers. The table (`bot_qa`) also contains the pairs of questions and answers that were originally missing from our database and are added (voluntarily) by the users of Robbi.

In order to expand or improve our database, we must reinitialize it after the changes have been made to our sql code-file. For this reason, at the beginning of the sql-file `createDB.sql`, we have added the command ”`DROP TABLE IF EXISTS bot_login`” that deletes the existing table.

The user of Robbi must provide us with the following details that are stored in the *bot_login* table:

- first_name
- name
- email
- password
- hint

These details enable us to track the number of users of Robbi. We have the possibility to send the user an E-Mail for authentication and other uses.

This following statement lets MySQL run our script:

```
mysql < createDB.sql
```

4.4.2 `db_connection.php`

In order to get access to our databases (mySQL and REDIS), we must first create a connection to our databses. For this purpose we have written two functions, `return_db_connection()` and `return_redis_connection()`, which return two connection-objects, one for the sql-database and the other for Redis respectively. In case we need a database connection, we include this file (*db_connection.php*) in our php-code by using the command `include("db_Connection.php");`. This enables us to use the functions like:

```
$conn = return_db_connection();
```

To connect to our *mySQL* instance, we use the extension *MySQLi* (i is for ”improved”) from *PHP*. This means that after creating a MySQLi connection object, we can use, for example, the following methods:

- mysqli_prepare(<*$connect object*>, <*SQL statement*>) -> to create a prepared sql statement object
- mysqli_execute(<sql statement object>) -> to execute the statement
- mysqli_stmt_get_result(<sql statement object>) -> to get the result of the statement
- mysqli_fetch_assoc(<result>) -> to get array object of the result
- mysqli_close(<*$connect object*>) -> close the connection

4.4.3 `html_parts.php`

Our project has a website with several pages. In order to give every page the same appearance, we wrote the code for the site-header and site-footer in separate functions, which can be used on every page. Since we have a login, it was necessary to have one header for the outside area and another for the inside - after the login. All of these functions contain just one `echo` statement, which prints the required code.

As we can see here, we linked the same *style.css* in each header. This way we can make sure to have a unified styling on all pages.

4.4.4 Login with Signup

At the first page (index.php) of our web application, we can see a login form. If the user does not have an account yet, there is a link, provided at the top right corner, to *sign up*. This link directs the user to another page with the sign up form.

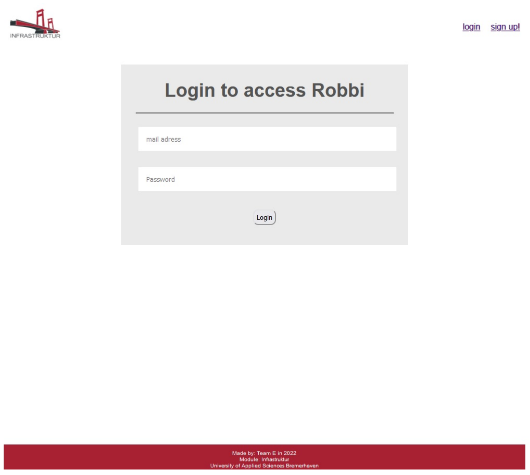


Figure 12: Homepage with Login

To fulfill the functionality of a sign up form and a login, we implemented the class `user`.

`class_user.php`

The *class_user.php* contains the class *user*. The user-class contains the functions that return either the personal details of a user or save a new user record. Furthermore, it contains one function to check a given password and another to check a given email. This class is used by the login form in the *index.php* and from the *sign_up.php*. All methods in this file work with the email address as an identifier. For instance, here is a get function for the email variable:

```php
public function get_name($email){
    $conn = return_db_connection();
    $get = mysqli_prepare($conn, "SELECT name FROM bot_login WHERE email = ? ");
    $get->bind_param('s', $email);
    mysqli_execute($get);
    $result = mysqli_stmt_get_result($get);
    $row = mysqli_fetch_assoc($result);
    mysqli_close($conn);
    return $row['name'];
}
```

Listing 30: Function of class user: get_name()

One of these access functions are used, for example, in the `is_pwd_right()` function and in the `exist_email()` function, which are both part of the user class. These get functions can also be used from outside of the class, since they are declared as `public`.

As a special feature of our password management, we want to highlight that we use functions to encrypt the input before it is saved in the database. The following snippet is out of `sign\_up.php` file:

```
$pwd_hash = password_hash($_POST["password"], PASSWORD_DEFAULT);
```

At that point (right before we call the `save_user()` function) we use the `password_hash()` function to produce a hash from the string that is in the associative array `\$_POST`, which comes right from our form (*sign_up.php*). The `PASSWORD_DEFAULT` means that the function uses the *bcrypt* algorithm to encrypt the passwords. The consequence that results from this is, that we also need a function that is able to verify if a given string matches with the decrypt hash. This function is used in `is_pwd_right()` (on line 2): `sign\_up.php`:

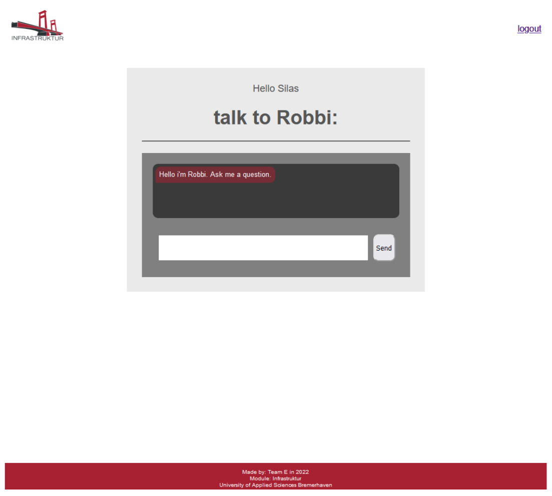
```
1 if($affected > 0){
2     if(password_verify($post_password, $password_hash)){
3         return true;
4     }
5     }else{
6         return false;
7 }
```

Listing 31: Out of `is_pwd_right()`: verify password

By storing the passwords in hash tokens, we get increased data security. Even if the database were hacked, the passwords would not be readily accessible.

4.4.5 Chat Functionality

After logging in, the user is directed to the *chat.php* that shows a chatroom.



Hello Silas

talk to Robbi:

Hello i'm Robbi. Ask me a question.

Made by: Team E in 2022
Module: Infrastruktur
University of Applied Sciences Bremerhaven

Figure 13: Screenshot of the Chat Room

As already mentioned, the inner area has its own header included and it comes from *html_parts.php*. This specific header contains a link to a javascript file named *inner_header.js*. This way we are able to react to certain events that happen on *chat.php*.

`inner_header.js`

We have four relevant functions implemented here:

- `updateScroll()`

  Is responsible for always scrolling the chat window to the highest point. This is done by

  ```
  element.scrollTop = element.scrollHeight
  ```

  Here we can see, that the height of scrolling is handed over to the `scrollTop` function.

- `printChat(toPrint, flag)`

  Creates a `<div>` tag (by using `createElement()`) and gives it to two *css* classes. The `flag` variable contains either the word *question* or *answer*. This way css can decide whether the message is displayed at the left side of the chat-box or the right. Finally, the `appendChild` function is used to add the new tag to the DOM. It ends with the call `updateScroll()`.

```
function printChat(toPrint, flag){
        var chat_object = document.createElement("div")
        chat_object.className = (flag+ " chat_object")
        chat_object.innerHTML = toPrint
        chat_object.id = flag + 1
        var chat_box = document.getElementById("dialoge")
        chat_box.appendChild(chat_object)
        updateScroll()
}
```

Listing 32: javaScript: printChat()

- `getAnswer(question)`

  This function contains the *Ajax* call for Robbi's search algorithm. Our `XMLHttpRequest` object lets us use the `GET` method to hand down the `question` variable to the dedicated php script (here *call_algorithm.php*).

  ```
  xhr.open("GET", 'call_algorithm.php' + "?question="+question,
     true);
  ```

  After receiving the answer in the`xhr.response` variable, we call the `printChat (xhr.response,answer)` method with it.

- `main()`

  This one is triggered by the submit button on the *chat.php*.

  ```
  <button type='submit' value='Send' id='sub_butt' onclick='main()
     '>Send</button>
  ```

  Here we print the entered question and call the `getAnswer(form_question)` function with it.

4.4.6 Robbis Functionality

To structure the functions of Robbi, we created a class for him. To give each user their own Robbi instance, we initialize the user variable in the `__constructor()` function with the email address from the `$_SESSION` array.

As mentioned before, our *Ajax* calls the *call_algorithm.php*. For the sake of completeness we want to show how the class is instantiated and triggered:

```
<?php
include ("class_robbi.php");

session_start();
$user = $_SESSION['email'];
$question = $_GET['question'];
```

```
7
8  $robbi1 = new robbi($user);
9  $answer = $robbi1->main($question);
10 echo $answer;
11 ?>
```

Listing 33: call_algorithm.php

As we can see in line 9, right after the creation of the new instance the response of the `main()` function is stored in `$answer`.

Reacting to Chat Events

In order to let Robbi grow and include the unknown questions and their answers, we used Redis' *Hash* datatype to store the chat history. Till now we have implemented only one scenario, in which Robbi gives the user the opportunity to teach him an answer to a question that is unknown to Robbi. Therefor, we declared five class variables, to have access from any method (from `$message1` to `$message5`).

```
class robbi{

    private $user;
    private $am_ready = "Okay, I'll save your next entry as answer to your question.";
    private $sorry_don_know = "I didn't find an answer. Do you want to help me? Say 'yes'";
    private $saved_it = "Saved new record.";
    public $message1;
    public $message2;
    public $message3;
    public $message4;
    public $message5;

    public function __construct($user){
        $this->user = $user;
    }
```

Listing 34: head of the class robbi

The most important parts of the class methods are outlined below.

- `get_message($number)` and `set_message($number, $message)`

  For class intern purposes, we needed functions that can return or set a specific variable in the Redis database.

  ```
  private function set_message($number, $message){
      $redis_conn = return_redis_connection();
      $redis_conn->HSET($this->user, "message$number", $message);
  ```

```
4     $redis_conn->close();
5 }
```

Listing 35: class_robbi.php: set_message()

As seen in line 2, we use `return_redis_connection()` to create connection to Redis. This function is implemented in *db_connection.php* just like the method for the mySQL connection. The command `HSET` expects three parameters:

∗ key

∗ field

∗ value

Accordingly, `HGET` needs only two parameters (for key an field).

- `sliding_window()`

  With our five class variables, we only ever want to monitor the 5 most recent messages in the chat. For this we use a function that inserts the new variable at the right place and moves all the other variables up by one position.

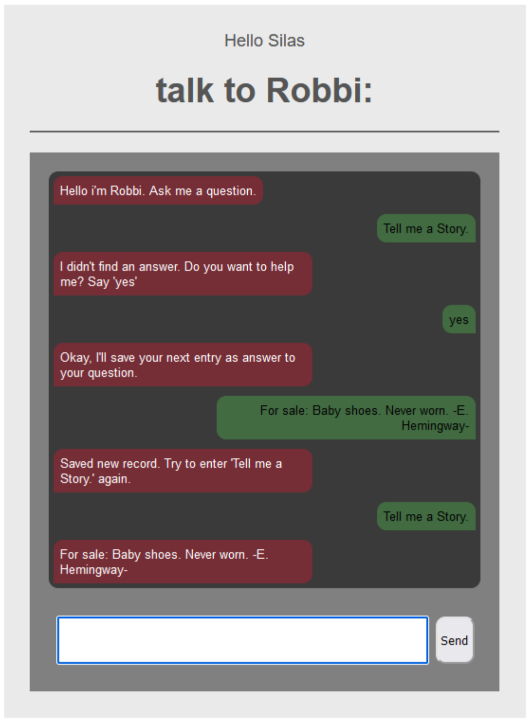
```
private function sliding_window($next){
    $this->update_message_variables();
    $this->set_message("1", $this->message2);
    $this->set_message("2", $this->message3);
    $this->set_message("3", $this->message4);
    $this->set_message("4", $this->message5);
    $this->set_message("5", $next);
    $this->update_message_variables();
}
```

  Listing 36: class_robbi.php: sliding_window()

  `update_message_variables()` just updates the class variables before and after the sliding window.

- `main()`

  This method is for reacting to different message patterns. For example, if the last Message (`message5`) is ”yes” and at the same time the penultimate message (`message4`) is equal to the string in `$sorry_don_know`, Robbi returns the content of `$am_ready` as the answer. This response must trigger a `sliding_window()` again. This way Robbi reacts to the following scenarios and so on.



Hello Silas

talk to Robbi:

Hello i'm Robbi. Ask me a question.

Tell me a Story.

I didn't find an answer. Do you want to help me? Say 'yes'

yes

Okay, I'll save your next entry as answer to your question.

For sale: Baby shoes. Never worn. -E. Hemingway-

Saved new record. Try to enter 'Tell me a Story.' again.

Tell me a Story.

For sale: Baby shoes. Never worn. -E. Hemingway-

Figure 14: Robbi asks for improvement

4.4.7 Algorithm

by Erfan Karimi

In this section, we explain how our Robbi finds an answer to an asked question. Our goal in this project is to build an artificial communication interface, where any web user can ask a computer a variety of questions and get an appropriate response. Considering its ability, Robbi inspires to be similar to an artificial intelligent device or Software. Robbie is quite primitive in our project but it could be developed into an artificially intelligent web-assistant. By adding more and more questions and their responses in our database, we can improve Robbi's performance all the time.

In order to find the answer, we thought of the search algorithm in google search engine. It doesn't compare the whole input, rather some part of it; e.g., a word. So if there isn't a one hundred percent accuracy, it just shows the next best answer or website it has found. That's almost how our search-script in PHP works. The entire SQL entries are searched for an asked question and the answer to the most matched question is returned. The need to search through every entry can make Robbi slow as the database becomes larger and larger. One way to improve the performance is that we terminate the search once an identical question in our database is matched.

```php
<?php
function search ($q){
  ini_set('display_errors', 1);
  ini_set('display_startup_errors', 1);
  error_reporting(E_ALL);

  include ("db_connection.php");
  $conn=return_db_connection();
  $sql="select bot_qa.q, bot_qa.a FROM bot_qa";
  $result = mysqli_query($conn, $sql);

  $bigger_count = 0;
  $current_count = 0;
  $current_question="";
  $target_answer="";
  $current_answer="";

  while($row = mysqli_fetch_assoc($result)) {
    $cond=0;
    foreach($row as $item){
      if ($cond % 2 == 0){ //Modulo, weil die Daten paarweise getrennt
          werden müssen
        $current_question = $item;
      }
      else{
        $current_answer = $item;
      }
      ++$cond;
    }
    if ( "$current_question" == "$q" ){
      return $current_answer;
    }
    $words = explode(" ", $current_question); // trenne die wörter im
       Array
    foreach ($words as $word) {
      //if (str_contains($q, $word)){
      if (stripos($q, $word)){
        ++$current_count;
      }
    }
    if ($current_count > $bigger_count){
      $bigger_count = $current_count;
      $current_count = 0;
      $target_answer = $current_answer;
    }
  }
  if($target_answer==""){
    $target_answer="I couldn't find an answer to your input. Do you want
       to teach me how to answer correctly? Say 'yes'";
  }
  return "$target_answer";
}
?>
```

Listing 37: php: Our search script in connection with how our chat bot finds its answers

In our script, the search query is called *$q* (question). Firstly, the script iterates over every entry and counts the number of words in the asked question that match with *$q*. The answer associated to the entry with the largest number of matches is saved as *$target_answer* and at the end is returned as the answer. During the search, if both the asked question *$q* and the question *$current_question* in our database are identical, the search is terminated immediately and the value of the *$current_answer* is returned.

4.5 Testing

by Silas Rathgeber

Testing our application gave us a little trouble. We have had a few failed attempts and to this day we still have not been able to figure out how to log into our application using the `curl` command. We worked around this problem by creating a php file that is not protected by the session.

In order to test how much computing resources our application requires, we created some scripts, which put the server under artificial load. Our start script can be controlled via the environment variables `$1,$2`, etc. Here we can specify how many runs, how many curl calls and which test scenario we want to execute.

In our Bash-scripts that we wrote to run the performance test, we used the following commands, variables and switches, among others:

Command	Short Explanation
`test -n`	The length of the string is not null.
`test -z`	The length of the string is null.
`test -e`	Returns 0 (true) if the file exists.
`echo -n`	Avoids the word wrap after the end of the echo-process.
`echo -e`	Enables echo to interpret "\n".
`curl -s`	Don't show progress meter or error messages.
`curl -c`	Gives the possibility to specify in which location the cookie of the target should be saved.
`curl -b`	Gives the possibility to specify which cookie should be sent.
`curl -X`	Gives the possibility to specify which request method should be used.
`$!`	Contains the process ID of the last started process.
`${1:-10}`	Sets the environment variable to 10 if not submitted.

Table 2: Test script commands, variables and switches

4.5.1 `startstate.sh`

This script starts the command and continuously writes its *std-out* to a specific file and the *std-err* in another. In order to observe how the docker load behaves, this program must be run before the test scenario is started and must continue to run during the test period.

4.5.2 `runTest.sh`

With this we can control which test should be executed and with which parameters it should run. Therefore, to start a test, this script must be used. As mentioned above, the environment variables can be used to select how many requests the test should run:

The first variable (`$1`) is intended for how many iterations the test should go through. The second (`$2`) is for the number of curl calls the first iteration should execute. It is important to know that the script multiplies this variable by a factor that increases by one per test cycle. So, in the second test cycle, twice the number of curl calls specified here will be executed, and in the third, three times as many, and so on. And the last one (`$3`) is for a specific curl command that we implemented in the `curl.sh`. Since we use our curl commands to address our algorithm, which processes and compares sentences, a command with a longer string is offered under test case 2 rather than in test case 1.

A possible test call could look like this:

```
./runTest.sh 4 10000 2
```

In addition, this script is responsible for detecting a failed test and making the test result available at the link below:

URL	https://informatik.hs-bremerhaven.de/infra-2022-e/test.php

4.5.3 `Test.sh`

Here we have the script that actually runs our test. It is responsible for recording the start and end times of the test. The variables for the number of curl commands and the test case variable are passed here from `runTest.sh`. From here a script is ran to create the numbers we need, as well as one that produces our timeline. And finally the plot is generated by the file *createPlotPng.sh*.

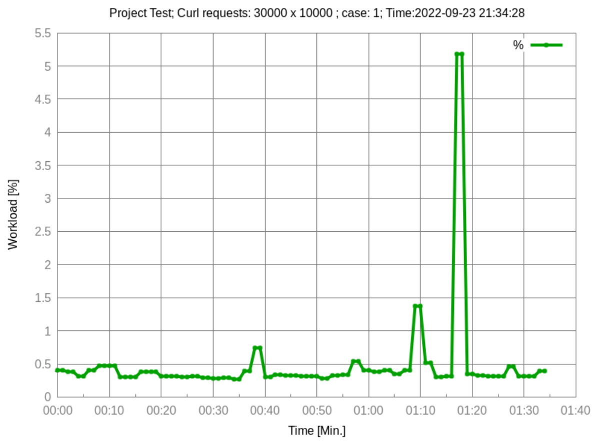
4.5.4 `curl.sh`

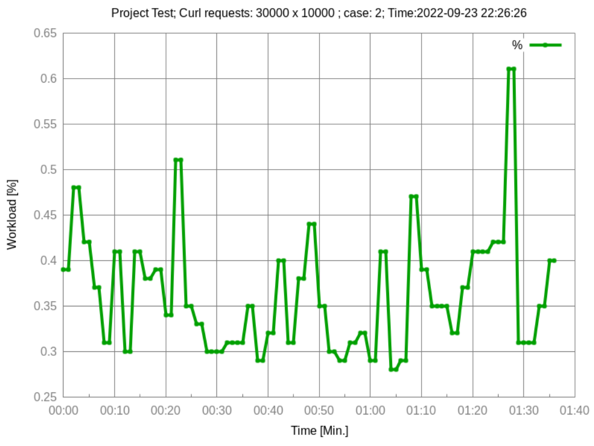
We wrote the actual curl command in this file. It receives the set of iterations to run and the specified "case". Using a switch-case statement, the script recognizes which command it has to use.Before that, however, it creates and changes to another folder in the path: /dev/shm/ to use the faster file system that is there.

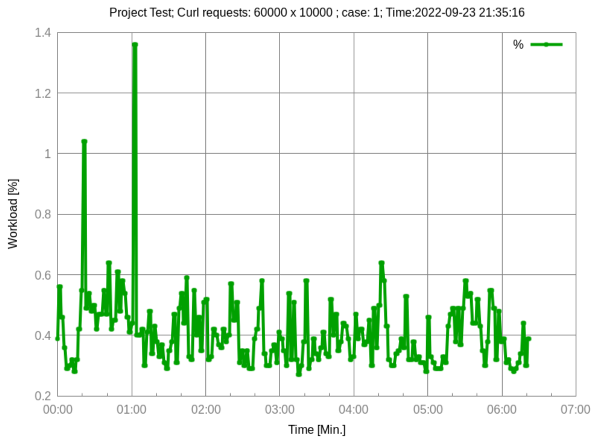
4.6 Test-Results

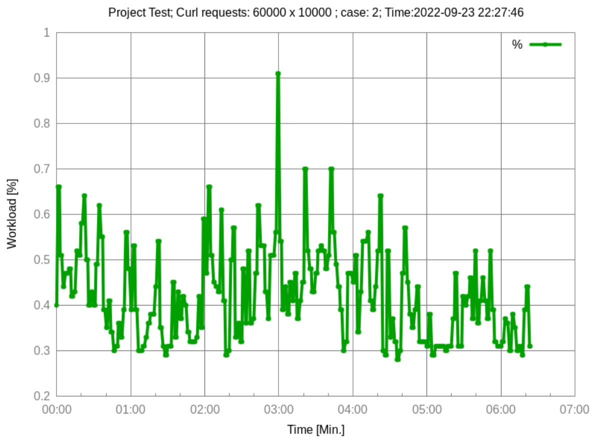
In order to get a better basis for testing, we fed Robbi's database with 1000 data sets with randomly selected words.

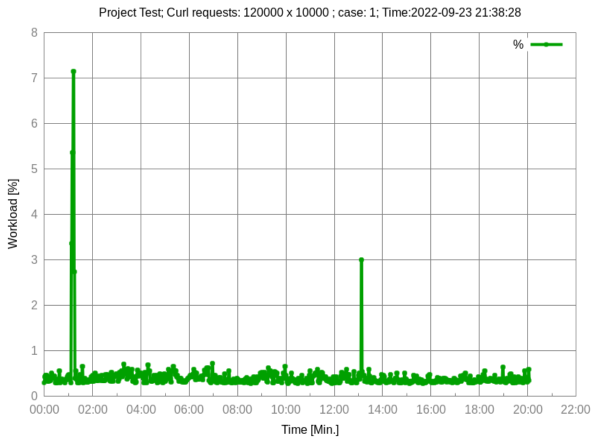
We compared two different test cases here. In the first test case we pass our algorithm a 3 word string per curl call and in the second test case it is a 20 word string. We ran the first test run with case 1 in three stages with 20,000 in the request multiplier. As a reminder, the request number is created in the script as a multiplier, which increases by one per run. So: 30000 in the first 60000 in the second and 120000 in the third run. Then the same with test case 2. We see the results here:

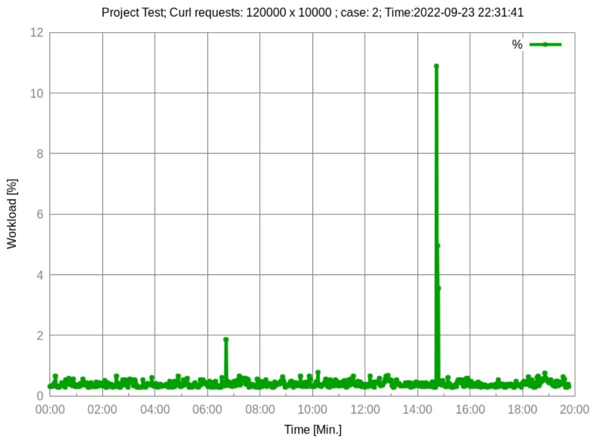












Although we don't really load the server with our curl commands, we can clearly see the difference between the test cases. On the left we have the test results of the first test case and on the right, those of the second test case.

5 Developement Workflow

by Silas Rathgeber

Although, during the semester, we did learn about ”branches” in git and we intended to use them in our workflow as well; however, we didn’t feel familiar enough with git to work with it at this level. For this reason we decided to work together on the main-branch instead of multiple development-branches.

To avoid conflicts during the ”rebase” process, we set the rule to pull before we continue working at the project. The second rule was to push at least once a day.

If we wanted to check the changes in the browser or to test some functionality, we deployed the project first; this is done with a deploy script.

5.1 Deployment

In our repository, there are scripts called ”ldeploy.sh” and ”deploy.sh”, which have a certain role in the development process of ours. Furthermore there is a directory named ”www” which includes every file that is necessary to run our project. The main task of these deploy-Scripts is to copy all the content of ”www” into the web-directory of the current User.

5.1.1 `ldeploy.sh`

”ldeploy” stands for ”local deploy”, which means that we want to deploy to the docker machine that belongs to the current user.
First of all, the script checks if the current executor is our group user, in which case the script sets the variable ”repo” on a certain absolute path; this is necessary in order to make the ”deploy.sh” run properly. After creating a folder for the data in the web directory, the script uses `rsync` to mirror the folder content from the www-directory to the web-directory. The following is the applied statement:

```
rsync -av $repo/www/ mydocker:$dst --delete --progress
```

Listing 38: Usage of rsync

The `-a` flag is a combination of multiple flags:

- `-r` ”recursively”
- `-l` considers symbolic ”links”
- `-p` considers ”permissions” of the source
- `-t` keeps the ”time” of changing the source
- `-g` keeps the right of the ”group”

”`-v`” is for (”verbose”) and shows us all the steps performed during synchronization on *standard out*. With the `--delete` switch, we want to clear the destination from all the files which were deleted from the source file. The last flag `--progress` lets `rsync` display the progress of the process.

Our next task is to determine if the data has really arrived at our intended destination. We have done that by creating a variable in which we write a timestamp. After that we put this timestamp into a file, which we copy to our destination. Eventually we retrieve this file by using the command `curl`.

```
happy=$(curl -s https://informatik.hs-bremerhaven.de/docker-$USER-web/
    robbi/happy.txt)
echo "$happy"
echo "$currentdate"
if [ "$happy" = "$currentdate" ];
then
    echo happy
else
    echo unhappy!
    exit
fi
```

Listing 39: our Happybit

If the value from the `curl` statement does not match with the timestamp in our variable ”$currentdate”, the script stops at this point.

5.1.2 `deploy.sh`

This script is for deploying the data on docker form our group user account. As already mentioned, it uses the `sudo` command to trigger statements on the group account:

```
sudo -i -u infra-2022-e git -C /home/infra-2022-e/repos/infra-2022-e/ pull
sudo -i -u infra-2022-e sh /home/infra-2022-e/repos/infra-2022-e/ldeploy.
    sh
```

Listing 40: our deploy script

With the help of the `sudo` command we pretend to be logged in to our group account. There are only two statements in that script; the first one is to *pull* all the data from the main repository and the second one is to execute the standard ”ldeploy.sh”, which we’ve explained above. The advantage that we have here is that every group member has to *commit* their changes if they want to see the changes on the group account. Hence there is no deploy to our ”productive” server without a `git commit` and a `git push` command.

6 Conclusion

by Silas Rathgeber

As Table Features (Section 4.1) shows, we couldn’t actually implement all the features that we initially planned to implement because it would have made our project bigger than we intended it to be. Robbi currently reacts to only two scenarios, which are strictly dependent upon what the user gives as an input. For instance, the string ”`not right!`” has to be entered exactly like this, otherwise Robbi would not understand that his last answer was incorrect. In summary, the code for reacting to conversational scenarios is still a bit amateur and could be made much more sophisticated. As an example, Robbi could be taught a selection of words that express agreement or we can come up with a completely different way of evaluating the conversation. Another interesting topic would have been sending emails to activate a user account.

Among other complications, structuring the code and our documentation proved to be quite challenging. Making Robbi intelligent enough to have a meaningful conversation is a big task, which would eventually lead to artificial intelligence, in our opinion. To decide which technology should be implemented to solve a certain problem was also quite demanding.

Besides the technical aspect of our project, we would like to emphasize that we found the social aspect of group work to be challenging as well. This is not unexpected at all because coordination and task distribution often leads to discussions and arguments. It must be made clear, however, that these arguments/discussions in our team-work have been constructive in nature. Hence we were able to complete our project successfully in time and achieved what we had intended. In conclusion, we can all claim to have learned a lot from this module.

We would like to add that all the functions and technologies that we have implemented are in working condition.

Finally we would like to thank our fellow student Fabian Lignitz, who provided us with the template for this LaTeX document.

List of Figures

List of Tables

List of Code Listings

References

[22] July 2022. URL: http://www.gnuplot.info/.

[al22] pulkitagarwal03pulkit et al. *Advantages and Disadvantages of PHP*. June 2022. URL: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-php/.

[Beg] WP Beginner. *What is: Apache*. URL: https://www.wpbeginner.com/glossary/apache/.

[BW] Fred Lo Bryan Jones Philip Lim and Warren Wang. *AJAX - Technology Evaluation*. Presentation provided at SlidePlayer.com. URL: https://slideplayer.com/slide/4814173/.

[dAm22a] C. d'Amato. *MySQL Tutorial*. Aug. 2022. URL: http://www.di.uniba.it/~cdamato/corsi/BasiDiDati-Materiale/Intro%20a%20MySQL.pdf.

[dAm22b] C. d'Amato. *MySQL Website*. Aug. 2022. URL: https://www.mysql.com/.

[Das] Ankush Das. *11 Reasons Why Linux Is Better Than Windows - It's FOSS*. URL: https://itsfoss.com/linux-better-than-windows/.

[Fou] MariaDB Foundation. URL: https://mariadb.org/.

[Gee] Geeks4Geeks. *XHTML Introduction*. URL: https://www.geeksforgeeks.org/xhtml-introduction/.

[git22] git-scm.com. *Documentation*. Sept. 2022. URL: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell#:~:text=A%20branch%20in%20Git%20is,branch%20pointer%20moves%20forward%20automatically..

[Gon] Andrés González. *AJAX - Asynchronous JavaScript and XML*. URL: https://guiaviajesvirtual.com/aghsoftech/index-webdesign.php?recharge=ajax.

[Gro] The PHP Group. *phpinfo — Outputs information about PHP's configuration*. Online PHP Manual. URL: https://www.php.net/manual/en/function.phpinfo.php.

[Hop21] Computer Hope. *Documentation*. May 2021. URL: https://www.computerhope.com/unix/unohup.htm#:~:text=The%20name%20nohup%20stands%20for,the%20process%20to%20continue%20running..

[jav] javaTpoint. *Get and Post Methods in PHP*. URL: https://www.javatpoint.com/get-and-post-methods-in-php.

[Jee] JavaScript Jeep. *Everything About XMLHttpRequest in JavaScript*. The figure in this subsection was taken out of that article as well. URL: https://betterprogramming.pub/everything-about-xmlhttprequest-in-javascript-8adacc98a209.

[Kel21] Thomas Williams & Colin Kelley. *gnuplot 5.5 - An Interactive Plotting Program*. Version 5.5 Development Snapshot November 2021. Originally prepared by Dick Crawford, Version 5.5 organized by Ethan A. Merritt et al. Nov. 2021.

[Mar22] Louisa Marshall. *Introduction to JavaScript*. Sept. 2022. URL: https://slideplayer.com/.

[McK21] Cameron McKenzie. *Documentation*. Oct. 2021. URL: https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Git-Branch-Create-Example-Command-Checkout-Commit-Tag.

[MDN] MDN. *Synchronous and asynchronous requests*. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests.

[Mor22] Pat Morin. *Introduction to JavaScript*. Sept. 2022. URL: https://carleton.ca/scs/people/pat-morin/.

[NMA22] NMAP.org. *Documentation*. Aug. 2022. URL: https://nmap.org/ncat/guide/ncat-usage.html#ncat-listen.

[Rad] Prof. Dr.-Ing. Oliver Radfelder. *gnuplot*. URL: https://informatik.hs-bremerhaven.de/oradfelder/gnuplot.html.

[Rad22] Oliver Radfelder. *MySQL Syntax*. Aug. 2022. URL: https://informatik.hs-bremerhaven.de/oradfelder/mysql.html.

[red22a] redis.io. *Documentation*. Sept. 2022. URL: https://redis.io/docs/about/.

[red22b] redis.io. *Documentation*. Sept. 2022. URL: https://redis.io/docs/manual/cli/.

[Rob22] Alex Robbins. *Documentation*. Feb. 2022. URL: https://cloudzy.com/knowledge-base/netcat-listener/.

[Roc] Eileen Roche. *Explaining XML*. Out of the Harvard Business Review Magazine, issue of July-August 2000. URL: https://hbr.org/2000/07/explaining-xml.

[W3Sa] W3SCHOOLS. *Echo and Print Statement*. URL: https://www.w3schools.com/php/php_echo_print.asp.

[W3Sb] W3SCHOOLS. *HTML Versus XHTML*. URL: https://www.w3schools.com/html/html_xhtml.asp.

[W3Sc] W3SCHOOLS. *Introduction to XML*. URL: `https://www.w3schools.com/xml/xml_whatis.asp`.

[W3Sd] W3SCHOOLS. *PHP include and require*. URL: `https://www.w3schools.com/php/php_includes.asp`.

[W3Se] W3SCHOOLS. *PHP Introduction*. URL: `https://www.w3schools.com/php/php_intro.asp`.

[W3Sf] W3Schools. *PHP Data Types*. URL: `https://www.w3schools.com/php/php_datatypes.asp`.

[web22] webstepbook. *JavaScript, Event-driven Programming*. Sept. 2022. URL: `https://www.webstepbook.com/supplements-2ed/slides/ppt/13-IntroJavascript.pptx`.

Statutory Declaration

We, as a group, affirm that we have independently written the work submitted by us. All passages taken verbatim or in spirit from published or unpublished works of others, have been marked as such. All sources and aids that we have used for the work are indicated. Not the thesis itself nor substantial parts of it have yet been submitted to another examination authority.

Bremerhaven, 23.09.2022 *Signatures* :

[illegible]

Zahid

[illegible]

[illegible]