České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Lebedeva Antonina

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: Webová aplikace pro sběr a vyhodnocení dat z uživatelských testů

Pokyny pro vypracování:

Analyzuje možnosti pro vzdálené provádění uživatelských testů a způsoby statistického zpracování dat z takových uživatelských testů. Na základě analýzy navrhněte a implementuje webovou aplikací pro sběr a vyhodnocení dat z těchto testů. Aplikace bude obsahovat správcovské rozhraní umožňující uživatelům definovat nový test, typ testu (within subject, between subject) a data, která se budou v testu sbírat. Aplikace bude data sbírat přes webovou službu. K webové aplikaci implementuje knihovny pro jazyky Java a Javascript, které usnadní vývojářům uživatelských testů odesílání dat do webové služby. Dále implementujte ve webové aplikaci automatické statistické vyhodnocení dat (kontinuálních a binárních) a výsledky vizualizujte ve webové aplikaci ve formě intervalů konfidence. Výslednou webovou aplikace a webovou službu otestujte. Navrhněte uživatelský test s alespoň dvěma faktory, implementujte prototyp na kterém bude možné uživatelský test provést a test proveďte. Prototyp bude data z testu odesílat do realizované webové služby pomocí jedné z realizovaných knihoven pro odesílání dat. Data budou vyhodnocena a vizualizována pomocí realizované webové aplikace.

Seznam odborné literatury:

[1] Neo4j Graph Data Modeling (ISBN 978-1-78439-344-1)
[2] Practical API Design (ISBN 978-1-4302-0973-7)
[3] Mining and indexing graph databases (ISBN 978-1-303-78261-9)
[4] Graph Data Management: Techniques and Applications (ISBN 978- 1613500538)

Vedoucí: Ing. Ladislav Čmolík, Ph.D.

Platnost zadání do konce zimního semestru 2018/2019

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

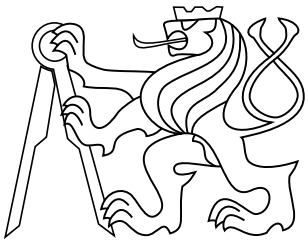
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze 27.09.2017

Master's thesis

Web application collecting and evaluating data from user experiments

Antonina Lebedeva



January 2018

Ing. Ladislav Čmolík, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Computer Science

Acknowledgement

Let me thank Ing. Ladislav Čmolík, Ph.D. for the professional guidance, for his assistance and advice during the work on this thesis. Also, I would like to thank my friends and parents for their support.

Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

Abstract

Hlavním účelem této práce je vytvoření platformy pro vývojáře a výzkumníky v oblasti uživatelských testů. Tato platforma bude umožňovat správu experimentů s uživateli, poskytovat přístup k nasbíraným datům a zobrazovat jejich přehledy. Mimo jiné bude nabízet statistické vyhodnocení dat a jejich vizualizaci. V analytické části budou řešeny hlavní podmínky a aspekty uživatelských experimentů a vyhodnoceny dvě metody: intervaly konfidence a ANOVA. V části návrhu jsou důkladně analyzovány doménové modely a sepsané funkční požadavky pro budoucí systém. V praktické části bude tento systém naimplementován a následně otestován s reálnými uživateli.

Klíčová slova

uživatelské testy, intervaly konfidence, ANOVA, web aplikace;

Abstract

The main purpose of this thesis is to create a platform for developers and researchers in the field of user testing. This platform will allow to manage their experiments with users, to have access to the collected data, and to see its overview, besides that, it will be a tool for the statistical analysis and the visualization of this data. Firstly, the research will be done to understand domain objects and structurize the given requirements. The analytical part will discuss major terms and aspects of user experiments and two evaluation methods: Confidence Interval and ANOVA. After that, the system will be designed and implemented, and finally, I will perform the testing with real participants.

Keywords

user experiments, research, Confidence interval, ANOVA, web application;

Contents

Abbreviations

AJAX	Asynchronous Javascript and XML
ANOVA	Analysis of Variance
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BSD	Between Subject Design
CI	Confidence Interval
CLI	Command Line Interface
CORBA	Common Object Request Broker Architecture
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
DCOM	Distributed Component Object Model
DF	Degrees of Freedom
DRY	Don't Repeat Yourself
ERB	Embedded Ruby
FTP	File Transfer Protocol
FSM	Final State Machine
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IQ	Intelligence Quotient
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MVC	Model View Controller
PHP	Personal Home Page
REST	Representational State Transfer
RPC	Remote Procedure Call
SD	Standard Deviation
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
UI	User Interface
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
WSD	Within Subject Design
WSDL	Web Services Description Language
XML	eXtensible Markup Language

1. Introduction

Nowadays testing takes more significant part in software development. It helps to ensure product quality and develop a new way of improving existing solutions to match user requirements.

An ordinary situation that arises in the software development involves choosing some solution from several options. These options could be UI elements, the color palette or different steps to accomplish a task. To solve this situation, user experiments are conducted to obtain information directly from end users. But raw data serves no purpose without careful analysis and evaluation.

For the sake of simplicity, the user and researcher are referred in a masculine form (he, him, his) throughout this master's thesis. Statistical data evaluating could provide valuable insight, also it approves or disapproves researcher expectations about his product.

1.1. Goals of the Thesis

The purpose of this work is to create a convenient service for researchers, through which they can easily collect and visualize the data obtained from these tests. The final product will serve as a basis for the management of experiments and collected data, which can easily be extended in the future.

1.2. Motivation

Currently, there are a lot of programs assisting in the usability testing. For instance, many of them record video/audio of the participant and the device where the software runs. Most support the functionality to take notes. Also, there are tools that could collect all the input information from the user: mouse movement, clicks, and even the eye tracking. Additionally, information about the satisfaction and impressions from users can be obtained in the form of responses to surveys. Numerous calculators for statistical processing methods are already created, but in the vast majority, they lack flexibility and convenience. Unfortunately, all these useful tools are available separately.

Therefore, the idea of a new system was born. It would combine a data collection, a statistical analysis, and customized visualizations, simultaneously providing means to persist and access the original data.

My personal motivation is to advance in the field of web application development using the latest technologies and, in doing so, to create a tool that will help people. Both of these together create an unmissable opportunity for me.

1.3. Structure of the Thesis

To achieve this goal, I need to follow certain steps. At the beginning, it is essential to carefully analyze methods of performing tests, possible variations in experiments, and types of variables, to classify the collected data, to consider existing assessment

methods and data visualization. After analysis the design of the future system emerges based on the conclusions from the analytical part.

Assignment of this thesis requires the presence of certain components of the system. The system as a whole will consist of 3 components.

For greater clarity, I name each component and will use these names further in the text. Component **"Web service"** serves as the main point where the researcher defines all the required experiment conditions, it is also the place of data collection from these experiments and their subsequent processing and visualization.**"Web service"** includes client and server parts.

The next component of the system is a **"Target application"** that will be tested. This part belongs to the researcher, all decisions regarding implementation remain with them. This component is the producer of data from experiments. These measured data subsequently will be stored and evaluated in the **"Web service"**. And accordingly, the researcher will have access to them through the client part of the **"Web service"**.

In order to connect **"Web service"** and **"Target application"**, it is necessary to create a mediator that is represented as a **"Transfer library"**. That library simplifies sending collected data to the **"Web service"** API. The library also serves to prepare consistent data in a predefined format. Therefore, the researcher must include the **"Transfer library"** in the code of its **"Target application"**.

The next step after the design is the implementation. In that part the structure of the project will be discussed in a greater detail as well as several technical aspects.

As a logical conclusion, the entire system will be tested using the prototype of a simple **"Target application"**.

2. Analysis

The following chapter presents theoretical fundamentals that are required to understand our approach to the given task - a platform for researchers and developers that will simplify the process of collecting and evaluating data from user tests. I would like to clarify that by "user test" I mean the user research of web or desktop applications.

It describes what *User Research* is and how those experiments are performed. Then it analyses existing experiment designs, their advantages, and disadvantages, and ways to evaluate and visualize data collected during testing. In order to subsequently based on these data, you could choose the most suitable method of evaluation and also take into account all the needs of the researcher and implement the most convenient service for the collection and analysis of user test data.

2.1. User research

User research is method that explores the behavior, motivation and needs of users in order to obtain suggestions to improve the product or service. According to Mike Kuniavsky [1], **user research** is the process of figuring out how people interpret and use products and services.

2.1.1. The Qualitative and Quantitative dimension

The user researches branches into **Qualitative** and **Quantitative** methods. The following **Figure 1** illustrates relations between "dimensions" and the types of questions.

Figure 1. Dimensions vs. Questions.

It is crucial to specify the research goal when choosing research method. Simply speaking, **quantitative** methods are much better suited for answering the question *"how much"*. **Qualitative** research answers the question *"why?"*. Currently, it is common to use a combination of both approaches. This is known as a mixed method.

The purpose of **quantitative** method is to obtain measurable data for the numerical motives, opinions and attitudes towards certain behavior. It is important to get enough data to achieve a statistically significant sample, and therefore we strive to work with as many participants as possible.

We can divide the measurable data into two types: **subjective** and **empirical** (quantitative).

- **Subjective data** is gathered by means of questionnaires, such as the post-test survey using a Likert scale, and audio record of participant's way of thoughts and actions. This data represent the self-reported participant subjective ratings for satisfaction, ease of use, ease of finding information and subjective evaluation of tested UI.
- **Empirical data** is objective data that can be counted and measured. During user tests we are able to count the number of the following occurrences:
  - **Successful Task Rate**
    Typically, participants will perform a set of planned tasks. The task will be considered complete if the participants report they have completed the task goal and the results correspond to the expected. It is also important to establish clear success criteria for each task and clarify where the participant should begin the task.
  - **Critical Errors Rate**
    Critical errors may happen due to unforeseen consequences, so that target of the scenario will become unreachable. Wrong data value could be one of those errors. Essentially the participant will fail to finish the task. The participant may not even realize that the task goal is incorrect or incomplete.
  - **Non-Critical Errors**
    Non-critical errors are errors that are recovered by the participant and do not result in the participant's ability to successfully complete the task. These errors result in the task being completed less efficiently.
  - **Time On Task**
    The amount of time it takes the participant to complete the task.

2.1.2. Types of User research methods

1. **Moderated in-person**
   This basic method is used to obtain feedback from live users interacting with everything from paper prototypes to fully implemented applications in a specially equipped laboratory.
   Usually, the laboratory consists of 2 rooms. In the testing room, there is a computer with a program being tested, a microphone and a camera. In this room the participant performs tasks, commenting on them aloud, and everything that happens is recorded including the computer screen and the participant's voice and facial expressions during testing. To guide the participant and his psychological comfort a moderator is co-located with the participant.
   The main disadvantage of this method that it requires a lot of time and finances. Typically, the length of the session is from one hour to 90 minutes and breaks between sessions up to 30 minutes. On this basis, the researcher will have time to test from 3 to 6 participants for 1 day. In this case, we work with a **small sample size**. Therefore, this method is often combined with others.

One of the advantage is the ability to decide if participant's poor performance was an aberration (sometimes called an "outlier") or if other end users might perform the task similarly [2]. And in the case with "outliers" do not take into account these results.

2. **Moderated remote**
   Participants do not have to travel to the labs, they can deliver results from all around the world. The disadvantage of that approach is poor ability to control the environment and other conditions. Screen sharing software allows the moderator to remotely watch the participant attempts to perform tasks with software or a website and allows to reveal problems.
3. **Unmoderated remote**
   Just like in the previous case, we can test the UI with participants all around the world. The researcher can collect a lot of data quickly and for a fraction of the cost of in-person testing. In many cases, a recording of the participant's screen and webcam is available, but there is no way to simultaneously interact with all participants and there is no way to identify participants who had misunderstood the task or those who had external problems that affected the results and total time.

2.1.3. Summary

The resulting application will focus on collecting and evaluating data that can be calculated and estimated, i.e. empirical data (for example, *task time, error rates, rating scales*). Unfortunately, collecting empirical metrics require a larger sample size in order to get tighter confidence intervals (See Section 2.3.1 for details). Consequently, they are usually conducted using the unmoderated remote mode.

From the previous discussion about user research methods, (See Section 2.1.2 for details) it is clear that it is not always possible to get rid of data that deviates from the average. Therefore, it will be always necessary to remember about these cases and carefully transform the collected values. For instance, *log-transformation* is used to properly evaluate task time data.

A more detailed description of the existing methods of data transformation and evaluation is provided in the Section 2.3.

2.2. Experimental designs

2.2.1. Variables

At the beginning, it is good to clarify what a variable is and illustrate it on examples. A **variable** is an object, event, idea, feeling, time period, or any other type of category you are trying to measure. Let us focus on two types of variables:

- **Independent variable** - experimental effect/factor which is actively controlled and manipulated by an experimenter. The *independent variable* causes some changes in *dependent variables*.
- **Dependent variable** - variable being affected by the *independent variable*.

One the one hand, when testing an interface, the *independent variable* may be the age of participants, their experience, as well as test conditions, and more specifically, the method of interaction or variant of the interface. On the other hand, the measured data from the experiment (the speed of the job, the number of errors, etc.) is the *dependent variable*.

Levels of an Independent Variable

In general, the number of levels of an independent variable is the number of experimental conditions. If an experiment were comparing 5 types of UI (buttons, menu etc.), then the independent variable (a type of UI) would have 5 levels.

Let us take a look at an example. There is a product to which we have added few new features and now we want to check whether it has improved the usability of our product. Tests with users showed that a new version of the product is really better compared to the previous one. Only here the problem is that we do not know exactly *why* we got such results, *what exactly influenced* these improvements? And in order to investigate *the cause* of the improvement, it is necessary to conduct tests with each new feature (level) separately and also with *all combinations* of these features.

It should be mentioned, that in practice many levels are often avoided because with the addition of the next level the number of tests will increase **exponentially**.

For instance, 3 new features will require 8 tests: *1 (test with old version) + 3 (each feature separately) + 3 (combinations of 2) + 1 (combinations of 3)*. Consequently, it is advisable to choose only 1 or 2 levels of the testing factor and try to avoid the combination as far as it's possible.

2.2.2. Between-subject design

In a between-subjects design, the various experimental treatments such as a part of the website or a version of the product are tested by different groups of subjects (users). This allows the subsequent comparison of the levels to be based on the comparison of independent groups of subjects. The number of user groups (subjects) depends on the level of the independent variable. For example, if we have 3 levels, then we need 3 disjunctive groups of participants. In case the tasks are longer and there is a probability that the participants may become tired, this design type is a preferable choice [2].

The following **Table 1** illustrates between-subjects design. The table shows the distribution of participants needed to test 3 different testing criteria (A, B, C). These criteria may mean **tasks** or it may be **versions of the product**, generaly speaking, it is called *level of the factor*. The designation **P#1** denotes the participant with its serial number. From the table, it is clear that if we want to test 3 tasks (or product versions) in groups of 3 people, then we will need 9 participants.

It is important to note that each participant tests only once.

A	B	C
P#1	P#4	P#7
P#2	P#5	P#8
P#3	P#6	P#9

Table 1. Distribution of participants in between-subject design.

Advantages

- **Easy extensibility**
  If after some time the experimenter wants to test another condition, it will be suf-

ficient to conduct an experiment with another equivalent group and then compare the results with previous groups.

- **No learning (carryover) effects**
  When Task A helps to accomplish Task B it is called a learning effect. We want to avoid it because usability problems associated with Task B may be missing. This design softens the learning effects caused by the execution of one set of tasks, before performing other similar tasks [2].

Disadvantages

- **Large sample size**
  The main disadvantage is that design requires a large number of participants to create any useful and analyzed data. Since each participant is measured only once, researchers need to add a new subject for each level.
- **Individual differences**
  Problem is that it is impossible to maintain homogeneity across the groups and this can skew data. To solve this problem, the researcher may use the random distribution of participants into groups or previously prepare equivalent groups.

2.2.3. Within-subject design

A within-subjects design differs from a between-subjects design in such a way that the same subjects perform tests at all levels of the independent variable and then their performance for each experiment is compared. In other words, the within-subject design uses the same participants with every condition of the research, including the control group. The control group does not receive any new conditions or receives standard conditions that can be understood as baseline and it is used to compare groups and assess the effect of researching subject.

The **Table 2** also, shows the distribution of participants for testing 3 tasks or designs. Unlike the previous design, only one group is required.

A	B	C
P#1	P#1	P#1
P#2	P#2	P#2
P#3	P#3	P#3

Table 2. Distribution of participants in within-subject design.

Advantages

- **Small sample size**
  The main benefit is that it requires a fewer number of participants. It also leads to the use of fewer resources.
- **Individual differences are controlled**
  Within-subject design can reduce the probability of errors associated with individual differences between participants. In the context of between-subject design, participants are randomized in the experiment, so that significant differences between the groups can be observed. With within-subject design, participants are

in the same conditions, so the results will be weaker depending on the individual differences between them.

Disadvantages

- **Order effects**
  There are several reasons for the occurrence of order effects. The most popular cause is the *practice*. This means that the participant warms up at the first task and performs it longer than the subsequent ones, which feel more familiar to him [3].
  Another reason is simple - *fatigue*. This is demonstrated by a decrease in the participant's performance by the end of the experiment.
  Also, any other *environmental condition* can affect the performance of the participant.
  The researcher can reduce the order effects by alternating the order of tasks with the condition that each possible sequence must be represented in approximately equal numbers of subjects. This is called **counterbalancing**.
  Its essence lies in the fact that the order of treatment varies from participant to participant.
  The simplest example contains only two possible conditions: **A** and **B**. Then the researchers may want to test all the sequences of those conditions. So that one group is tested with condition **A**, followed by condition **B**, and the other is tested with condition **B** followed by condition **A**. What if we have 3 conditions? The process is exactly the same, so for all sequences the researcher will need 6 groups. For 4 conditions - 24 groups, for 5 - 120 and so on. As you can see, counterbalancing becomes rather complicated with each new condition (level of an independent variable). To be more precise, the number of groups is always equal to the **factorial** of the number of conditions.
  Fortunately there are incomplete versions of that technique, the so-called compromise, designed to balance the strengths of counterbalancing with financial and practical reality. One such incomplete counterbalancing is the *Latin Square method*. Latin square is an n x n array filled with n different symbols, each occurring exactly once in each row and exactly once in each column.

Group 1	A	→	B	→	C	→	D	→	E
Group 2	B	→	C	→	D	→	E	→	A
Group 3	C	→	D	→	E	→	A	→	B
Group 4	D	→	E	→	A	→	B	→	C
Group 5	E	→	A	→	B	→	C	→	D

Table 3. Example of Latin Square for 5 conditions.

  The **Table 3** illustrates the Latin Square for 5 conditions. And as you can see, each sequence (**A** → **B**, **B** → **C**, **C** → **D** etc.) repeats exactly 4 times in this case.
- **Carryover effects**
  Carryover effect or learning effect was described in the previous design. It can lead to results distortion because the certain order of tested conditions can actually affect the behavior of the participany or cause a false response.
  The Carryover effect is also solved using **counterbalancing**. *Latin square* method copes well with the order effect, however, there is still weakness in the carryover

effect.

The preceding paragraph describes the Latin square method, and if we look at the **Table 3**, we can notice that **A** always precedes **B**, and this means that anything in condition **A** that potentially affects **B** will affect all but one of the sequences. The same situation with **A** and **E**.

To prevent this effect, it is recommended to use a *balanced Latin Square method*, which guarantees a reduction of carryover effect risks. The approach for an even number of conditions is slightly different from the odd ones. For experiments with an even number of conditions, the first row of the Latin Square will follow the formula $1, 2, n, 3, n-1, 4, n-2...$, where n is the number of conditions. For subsequent rows, you add 1 to the previous value. If the addition of 1 leads to a number greater than the number of conditions, then this number will turn into 1. It would be better to consider this with an example.

Group A	1	→	2	→	6	→	3	→	5	→	4
Group B	2	→	3	→	1	→	4	→	6	→	5
Group C	3	→	4	→	2	→	5	→	1	→	6
Group D	4	→	5	→	3	→	6	→	2	→	1
Group E	5	→	6	→	4	→	1	→	3	→	2
Group F	6	→	1	→	5	→	2	→	4	→	3

Table 4. Example of balanced Latin Square for 6 conditions.

In this example [**Table 4**], it is clear that **2** follows **1** just once, as well as pairs **2** → **3**, **3** → **4** and so on. This allows the researcher to exclude any sequence that may affect the carryover effect.

In the case of an odd number of conditions, two tables are required, the first is composed using the previous algorithm, and the second is its mirror image.

1	2	5	3	4
2	3	1	4	5
3	4	2	5	1
4	5	3	1	2
5	1	4	2	3
4	3	5	2	1
5	4	1	3	2
1	5	2	4	3
2	1	3	5	4
3	2	4	1	5

Table 5. Example of balanced Latin Square for 5 conditions.

With this design, every single condition follows another two times, and statistical tests allow researchers to analyse the data [4].

- **More difficult extensibility**
  Suppose that an experimenter will want to test a new version of the product in a year. Then, in the case of within-subject design, he will need to either find a new group of participants and test previous versions of the product, including a new one or to find an old test group and test only the new version.

2.3. Evaluating methods

Futher, two evaluation methods: the confidence intervals and ANOVA will be analyzed. Both methods are used to test statistical hypotheses. That is, the answer to the question: is there a significant difference between two (or more) approaches to the solution of some problem.

ANOVA works only with data which has normal distribution and returns just one value that indicates that one group is not equal to the other group [5]. It does not provide more detailed information.

However, it is always appropriate to add the visibility in addition to the available calculations; for this purpose, the confidence intervals are more suitable. They are easily visualized using a *box-and-whiskers* or *error bars* plots.

2.3.1. Confidence intervals

Introduction

Estimating the results of experiments, we only have data from a certain sample, but we want to generalize them to the whole population (specific target audience), to which unfortunately we do not have access. Also, we need to know how accurate our values are because the sample size directly affects the accuracy of the results. For this, we will find a range of values in which an unknown population parameter will be with a specified probability. These ranges are named **confidence intervals**.

The confidence interval can be regarded as an indicator of the measurement accuracy. Also, the interval is a measure of the stability of measurements, that is, if you repeat the measurements (experiment), an indication of how close the obtained value will be in comparison with the original value.

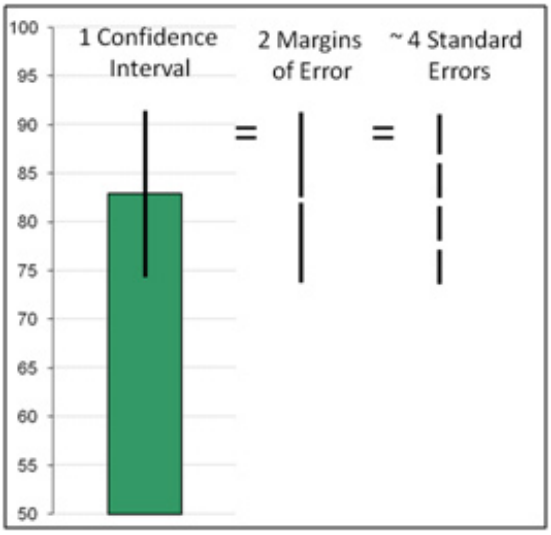


Figure 2. Confidence interval components. Source:[1]

The confidence interval is equal to 2 *Margins of errors*, and a margin of error is equal to about 2 *Standard errors* (for 95% confidence) [6]. This is shown in **Figure 2**. Formulas and a detailed explanation of all these terms will be presented later.

Let us look at the properties that can affect the width (accuracy) of the confidence intervals.

1. **Confidence level**
   The confidence level indicates the probability that an unknown population value will be covered by a confidence interval. The typically used values are 90% 95% and 99%. That is, if we perform the test (measurement) on the same user population 100 times, it was only 90, 95, 99 (depending on the confidence level) times, when the mean value will be within the given confidence interval. The higher the confidence level, the wider the interval we get.
2. **Variability**
   This is the difference in the values of a characteristic for different units for the whole population. High variability generate wider confidence intervals. In order to measure variability, the standard deviation is used.
3. **Sample Size**
   The sample size is one of the few things on which the researcher can affect. There is an inverse square root relationship between confidence intervals and sample sizes. If you want to cut your margin of error in half, you need to approximately quadruple your sample size. As the sample size decreases, the confidence intervals get wider.

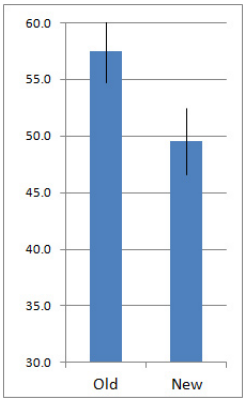
Interpretation of confidence intervals

For rapid verification of statistical significance, the overlap of confidence intervals is used. Statistical significance is used to determine whether the null hypothesis should be rejected or retained.

The **null hypothesis** is a hypothesis that is tested for consistency with available sample (empirical) data. Often, as a null hypothesis, there are hypotheses about the absence of relationships or correlation between the investigated variables in two or more samples. If the null hypothesis is rejected, then an **alternative hypothesis** is confirmed, which is a logical negation of the null hypothesis. The null hypothesis is rejected if the p-value is less than a predetermined level (alpha α). *Alpha* is called the significance level, and is the probability of rejecting the null hypothesis given that it is true. It is usually set at or below 5%.

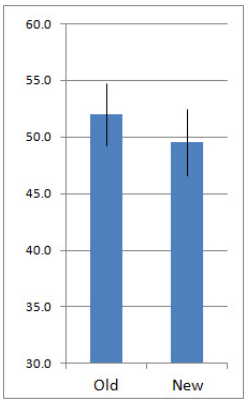
Next we will look at 3 possible options for the location of the confidence intervals.

1. **No overlap**



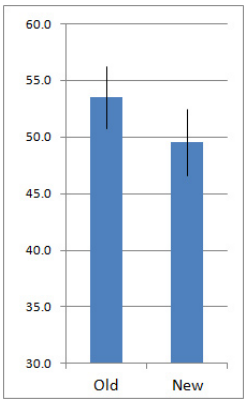
This variant is considered to be the best possible case scenario, because if the intervals do not overlap then you can be at least 95% confident that difference is *statistically significant*.

2. **Large overlap**



If there is a large overlap, then the difference is *not significant* (at the p <.05 level).

3. **Some overlap**



In case of a small overlap, it can not be said for sure that the difference is statistically significant. It is also wrong to draw the opposite conclusion. Usually, this indicates problems with the configuration of the test and with the received data. It's recomended to repeat the user experiment or it is also possible to calculate the confidence intervals for the difference and make a comparison.

Discrete and Continuous Values

To calculate the confidence interval, the first thing we need is to determine the type of data: continuous or discrete.

- **Continuous**
  Continuous data are metrics like *rating scales*, *task-time*, *revenue*, *weight, height* or *temperature*. For instance, time could be 23.45 seconds. We consider the *rating scales* (Likert scale as well) as continuous values.
- **Discrete**
  Variables such as *number of people* are called discrete variables since the possible scores are discrete points on the scale (could be 6 people, but not 4.53 people). *Completion rate* (or *Successful Task rate*) and *Error rate* are discrete values as well.

Then to compute a confidence interval, we need to know:

- **The mean** (for continuous data) or **proportion** (for discrete data)
  The mean describes the middle or most typical value. To get this average value, you simply divide the sum of all values by their number.

$$\overline{x} = \frac{\sum x}{n} \quad \text{or} \quad \overline{p} = \frac{p}{n} \tag{1}$$

where

$\overline{x}$ - the mean
x - data values
n - the sample size
$\overline{p}$ - the proportion
p - the number of participants who successfully completed the task (or respond with "yes" or another positive answer)

- **The sample standard deviation**
  The standard deviation describes how far each value is from the mean. You can think of the standard deviation as the average difference between each value and the mean.

$$SD = \sqrt{\frac{\sum(x - \overline{x})^2}{n - 1}} \tag{2}$$

where

SD - the standard deviation
x - data values
$\overline{x}$ - the mean
n - the sample size

- **Confidence level**
  You can choose from the most common options: 90%, 95% or 99%. But it is important to bear in mind that the confidence level can affect the width of the interval.

CI for Continuous Values

The best approach for constructing a confidence interval for continuous data is to use the t-distribution. We could look at *rating scales* as continuous values, in case of using scales from 0 to 100 points.

The t-distribution is like a normal distribution (also called a z-distribution) except that it takes the sample size into account. The t-distribution will provide the best interval regardless of the sample size.

The formula for the t-confidence interval takes the following form:

$$\overline{x} \pm \underbrace{\left(\underbrace{t_{(1-\frac{\alpha}{2})}}_{\text{Critical value}} \times \underbrace{\frac{SD}{\sqrt{n}}}_{\text{Standard error}} \right)}_{\text{Margin of error}} \tag{3}$$

where

$\overline{x}$ - the mean
$t_{(1-\frac{\alpha}{2})}$ - t-critical value from the t-distribution for $n - 1$ degrees of freedom and the specified level of confidence
SD - the standard deviation
n - the sample size

To find the t-critical value, we need to know α (alpha) and the *degrees of freedom*. *Alpha* is the *level of significance* used in the study, typically 0.05. It is also 1 - *confidence level*, which is typically 95% (1 - 0.95 = 0.05). The degrees of freedom (df) equals to $n - 1$.

An abbreviated t-table [**Appendix A**] is used to find the critical value. We should find column *df* and move to the right in the table until we reach our desired *significance level*.

CI for Discrete Values

Use the **adjusted-Wald** binomial confidence interval for *completion/error rates*. The adjusted-Wald interval (also called the modified Wald interval), provides the best coverage for the specified interval when samples are less than 150.

$$\hat{p}_{adj} \pm z_{(1-\frac{\alpha}{2})} \times \sqrt{\frac{\hat{p}_{adj}(1-\hat{p}_{adj})}{n_{adj}}} \tag{4}$$

where

$\hat{p}_{adj}$ - the adjusted proportion,

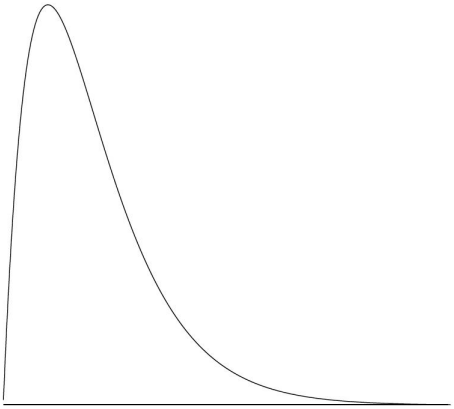
$$\hat{p}_{adj} = \frac{x + \frac{z^2}{2}}{n + z^2} = \frac{x + \frac{1.96^2}{2}}{n + 1.96^2} \approx \frac{x+2}{n+4}$$

x - the number of participants who successfully completed the assignment
n - the sample size
$z_{(1-\frac{\alpha}{2})}$ or z - the critical value of the normal distribution (1.96 for 95%)
n_{adj} - the adjusted sample size $n + z^2$

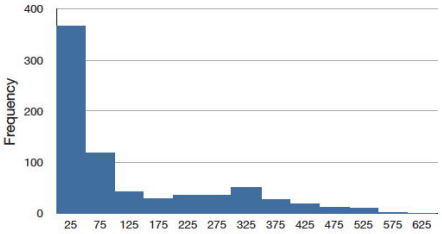
CI for Task Time

Task-time data can be divided into 2 main cases, which in their own way affect the calculation of confidence intervals. Each case will be considered separately below.

- **Positively skewed data**
  Task time data has a tendency to be positively skewed (see **Figure 3**). Due to the fact that the task-time data is positively skewed it should be log-transformed prior to using the t-confidence interval (see Section 2.3.1 for details). For confidence intervals using task times you should perform a **log transformation** on the raw values, and then compute the t-interval method, this means calculating the average mean and standard deviation of the logarithmic times. Finally, we get a logarithmic confidence interval so we need to transform it back by exponentiating the values. This method corrects the skew in task time data to generate accurate intervals.
  The **log transformation**, a widely used method to address skewed data, is one of the most popular transformations used in biomedical and psychosocial research. For transformation simply take a logarithm of every task-time value $y = \ln(x)$.
  Another popular use of the log transformation is reducing the variability of data, especially in data sets that include "outlyin" observations.



a) A continuous distribution with a positive skew.



b) A distribution with a very large positive skew.

Figure 3. Examples of positive skew. Source: [7]

- **Large Sample size (> 25)**
  For large sample task-time data the median is the best point estimate of the middle task time, so it is recommended to compute a confidence interval around the median using the binomial distribution method.

$$np \pm z_{(1-\frac{\alpha}{2})} \times \sqrt{np(1-p)} \tag{5}$$

  where

  n - the sample size
  p - the propotion (in our case the median = 0.5)
  $z_{(1-\frac{\alpha}{2})}$ - the critical value from the normal distribution (1.96 for 95% confedence level)

  The results of the equation are rounded up to the next integer and the boundary of the confidence interval is between the two values in the ordered data set.

CI for different designs

To make a better comparison of data from experiments and to determine if there is a *significant difference* between the mean values, the *confidence interval around the difference* can be calculated.

This interval not only confirms (or rejects) the presence of a significant difference, but also shows the size of this difference (the effect size). It should be noticed that different approaches are used for different designs (BSD and WSD) [6]. If we are talking about Within-subject design (i.e., the same group of participants goes through all the compared methods), then calculations are made for the difference of values from each participant. In case of a Between-subject design (i.e., different groups for each compared method), the difference of confidence intervals for each independent group is calculated.

We also divide the data into *continuous* (such as questionnaire data or task times) and *discrete* (such as a task completion rate or conversion rate).

Formulas of confidence interval around the Difference

- **Between-subject design**
  - Continuous data

$$(\overline{x}_1 - \overline{x}_2) \pm t_a \times \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \tag{6}$$

where

$\overline{x}_1$ and $\overline{x}_2$ - the means from group 1 and 2
s_1 and s_2 - the standard deviations from groupes
n_1 and n_2 - the sample sizes of groupes
t_a - the critical value from the *t*-distribution for *n-1* degrees of freedom

  - Discrete data

$$(\hat{p}_{adj1} - \hat{p}_{adj2}) \pm z_\alpha \sqrt{\frac{\hat{p}_{adj1}(1 - \hat{p}_{adj1})}{n_{adj1}} + \frac{\hat{p}_{adj2}(1 - \hat{p}_{adj2})}{n_{adj2}}} \tag{7}$$

where

$\hat{p}_{adj1}$ and $\hat{p}_{adj2}$ - the adjusted proportion for sample 1 and 2
n_{adj1} and n_{adj2} - the adjusted sample sizes of samples
z_α - the critical value from the normal or *z*-distribution

- **For within-subjects**

  - Continuous data

$$\overline{D} \pm t_a \times \frac{SD_D}{\sqrt{n}} \tag{8}$$

where

$\overline{D}$ - the mean of the results difference
SD_D - the standard deviation of the difference
n - the sample size
t_a - the critical value from the *t*-distribution for *n-1* degrees of freedom

  - Discrete data

$$(\hat{p}_{2adj} - \hat{p}_{1adj}) \pm z_\alpha \sqrt{\frac{(\hat{p}_{12adj} + \hat{p}_{21adj}) - (\hat{p}_{21adj} - \hat{p}_{12adj})^2}{N_{adj}}} \tag{9}$$

where

$\hat{p}_{1adj}$ - $\frac{m_{adj}}{N_{adj}}$
$\hat{p}_{2adj}$ - $\frac{r_{adj}}{N_{adj}}$
$\hat{p}_{12adj}$ - $\frac{b_{adj}}{N_{adj}}$
$\hat{p}_{21adj}$ - $\frac{c_{adj}}{N_{adj}}$
z_a - the z critical value for the level of confidence (95%)

The variables m, r, b and c are explained in the **Table 6**.

	Desing B pass	**Design B fail**	**Total**
Design A pass	a_{adj}	b_{adj}	m_{adj}
Design A fail	c_{adj}	d_{adj}	n_{adj}
Total	r_{adj}	s_{adj}	N_{adj}

Table 6. Results of experiment.

Recapitulation

In this subsection I would like to repeat the most important points from the previous parts.

- The design of the experiment can be Between-subject and Within-subject (see Section 2.2.2 and 2.2.3);
- The data is divided into continuous, discrete and the other (string);
- When calculating the confidence interval for continuous data, the *mean* is used and for discrete data *proportion* is used (see equation (1));
- Task time is continuous data and typically is positively skewed, so it is recommended to apply a log transformation to data and only then to calculate the confidence intervals using the equation (3);
- In case if the sample size of the task time is more than 25, it is recommended to calculate the CI around the median using the equation (5);
- For discrete data is used the Adjusted-Wald confidence interval (see equation (4));
- The confidence intervals are easily visualized on boxplots;
- The confidence interval around the difference helps us to distinguish between trivial differences and significant differences that users would probably notice;
- Also, the CI around the difference is used in order to calculate the *effect size*;
- In case of experiment with the Between-subject design CI around the Difference is generated using values (mean, standard deviation) for each independent group (see equation (6) for continuous data and equation (7) for discrete);
- In case of experiment with the Within-subject design CI around the Difference is calculated using values of difference (see equation (8) for continuous data and equation (9) for discrete);

2.3.2. Analysis of Variance

There is another method for studying the dependencies in data from experiments, or more specifically, for analyzing the statistical significance of the differences between means of independent groups. This method is called *Analysis of Variance* or can also be found the abbreviation *ANOVA*. It determines if an *independent variable* (the test conditions or factors) has a significant effect on a *dependent variable* (the measured data) [8]. ANOVA has several types of settings depending on the test conditions (number of independent variables, design of the experiment):

- **One-way**
  Used in tests with only one independent variable (factor) and several levels of this factor.

- **Two-way**
  There are explore two independent variables in the experiment. Each of the factors has 2 or more levels.
- **Repeated Measures ANOVA**
  Essentialy, it's one-way ANOVA, but for experiments with within-subject design.

We will focus on two types: one-way ANOVA and repeated measures ANOVA.

Specifically, ANOVA is used for null hypothesis testing. The null hypothesis, in this case, has approximately the following formulation:

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \cdots = \mu_k$$

Which is equivalent to the following assertion: *"There is no difference between means of independent groups"*. Then we use ANOVA to decide whether to accept the null hypothesis or reject it, thereby accepting an alternative hypothesis.

The ANOVA produces an p**-value** and F**-statistic**, on the basis of which the rejection or acceptance of the hypotheses is resolved. Before the experiment, the null hypothesis is first determined and the threshold value, **the significance level**, is selected. The most frequently used value is 5% (0.05) and denoted as α. This value could be considered as the inverse of the confidence level, so if significance level is 10% (0.1), the corresponding confidence level is 90%. The significance level represents the probability of an error. It means, there is always a 5% (depending on the significance level) risk of erroneous interpretation of the results.

Once we get the value we can make a decision about the null hypothesis in two possible scenarios:

- The p-value is **less or equal** than selected significance level
  Then the hypothesis test is statistically significant. The null hypothesis is **rejected**.
- The p-value is **more** than selected significance level
  Then the null hypothesis is **accepted**. There are no significant differences.

It is very important to keep in mind that the results of interpreting the confidence intervals as well as the results of the p-value comparison should agree. For instance, if the difference is significant, then the mean of the null hypothesis (specifically, the mean of one independent group) is contained in a particular confidence interval. And accordingly, the p-value will be less than the significance level, but only in case that the confidence level and the significance level correspond.

One-way ANOVA

In order to obtain more accurate results, it is highly recommended to comply with all or most of the following data requirements for one-way ANOVA test:

1. The **dependent variable** (or the collected data) should be **continuous**. It may be the execution time, the IQ level, weight and so on.
2. **Participants** of the experiment should be in different **independent groups**. They can be separated by personal characteristics (age, profession, practice) or by conditions of the test (factors). It corresponds to the previously described Between-subject design (subsection 2.2.2). If the data does not satisfy this criterion, then it's needed to perform another statistical test, for example, *Repeated measures ANOVA*.

3. Also, gathered data should **not** contain **significant outliers**. The term "outlier" was mentioned earlier. Their presence in the data may reduce the reliability of results. It is extremely important to determine possibly very different values.
4. ANOVA only expects input data (the *dependent variable*) to be approximately normally distributed. This applies to each category (group) of the independent variable. This criterion is less strict, unlike the previous ones. Even if it's slightly violated, it is still possible to receive valid results.

Below is a **Table 7** containing all the formulas and concepts needed to calculate one-way ANOVA test.

	SS	**DF**	**MS**	**F**
Between	$\sum_g^G n_g \times (\overline{x}_g - \overline{x}_G)^2$	$k-1$	$\frac{SS_B}{k-1}$	$\frac{MS_B}{MS_W}$
Within	$\sum_g^G n_g \times (x - \overline{x}_g)^2$	$N-k$	$\frac{SS_W}{k-1}$	-
Total	$SS_B + SS_W$	$N-1$	-	-

Table 7. One-way ANOVA calculations.

where

SS_B - the Sum of Squares Between groups
SS_W - the Sum of Squares Within groups
MS - the Mean Square
DF - the degrees of freedom
F - the F-statistic
N - the total sample size, total number of participants
k - the number of independent groups
n_g - the number of participants in each group, $g \in 1, 2..k$
x - the individual value in group g, $g \in 1, 2..k$
$\overline{x}_g$ - the mean of the each group, $g \in 1, 2..k$
$\overline{x}_G$ - the mean of all values

After the *F-statistic* calculation, we can figure out the p-value. In order to do this, we need to use the f-distribution table that can be found in **Appendix B**. In that table, the rows represent **denominator** degrees of freedom (DF_{Within}) and the columns represent **numerator** degrees of freedom ($DF_{Between}$). Also, the table values depend on the level of significance (alpha). Example of notation the critical F-ratio at the significance level (alpha) = 0.05, with 3 independent groups and 12 total participant's number is $F(0.05, 2, 9) = 4.2565$.

To calculate the p-value, there is large number of calculators and functions which just need to specify several parameters: the significance level, the obtained F-statistic value, the DF numerator and DF denominator. It is also possible to make a decision based on comparison of the F-statistic values rather than p-values.

Then we simply compare the critical F-ration and calculated F-statistic. If F-statistic is equal to or larger than this critical F-value, then there is the significant difference at that level of probability (alpha), therefore, $p < \alpha$.

Repeated measures ANOVA

The repeated measures ANOVA is almost identical to the one-way ANOVA, except one additional calculation must be performed to account for shared variability. The shared variability arises as a result of within-subject design when the same participant takes part in several testing groups.

The following **Table 8** is very similar to the one-way ANOVA calculations table. Many of the values described in the previous table are also found in this table, so the description will be provided only for the new ones.

	SS	**DF**	**MS**	**F**
Between	$\frac{\sum(\sum x_i)^2}{s} - \frac{T^2}{N}$	$k-1$	$\frac{SS_B}{k-1}$	$\frac{MS_B}{MS_{Error}}$
Within	$\sum x^2 - \frac{\sum(\sum x)^2}{s}$	$N-k$	-	-
Subjects	$\frac{\sum(R_j)^2}{k} - \frac{T^2}{N}$	$s-1$	-	-
Error	$SS_W - SS_{Subjects}$	$DF_W - DF_{subjects}$	$\frac{SS_{Error}}{DF_{Error}}$	-
Total	$SS_B + SS_W$	$N-1$	-	-

Table 8. Repeated measures ANOVA calculations.

where

s - the number of participants in group (the same number in each group is assumed)
T - the sum of all values, $\sum x_i$, $i \in 1, 2, ..N$
x - the individual value in group g, $g \in 1, 2, ..k$
R_j - the sum of values for each subject, $= \sum r_i$, $i \in 1, 2, ..k$, $j \in 1, 2, ..s$

The principle remains the same: calculate the F-statistic, then compare it with the critical value from the f-distribution table [**Appendix B**] and make a conclusion, accept or disprove the null hypothesis. In this case the value of $DF_{Between}$ is used as the **denominator** for degrees of freedom and the DF_{Error} value - as the **numerator** for degrees of freedom.

2.4. Summary

In this chapter, two basic statistical methods for evaluating the collected data from user tests were analyzed. One of the methods involves confidence intervals, the other

is the ANOVA test. In essence, these two methods are equivalent to the problem of determining a significant difference.

The confidence interval produces an information about a range of values that can be graphically represented. This range can contain the mean value of the null hypothesis with a certain degree of probability (the confidence level) defined in advance. This information allows us to draw conclusions about the statistical significance.

P-values, however, do not provide such important information as the size of the difference or direction of the effect. Confidence intervals are able to cope better with this problem, since they contain more useful information. Another key property of the confidence interval is that it takes into account the information at the measurement stage. For example, an interval contains values in the original data units without any transformations. In addition, there are several factors affecting the accuracy (width) of confidence intervals (such as sample size, standard deviation in groups, etc.) Thus, the representation of the confidence interval tells us a lot, more than just the statistical significance. In contrast to *p*-values, confidence intervals denote the direction of the impact being investigated. All the work with confidence intervals is to control whether they include a certain value (typically, the mean of the compared group) or not. Another detail to consider is that there is a possibility that the result may be significant with a larger sample.

In conclusion, it should be emphasized that *p*-values and confidence intervals are not contradictory statistical concepts. Confidence intervals can be calculated from *p*-values and vice versa. These two concepts complement each other.

Summing up the foregoing, we come to the conclusion that with all the advantages and disadvantages, the best approach to evaluate and visualize results of user experiments involves using confidence intervals.

3. Design

This section describes the concept of the future system, its main parts, the set of functions, the methods of communication between components, and discusses the architecture of the web service.

3.1. System overview

In the introduction of this thesis, the structure of the future system was mentioned.

Let us assume a hypothetical researcher or a developer that wants to compare the efficiency of solutions of his software product. It might be an arrangement of UI elements, different shapes or colors, or any other solution which can affect the performance efficiency of tasks performed by participants. This software product is essentially a **"Target application"**.

So if the researcher decides to conduct a research with the real users (participants), he can use the **"Web service"** in order to make a comparison or to simply keep a record of the experiments and their data. On this site the researcher will register and create a new experiment where he can describe all the necessary data, for example, the variables that he is going to study. If we talk about efficiency, it is logical to assume that the researcher will be interested in the *time* spent on the task and the number of *errors* made while performing it.

But this is not enough for a functioning system, for its completion there is a lack of a bridge between participants and the **"Web service"**. A **"Transfer library"** will solve this issue when the researcher includes it in the code of his product. This library knows how, in which form, and where to send data samples from the participants of the experiment. After the researcher sets up both points for successful communication with each other, he can conduct user tests and then access this data through the **"Web service"** interface.

Figure 4 illustrates all the components and their interconnections. Please note that this is just an abstract view of the system, many components are simplified for better perception.

Web service

The **"Web service"** component includes both client and server parts. And all the business logic as well as access to the database is performed on the server side. In the Figure 4, I semantically divided the controllers into an HTML controller and an API controller. The HTML controller receives requests from the browser and as a response sends a corresponding page containing data from the database, if any is available. The API controller is primarily designed to communicate with the **"Transfer library"**, that is, for receiving and further saving the collected data into the database. So that after the experiments, the researcher has the opportunity to work with the collected data through the client part of the web service.

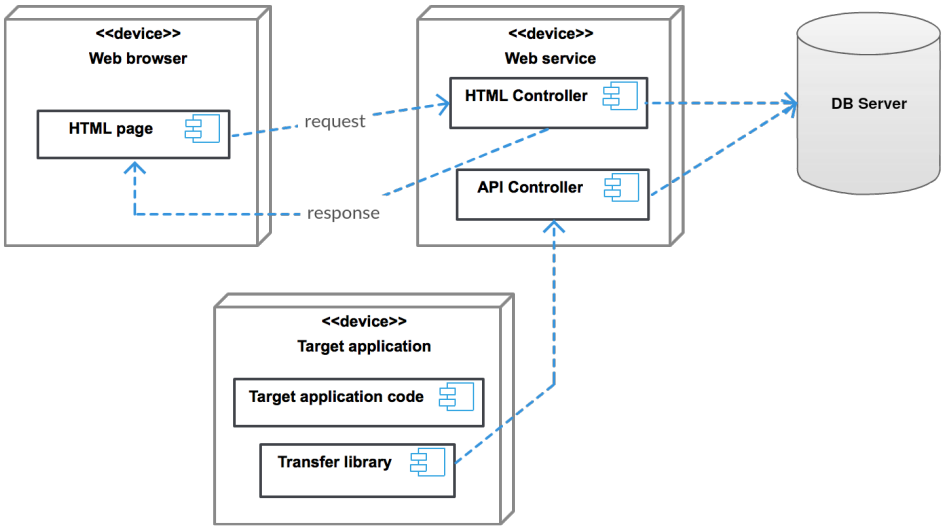


Figure 4. System components diagram.

Target application

This is a completely different web or desktop application with its own web server and hosting on which the code of the target application is located. The target application is a subject of testing, the participant interacts with it, thereby producing data for later analysis. This application measures the execution time, the number of errors, and all sorts of other variables that the researcher wants to work with.

Transfer library

This is a single JavaScript or Java file that is included in the code of the **"Target application"** and is used to transfer data to the **"Web service"** server. Basically, it is a small library that provides an API to communicate with the server.

3.2. Architecture comparison

Before designing a web service it is very important to choose a right architecture that would fit project's requirements. I will discuss 2 technologies: REST and SOAP. Each protocol has certain advantages and equally problematic disadvantages. The choice between REST and SOAP should be based on the individual characteristics of the application. Exactly these requirements of the system in connection with the pros and cons of the two solutions will be considered below.

Before I go any further, it is important to clarify that it is not correct to compare REST and SOAP directly, although they both have some similarities in the use of the HTTP protocol, SOAP itself is a protocol while REST is an architectural style. Both technologies rely on well-defined communication rules that must be followed in order to successfully transmit information.

Simple Object Access Protocol (SOAP)

SOAP actively uses exclusively XML to encode requests and responses, as well as strict data typing, which ensures their integrity during the transfer between the client and

the server.

In 1998, Microsoft released SOAP as a replacement for older technologies such as the Distributed Component Object Model (DCOM) and the Common Object Request Broker Architecture (CORBA). These technologies were based on a binary message and therefore were not successful on the Internet, in contrast to the XML that is used by the SOAP [9]. Another benefit is that SOAP passes through the firewall without problems, while other distribution systems (for example, DCOM) are often filtered by firewalls.

SOAP can be used with any application layer protocol: SMTP, FTP, HTTP, HTTPS, etc. However, its interaction with each of these protocols has its own characteristics, which must be defined separately. The message transmission is often used over the HTTP protocol.

The main characteristic of SOAP is that it is highly extensible using WS-* specifications when it really makes sense, for example WS-Security, WS-Addressing, WS-Transfer, WS-Trust and others [9].

SOAP is always used in conjunction with WSDL file, which provides a definition of how the web service works.

Representational State Transfer (REST)

The term REST was introduced in 2000 by Roy Fielding, one of the authors of the HTTP protocol. REST provides a lighter weight RPC (Remote Procedure Call) alternative. Systems supporting REST are called RESTful-systems. Unlike web services based on SOAP, there is no official standard for the RESTful web API.

However, there are six mandatory restrictions for building distributed RESTful applications, such as client-server architecture, statelessness, cacheability, layered system and so on [10].

If the service application violates any of these restrictive conditions, this system cannot be considered RESTful.

To obtain information and also modify the resources, REST uses the URL approach. The REST architecture is installed on the HTTP protocol and uses existing methods (GET, POST, PUT, DELETE etc.) for resource management, which perfectly match CRUD (Create-Read-Update-Delete) operations.

Comparison

- SOAP is an XML-based protocol, which means that all requests and responses will only be transmitted to XML, REST does not have a strictly defined message format, it supports ASCII, XML, JSON, and any other formats recognized by both the client and the server. In addition, there are no built-in data typing requirements in the REST model. As a result, packets of requests and responses in REST are much smaller than the corresponding SOAP packets and, due to this, they are processed faster and easier.
- SOAP uses HTTP as a transport protocol, while REST is based on it. This means that all existing HTTP-based workflows, such as server-side caching, scaling, etc. continue to work in the REST architecture but for SOAP external tools is needed. Instead of this SOAP services have the additional ability to work with any other transport layer protocol instead of HTTP. Also from a developer perspective, REST requests are generally easier to formulate and understand, since they use existing and well-understood HTTP interfaces.

- Basic concept of REST is based on the resources, while SOAP uses interfaces based on objects and methods. The SOAP interface could contain an almost unlimited number of methods. The REST interface, on the other hand, is limited to four possible operations that correspond to the four HTTP methods. The SOAP request is always POST and often only at one endpoint, so all the information about the desired response of the server is contained in the request body. On the other hand, to achieve the same goal, REST uses the HTTP method, the URL, and the body of the request.
- SOAP architecture is language, platform, and transport independent, different from REST, which requires use of HTTP.
- SOAP provides significant pre-build extensibility in the form of the WS-* standards while REST does not. It is possible to provide similar functionality using third-party solutions.
- One of the most important SOAP features is built-in error handling. If there is a problem with your request, the response will contain error information that can be used to fix the problem. On the other hand REST uses standard HTTP code statuses to define error types.

Based on the above arguments, I will implement the **"Web service"** and the **"Transfer library"** using REST architecture. Since the API communication in this project is quite simple and its calls will come from the JavaScript language, it will be appropriate to use the JSON data format as well as the HTTP protocol which has a lot of convenient features for this particular case.

3.3. Web service structure

In this section functionality of the biggest component ("Web service") will be discussed. The application is intended to conveniently set up the experiment definition. Each user of the system will have the possibility to register in the system, login to the system, and create a new experiment. All aspects discussed in the analysis chapter will be used in the experiment definition as well as in calculations. These include the experimental design type, the data type, and their distribution.

Of course, if there is a research group instead of one researcher, then the system can be used by several users through one account. Therefore, the experiment definition contains some additional fields for description. It helps in the future to quickly recall details and goals of certain parts of the experiment definition.

Usually, the experiment consists of two or more logically separated parts: the introductory part which is designed to collect general information about the participant, the main part of the experiment itself, and the final part of the post-test questionnaire. Each part will have its own unique variables. The age, the gender, the experience in the particular area could be present in the intro part, whereas the main part may have the task time. It is important to note that all values of the variables of one part will be created, sent and stored together, if we are talking about *age* and *gender* variables, then they will be a pair of values.

It is required to collect data after determining the experiment. When all tests with participants are conducted, the researcher may want to stop gathering new data which usually happens during the remote testing. To make this possible it is necessary to have at least three states: *new*, *open* and *closed*, where *new* is the initial state. It is not

difficult to imagine a situation in which the researcher notes an error in the experiment definition only after obtaining the first samples of data. In this situation it would be suitable to change the state from *open* to *edit*, thereby prohibiting the experiment from receiving new data and allowing to edit its definition. Therefore experiment's initial state can be *edit* state instead of *new*. And what if, after the changes in the definition of the experiment, all data obtained up to this point will lose its meaning. There are at least 3 ways out of this situation: erase the old data, try to keep samples that still make sense (for example, if the researcher has changed only the name of the variable or made another minor change) and allow the user to choose to erase the data or leave it. In fact, the transitions between *edit* and *open* states have the character of testing, in other words, debugging. So, to make it easier to distinguish between these states, I decided to introduce an additional state for exactly this purpose - *debug*. Now *debug* and *open* states symbolize development and production environments.

The **Figure 5** shows all the experimental states and their possible transitions. The initial state is indicated in blue, the end state is gray, the rest - yellow.

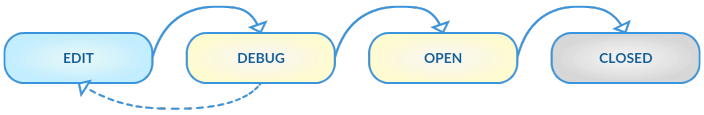


Figure 5. Experiment final state machine.

In the moment when all the data is already collected, it is possible to visualize it. Generally, the collected data is best represented as a *histogram* - a kind of bar chart, where the x-axis shows the values, and the y-axis shows their quantity. The next chart that interests researchers is a *boxplot* - the visual representation of confidence intervals.

For a better overview of the processed data, it would be better to add the desired charts and see them all together on one page.

Remaining components and properties of the experiment will be thoroughly described in the next section.

3.4. Entity relationship diagram

This domain model illustrated in **Figure 6** was generated based on the (prototype) code and tables in the database using the *rails erd* library [11]. This diagram contains all tables and corresponding entities of the **"Web service"**.

The description of entities is provided below. Next to names of the properties their data types are placed. Each entity has properties `created_at` and `updated_at` of data type "Date", which were not written out for the sake of brevity. These properties hold timestamps for corresponding actions.

3.4.1. User

The *User* entity represents the end user of the system. In this case, it is the researcher or a group of researchers. This model contains the properties needed to register and login into the system: `email` and `encrypted_password`. The password will not be stored in an open form, otherwise it would be too insecure. Besides, the properties as `current_sign_in_at` or `current_sign_in_ip` provide information about connections, and other information for manipulating the user account.

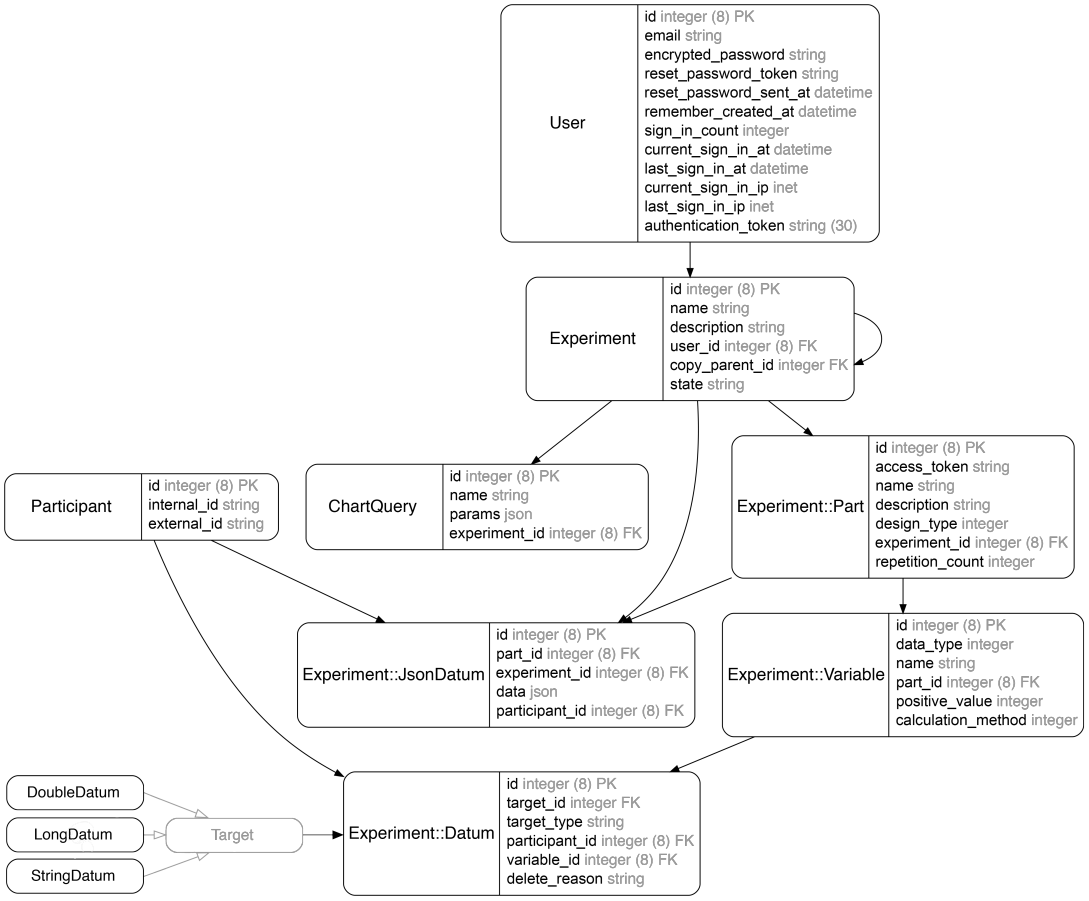


Figure 6. Entity relationship diagram.

3.4.2. Experiment

This entity represents the user experiment. It contains `name` and `description` properties, which are needed to simply identify the test by the researcher. Attribute `state` serves to store the current state of the experiment (see Figure 5). This attribute could hold one of the following values: *edit*, *debug*, *open*, and *closed*. The last attribute `copy_parent_id` is intended to store a reference to the experiment from which this copy was made. The *Experiment* entity belongs to the *User* entity. They have a many-to-one relationship.

3.4.3. Experiment::Part

The *Experiment::Part* symbolizes the division of the experiment into logical sections, each of which is aimed at collecting certain data. Just like in the *Experiment* entity, properties `name` and `description` are used for the researcher's notes. The *Experiment::Part* entity belongs to the *Experiment* entity. They also have a many-to-one relationship. The property `access_token` is the connecting key in the communication between the **"Web service"** and **"Target application"**. Of course, this token must be unique, because it will identify the experiment and its specific part. Therefore, the original idea was to generate a unique hash string but in practical use it would not be convenient for the researcher, since there could be more than one part and it's necessary to precisely control which token belongs to which part and the human factor may interfere in a given situation. With this in mind, it was decided to replace the hash

string with a combination of the experiment's name in conjunction with the name of the appropriate part separated by a hyphen. The received token is much more readable by a human and also it still has its uniqueness in the combination "experiment name - part name".

The `design_type` property is responsible for the type of experimental design: *between subject* or *within subject*, it was already explained in the analysis chapter. Property `repetition_count` describes the number of tuples that will come from one participant within one part.

3.4.4. Experiment::Variable

For each part of the test, you need to define the variables that will be measured (in the **"Target application"**), sent (by the **"Transfer library"**) and processed (on the **"Web service"**). This variable corresponds to both the independent variable in the experiment and the dependent one. For example, the participant's age or other indicators are independent variables. This also applies to the identifier of the group within which the participant performed assignments or may be the identifier of the task. The task time or error rate could serve as an example of the dependent variable.

A variable is defined by `name` and `data_type` properties, which should be one of three types: *Long*, *Double*, or *String*. In order to perform correct calculations, property `calculation_method` is added to this table. It is an enum with several possible values: *log transformation*, *normal distribution*, and *binomial distribution*. Each value determines which formula will be used to calculate the confidence intervals. For example, if the *log transformation* value is selected, this indicates the necessity to transform values before calculating the confidence intervals.

Property `positive_value` is used to determine the value that the researcher considers to be a positive response, it is essential for the calculation of the discrete values.

3.4.5. Experiment::Datum

The incoming data needs to be stored somewhere. For this purpose there is the *Experiment::Datum* entity, which represents a unit of measured data. The data is associated with a participant through the `participant_id` property.

The following properties `target_id` (entry id) and `target_type` (name of the model class) are designations for the *polymorphic association*. The polymorphic association allows the entity to belong to more than one model with a single association. A helper entity *Target* can be perceived as an interface for *LongDatum, DoubleDatum, StringDatum* tables. This approach was taken because of the need to keep the collected data in three different tables due to the inconsistency of data types.

3.4.6. LongDatum, DoubleDatum, StringDatum

In diagram 6, these entities are drawn without attributes for compactness. These entities represent cells for different data types. They contain only the property `value` and belong to entity *Experiment::Datum*.

3.4.7. Participant

This entity represents the participant of the experiment. It has neither the name nor any other personal information. To identify participants the `internal_id` or `external_id` is used. Both values take an alphanumeric value. `internal_id` is a server-side generated

UUID (Universally Unique Identifier) string. The UUID is a 16-byte number. In hexadecimal notation, the UUID looks like:

```
7df4b211-135d-4210-af3d-982c9e045efc
```

If the researcher has his own user management and wants to use their unique identifiers, then the presence of the attribute `external_id` makes it possible. That is why the property `external_id` is not produced on the **"Web service"** server, but only stored there. Along with experimental data, an `internal_id` or an `external_id` is sent in order to link the data to the participant.

3.4.8. Experiment::JsonDatum

This table is also intended to store data in the JSON format. It belongs to several other entities: *Experiment*, *Experiment::Part*, and *Participant*. The entity has only one attribute - `data` with the *json* data type. The *json* data type is not a standard, however many databases such as MySQL, PostgreSQL, MariaDB support it. Such data can also be stored as a text, but for JSON data types, the advantage is to observe the validity of the stored data as well as the existence of specific functions and operators available for data stored in these data types.

The data from the experiment always comes in a connection with each other, for example, if the participant in the intro part submits his age and gender, then these two values are related and this connection must be maintained. Current data storage does not provide a way to restore this connection. One of the solutions to that might be the addition of some indicator for each data bundle, i.e., adding one more column. And then in order to filter the data of one variable by another it will be necessary to create fairly complex queries to the database using 3 tables: *Experiment::Variable*, *Experiment::Datum* and one of *StringDatum*, *LongDatum*, *DoubleDatum*. The introduction of the new JSON table makes it possible to produce much simpler and more effective queries, like these:

```
SELECT data -> "age" FROM "experiment_json_data"
WHERE "experiment_json_data"."part_id" = 123
AND (data ->> "gender" = "male")
```

This query returns all values of the variable named *age* filtered by a variable named *gender* and its value "male" in the context of the part with *id* equal to `123`.

3.4.9. ChartQuery

This entity is used to store the parameters of the boxplot definition. It has a property `name` that describes the graph, thereby helping the researcher to navigate among them. In addition, it is unique in the experiment's context.

Property `params` also has the data type JSON and contains the settings for the boxplot, namely the `target_variable` whose values will be calculated, the `filter_variable`, the variable which will serve as a filter for `target_variable`, and `filter_variable_values` will store the values of the selcted filters for each boxplot section. The JSON data type is extremely flexible and easily extensible, in this case it is very convenient because if there is a need to add a new parameter to define the chart during the development, you will not need to change the database and add or remove columns.

It belongs to the entity *Experiment* with many-to-one relationship.

3.5. Transfer library design

From the Figure 4 it is clear that the **"Web service"** includes a separate API controller for communicating with the **"Transfer library"**. Every sample of data must be associated with an individual participant. Therefore, before sending the data, it is necessary to register the participant and then send the data with his personal identifier (`intertnal_id`).

Since one of the libraries will be written in JavaScript and will be used on the client (web browser), then all transmitted data including the tokens will be visible using the browser built-in developer tools. However, without the participant's `internal_id`, the data is not valid. So if a participant wants to change his results, it will be complicated because the server is checking the number of data repetitions from one participant, then he will have to go through a new registration and send the data again, which is equivalent to a situation if he tries to conduct the experiment again.

Besides that, the library must be implemented according to REST guidelines. REST implies that every URL endpoint should have a specific format. There are several distinct rules for this. For example, it is highly recommended, when designing URL addresses for endpoints, to use the resource names in the plural form (ex.: books, users). Another common rule is that the GET method is used for *Read* operation from CRUD.

Further, I will give examples of endpoints for standard CRUD operations:

```
GET /books           - request for a list of all books

GET /books/1         - request for certain book with id = 1

POST /books + body   - create new resource

PUT /books/1 + body  - update resource

DELETE /books/1      - delete resource
```

It implies that the API controller must provide 2 endpoints in the following form:

- **Participant registration**
  Endpoint:

  ```
  POST /api/v1/participants
  ```

  Request example (optional):

  ```
  body: {
    "external_id": "my-own-unique-id"
  }
  ```

  Response example:

  ```
  200 OK

  body: {
    "internal_id": "1ee21a7d-d5ff-4c92-a723-5fa9239cddd4"
  }
  ```

- **Send data**
  Endpoint:

```
POST /api/v1/experiments/parts/:part_id/data

(part_id = "test-info-part")
```

  Request example:

```
body: {
  "internal_id": "1ee21a7d-d5ff-4c92-a723-5fa9239cddd4",
  "variable_values": [
    {
      "name": "age",
      "value": "25"
    },
    {
      "name": "gender",
      "value": "male"
    }
  ]
}
```

  Response example:

```
422 Unprocessable entity

{
  "status": 422,
  "message": "internal_id or external_id are missing"
}
```

Below I will list all the errors returned by the **"Web service"** grouped by HTTP status:

403 Forbidden

- Experiment is currently unavailable

404 Not Found

- Experiment's part not found
- Variable with name `<name>` not found
- Participant not found

422 Unprocessable entity

- `internal_id` or `external_id` are missing
- `variable_values` are missing
- `variable_values` must be an array
- Size of `variable_values` is not correct
- Variable's repetition count is exceeded

3.6. Requirements

3.6.1. Functional requirement

Based on all of the above and also taking researchers' needs into consideration it is possible to make a more specific list of desired requirements for the future **Web service**:

F1: Register to the web application
The user will be able to sign up to application with his own email and password.

F2: Login to the application
The user will be able to login with an email and a password.

F3: Logout the application
The user will be able to log out from the application.

F4: Edit personal account information
The user will be able to edit its email, password and add its name.

F5: Delete account
The user will be able to permanently remove its account.

F6: Create new experiment
The user can define and then save a new experiment. The definition of the experiment also includes the definition of a part and its variables.

F7: Edit the experiment
The user will be able to edit the experiment but only if the current state of the experiment allows this (*edit* state).

F8: Delete the experiment
The user will be able to permanently delete the experiment with its associated entities, i.e. parts, variables and its values.

F9: Duplicate the experiment
The experiment could be copied. The whole definition will be copied excluding the collected data, also due to the preservation of the uniqueness of the part's *access tokens*, the ordinal number will be added to the name of the copied experiment.

F10: Export experiment as JSON file
The user could export any of his experiments in the form of a JSON file. This file will contain a description of the experiment, its parts and variables as well as the values of the collected data in the nested tree form.

F11: Change experiment's state
The user will be able to change the state of the experiment according to the transition rules. See [Figure 5].

F12: Show histogram for collected data of selected variable
The user could select a variable within the experiment's part and display a histogram for it.

F13: Show variable's raw data
Also, the user can view raw data (= the values) of the selected variable.

F14: Add new boxplot definition for the experiment
The user will be able to define boxplot parameters, that is, select the *target variable*, the *filter variable*, and the *filter variable values* from the available values.

F15: Show all boxplots associated with the experiment
On the experiment detail page, the user can see all previously defined charts (boxplots).

F16: Show detail table with CI values
Under the boxplot the user can see the same values but in the form of an overview table.

F17: Delete the boxplot definition
The user will be able to permanently remove a boxplot definition.

F18: Download the selected chart image
The user could save the image of the selected boxplot or histogram.

F19: Change settings of the selected chart:
Each graph will have its own settings, which include the following parameters:

1. Show/hide legend
2. Change legend position
3. Show/hide X axes
4. Show/hide Y axes
5. Set step size for Y axe

F20: Register participant
The participant will be able to register, i.e. server will generate a new unique UUID and save it.

F21: Save collected data
The server will accept, validate, and save data from the participants.

3.6.2. Non-Functional requirement

N1: REST architecture
The whole project is built according to the REST architecture constraints.

N2: Data type for API messaging is JSON
All data communication between the components is using JSON format.

N3: PostgreSQL database
The underlying database of the server will be PostgreSQL.

4. Implementation

This chapter contains details of the implementation, a small introduction to the used technologies, the structure of the **"Web service"** project, and code snippets with their description. Moreover, I will discuss some problems encountered during the development and possible solutions to them. The **"Web service"** is a major part of the system, but the system requires the **"Transfer library"** in order to function as intended. That is why the library will be also described in this chapter. Finally, the finished application will be presented using screenshots and additional comments.

4.1. Web service

4.1.1. Selected technologies

In this section I will briefly describe the most significant technologies and tools that were used to implement the project. Each tool is explained in the following subsections, and it is also important to mention *Git*[1] as a versioning system which was used to maintain code changes, to capture the progress, and to store sources on the remote server. It was used throughout the project because it is a standard recommended by many developers.

Ruby on Rails

In my work I used the **Ruby** programming language and the web framework **Ruby on Rails** also known under the names *Rails* or *RoR*. This framework implements the MVC (Model-View-Controller) architectural template for web applications, and it also includes all the necessary tools for the basic needs of the web developer, such as the default web server, scaffolding[2] and other useful packages for testing, deployment or debug. It is an open source software, and it is distributed under the MIT license [12].

To store the model objects in the relational database, the ActiveRecord library is used. Using this design pattern the "rows" in the database are converted to instances of objects and "columns" to their attributes. Besides the MVC architecture, Rails also use other principles and paradigms of the web development among which are:

DRY (Do not repeat yourself) - it has mechanisms for the reuse, which make it possible to minimize duplication of code in the application.

Convention over configuration - an explicit configuration specification is required only in non-standard cases.

In Ruby on Rails, by default the *view* is described using ERB (Embedded Ruby) templating system. It is a HTML file with additional inclusions of Ruby code fragments. Many other templating languages can be used instead of ERB. For this particular project I chose *Slim* template language. Slim is a template language whose goal is to reduce the view syntax to the essential parts without becoming cryptic [13]. Slim offers a minimalistic look at HTML templates. With such templates it is more convenient to

[1] https://git-scm.com/

[2] automatic code generation for models, view, controllers, etc.

work, and they are simple to maintain. The resulting HTML code is always valid and not a single closing tag will be missed by mistake.

Rails have their own package and dependency manager - *Bundler*[3]. Libraries or packages for Ruby language are called *gems*. A file in which all the required libraries for the application are listed is called *Gemfile*.

PostgreSQL

In the previous chapter, I described the models in which I use the JSON data type. In this section I would like to take a closer look at the implementation of this data type in a specific database. PostgreSQL has 2 JSON data types: *json* and *jsonb*. In general, they look almost similar, the main difference is how they process the input data. The *json* data type stores the data in its original form, in contrast with the *jsonb* type which stores the data in a decomposed binary format, so it may take more time due to additional conversion overhead, but as a result, working with this type will be much faster, since no reparsing is needed. *jsonb* also supports indexing, which can be a significant advantage [14]. Thus, I chose the *jsonb* data type.

Chart JS

For the graph visualization, I chose the library *Chart.js*. It is a simple, nice-looking, Open Source JavaScript library that uses HTML5 Canvas for the chart rendering, supports 8 chart types and is fully responsive [15]. Chart.js animates charts out of the box, but it is possible to customize animation styles and the duration. There are various options to change tooltip's style, location, and content, or it can be completely disabled. All that is required to draw a graph is just one `canvas` tag and a js script in which a new *Chart* instance is created with the settings: `type`, `data`, `options`, etc.

The default version of the library does not support the boxplot chart, but, fortunately, there is the fork[4] that adds the desired chart type under the name *"Error bars chart"*. This version is fully compatible with Chart.js 2.x.x.

UI

For the visual side of the site the following libraries and frameworks were used: *material icons* (set of icons), *skeleton* (lightweight, responsive css framework), *toastrJS* (a library for non-blocking flash notifications), *jQuery UI* (set of UI interactions, effects, widgets, and themes).

Heroku

This is a cloud-based PaaS (Platform as a Service) platform that supports a number of programming languages. Heroku initially supported only the Ruby programming language, but at the moment the list of supported languages also includes Java, Node.js, Scala, Clojure, Python, Go and PHP [16].

Heroku allows developers to deploy, run, build and manage applications. It provides CLI, Pipelines (Continuous integration), several environments for deploying and running applications, and a lot of other useful products. Heroku has an integration with systems such as Git and GitHub, so a developer can connect the GitHub repository with an application on Heroku or use the repository provided by the Heroku itself.

[3] http://bundler.io/
[4] https://github.com/CAYdenberg/Chart.js-ErrorBars

Heroku requires a minimum of effort for the deploy: it automatically detects the language, the framework, and the specified platform, prepares the deploying script depending on the platform, furthermore, it recognizes a database adapter which is used in the application and sets up it accordingly and it provides an automatic deploy after pushing the code to the repository.

Summing up I can say that Heroku creates an ideal sandbox environment for deploying small web apps for free, which was the reason behind my choice.

4.1.2. Project structure

This is the structure of the project. It should be noted that this list is not complete for the sake of simplicity.

```
server
├── app
│   ├── assets
│   │   ├── images
│   │   ├── javascripts
│   │   └── stylesheets
│   ├── controllers
│   │   ├── api
│   │   │   └── experiment_data_controller.rb
│   │   ├── application_controller.rb
│   │   ├── chart_queries_controller.rb
│   │   ├── experiments_controller.rb
│   │   └── home_controller.rb
│   ├── helpers
│   │   ├── application_helper.rb
│   │   └── experiments_helper.rb
│   ├── models
│   │   ├── experiment
│   │   │   ├── part.rb
│   │   │   ├── variable.rb
│   │   │   └── ...
│   │   ├── experiment.rb
│   │   ├── participant.rb
│   │   ├── user.rb
│   │   └── ...
│   └── views
│       ├── chart_queries
│       ├── experiments
│       ├── home
│       ├── devise
│       └── layouts
├── bin
├── config
├── db
├── lib
│   └── ci_calculator.rb
└── test
```

It is important to mention that Rails rely on naming conventions throughout the whole application, so there are several naming rules to follow regarding folders and files. For example, there is a model with name *"User"*. So that the controller file will be named `"users_controller"`. Then the controller will have a subfolder in `"views"` folder with the name `"users"` and that subfolder will contain template files with names corresponding to equivalent controller's action names.

Following is a detailed description of the purpose and contents of some folders from the file structure above.

`app` - Main project folder, contains major parts of the application.
`assets` - This folder holds static project assets such as images, all **`js`** files and **`css/scss`** files.
`controllers` - This directory holds controller's files. If the controller has a namespace, it must be in the folder with the name of this space. Each new namespace creates a new nested folder. In this particular case, the namespace and the corresponding folder is **`api`**.
`helpers` - This folder is for view helper files. In the *Rails* context view helper class is always associated with the controller and all of its methods are available in the corresponding view files.
`models` - Folder for model files. Each model inherits from the ActiveRecord and has such methods as **`where`**, **`select`**, **`order`**, etc. and its instance is an object representation of the database row. In this case, there is also a subfolder corresponding to the namespace **`experiment`**.
`views` - This folder holds all view templates, divided into subfolders that related to the controllers. In this case the principle of *"Convention over configuration"* is clearly seen: each controller has a subfolder excepts api controller, also there is a subfolder `layouts` designed for the application template files that define the layout and common page blocks so the remaining files can only contain the page content without having to repeat the same page blocks (for example meta tags or top navigation menu).
`bin` - Contains Rails script that starts your app and can contain other scripts you use to setup, update, deploy, or run an application [17].
`config` - All configurations are available here: application's routes, database, and more. Also, there is a folder **`locales`** which contains language files (`en.yml`) to support different language mutations.
`db` - This folder is designed to hold a current database schema as well as database migrations.
`lib` - Extended modules are stored here. For example, *CiCalculator*, it is a helping class that holds Confidence Interval settings such as `data`, `calculate_method`, `precision` (= 3 by default) and `confidence_level` (= 0.95 by default) and provides all the calculations (compute means, proportions, standard deviations, lower/upper CI bounds and so on).
`test` - This folder contains unit tests, fixtures, and other test files [17].

4.1.3. Prerequisites

The following libraries and tools are required to run the **"Web service"** application:

- Ruby 2.4.0+
- Rails 5.1.0+
- NodeJS 8.3.0+
- npm 5.6.0+

- PostgreSQL 9.5+

4.1.4. How to start the app

It is important to remember that all subsequent commands are always executed inside the project folder from the command line. To start working it is needed to install the Ruby and NodeJS of the specified version. After that it is required to download the dependency manager *bundler* with the following command:

```
gem install bundler
```

This dependency manager then downloads Rails framework and all the necessary libraries specified in the *Gemfile*. In addition, you also need to download the libraries for the NodeJS using the commands:

```
bundle install
npm install
```

During the initial configuration you need to download the dependencies and create the database with the command:

```
rails db:create
```

And then you need to execute migrations. Remember, that every time you add a new migration, you need to run this command again, since the Active Record keeps a list of the performed migrations, only the new ones will be executed.

```
rails db:migrate
```

After that it is no longer required, you just need to start the web server with a simple command:

```
rails s
```

This will start the application on a default port `3000`, if you want to change the port then simply use the following command:

```
rails s -p 8080
```

4.1.5. Code samples

In this part I will show code snippets from the project that were mentioned earlier in the text.

Database

Database configuration in Rails looks as follows:

```
default: &default
  adapter: postgresql
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  timeout: 5000

development:
  <<: *default
  database: dev_db
  username: root
  password: ""

test:
  <<: *default
  database: test_db

production:
  <<: *default
  database: prod_db
```

Rails distinguish 3 main environments: `development`, `test` and `production`. For this reason you need to define database configuration options for each environment. General parameters can also be specified in the `default` block. It is highly recommended to have different databases for these three environments.

Router

Rails *router* serves as an entry point for each incoming request. All routes are described in the file `config/routes.rb` where by using different syntax structures URL address and its HTTP method can be mapped to the appropriate controller and its action. It can also generate paths and URLs, avoiding the need to hardcode strings in your views [18]. Standard CRUD routes are being generated automatically by just one code line `resources :experiments` (it will create 8 standard routes, see below), otherwise custom routes with all the params such as URL, HTTP method, matched controller's action, and optionally path's name needs to be defined separately.

The following routes (8 standard and 5 custom)

```
GET    /experiments(.:format)                experiments#index
POST   /experiments(.:format)                experiments#create
GET    /experiments/new(.:format)            experiments#new
GET    /experiments/:id/edit(.:format)       experiments#edit
GET    /experiments/:id(.:format)            experiments#show
PATCH  /experiments/:id(.:format)            experiments#update
PUT    /experiments/:id(.:format)            experiments#update
DELETE /experiments/:id(.:format)            experiments#destroy

GET    /experiments/:id/copy(.:format)       experiments#copy
GET    /experiments/:id/to_debug(.:format)   experiments#to_debug
GET    /experiments/:id/to_edit(.:format)    experiments#to_edit
GET    /experiments/:id/to_open(.:format)    experiments#to_open
GET    /experiments/:id/to_closed(.:format)  experiments#to_closed
```

were generated using this code:

```
resources :experiments do
  member do
    get 'copy'
    get 'to_debug'
    get 'to_edit'
    get 'to_open'
    get 'to_closed'
  end
end
```

Experiment FSM

Experiment states and their transitions were implemented using the gem library *state_machines-activerecord*. It adds the experiment state machine to the model attribute `state` by observing possible values and changes and has the following syntax:

```
state_machine :state, initial: :edit do
  event :to_debug do
    transition :edit => :debug
  end

  event :to_edit do
    transition :debug => :edit
  end

  event :to_open do
    transition :debug => :open
  end

  event :to_closed do
    transition :open => :closed
  end

  after_transition on: :to_edit, do: :clear_data
end
```

This code is responsible for several things: the attribute which holds the experiment state, the initial state, and transitions defined by the keyword **event**. The last line stands for adding an extra action after a certain transition, in particular, the execution of the method `clear_data` on the *Experiment* model instance.

The call of these transitions is performed in the *ExperimentsController*. Each event of the status change has its own route and action in the controller, which essentially performs the transition accordingly to the name of the action. Method `change_state` generalizes all of these actions into one universal code. It calls the transition by its name, sets the appropriate flash notification, and then redirects to the experiment detail page.

To make this possible without writing controller actions for each transition, I used the standard `action_missing` and `method_missing` methods, which are automatically called if a nonexistent method is called on a controller.

```
def change_state(transition)
  if @experiment.send transition
    flash[:notice] = 'Experiment state was successfully changed.'
  else
    flash[:alert] = 'Invalid experiment state transition.'
  end
  redirect_to @experiment
end

def action_missing(name)
  begin
    self.send name
  rescue
    super
  end
end

def method_missing(name, *args, &block)
  return change_state(name) if respond_to?(name, true)
  super
end
```

Chart.js modification

As I wrote earlier *Chart.js* library was used for the data visualization and specifically histograms, bar chart, and error bars (from the previously mentioned fork). This fork adds a new kind of graphs - error bars, which are defined in the following way:

```
var myChart = new Chart(ctx, {
  type: 'barError',
  data: {
    labels: ["January", "February", "March", "April", "May"],
    datasets: [
      {
        label: 'Dataset 1',
        backgroundColor: "rgba(220,220,220,0.5)",
        data:  [1,   2,   3,   4,   5],
        error: [0.1, 0.2, 0.3, 0.4, 0.5]
      }
    ]
  },
  options: {
    responsive: true,
    ...
  }
});
```

This code snippet describes the creation of a new instance of Chart with the `type` *barError*, then the `data`, as well as the settings `options`, are specified. Let us take a closer look at how that data is set, namely the *datasets* parameter. It is an array and can consist of an unlimited number of objects, each of which is a single dataset with its own label, color, mean values (`data`), and the corresponding values of the error margin (`error`). Note that only the size of the error is specified because the confidence interval is symmetric towards to the mean value, the exception is when the logarithmic transformation of data is used to correct the positively skewed time data, and the resulting interval becomes asymmetric. In order to satisfy all possible cases, I have slightly corrected this library, so now not only the size of errors is set but also the upper and lower bounds of the interval. Next, I will give an example of the `datasets` parameter:

```
datasets: [
  {
    label: 'Dataset␣1',
    backgroundColor: "rgba(220,220,220,0.5)",
    data:   [1, 2, 3, 4, 5],
    uppers: [1.1, 2.2, 3.3, 4.4, 5.5],
    lowers: [0.9, 1.8, 2.7, 3.6, 4.5]
  }
]
```

4.2. Transfer library

The library consists of just one file that defines a class *Transfer* with 2 static methods: `registerParticipant` and `sendData`. These functions are called in the following way:

```
Transfer.registerParticipant();
Transfer.sendData(partToken, data);
```

Both functions perform an AJAX call to the endpoint. Since AJAX is an asynchronous call that returns a *Promise*[5] because it needs time to process a request. Next, I can add a callback function for the successful completion of the request `promise.done(..)` and for the unsuccessful `promise.fail(..)`. Transfer library methods return a promise object, so the developer has a possibility to add custom callbacks via `done` and `fail` methods.

4.3. Results

In this part I would like to show few UI examples of the resulting application and describe them.

List of experiments

The **Figure 7** shows a list of experiments. They are written out in the form of a table. Experiments can be viewed in a more detail, edited if they have the "EDIT" status, or deleted.

[5] object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value [19].

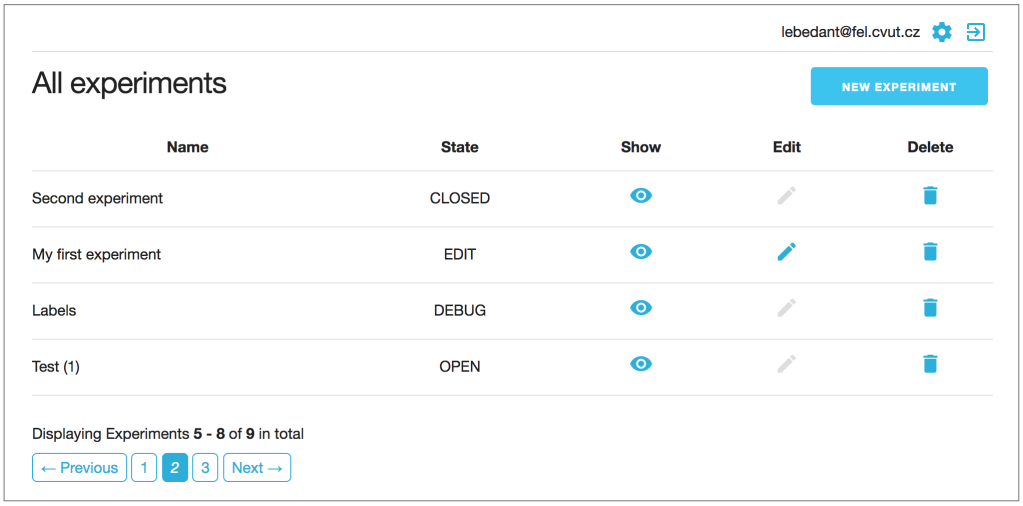


Figure 7. List of the experiments.

On the top side there is a user's profile and an icon (settings) that leads to the form where the user can change his personal information and also the icon (exit) to log out from the system.

Detail of experiment

Figure 8 shows the detail of the experiment.

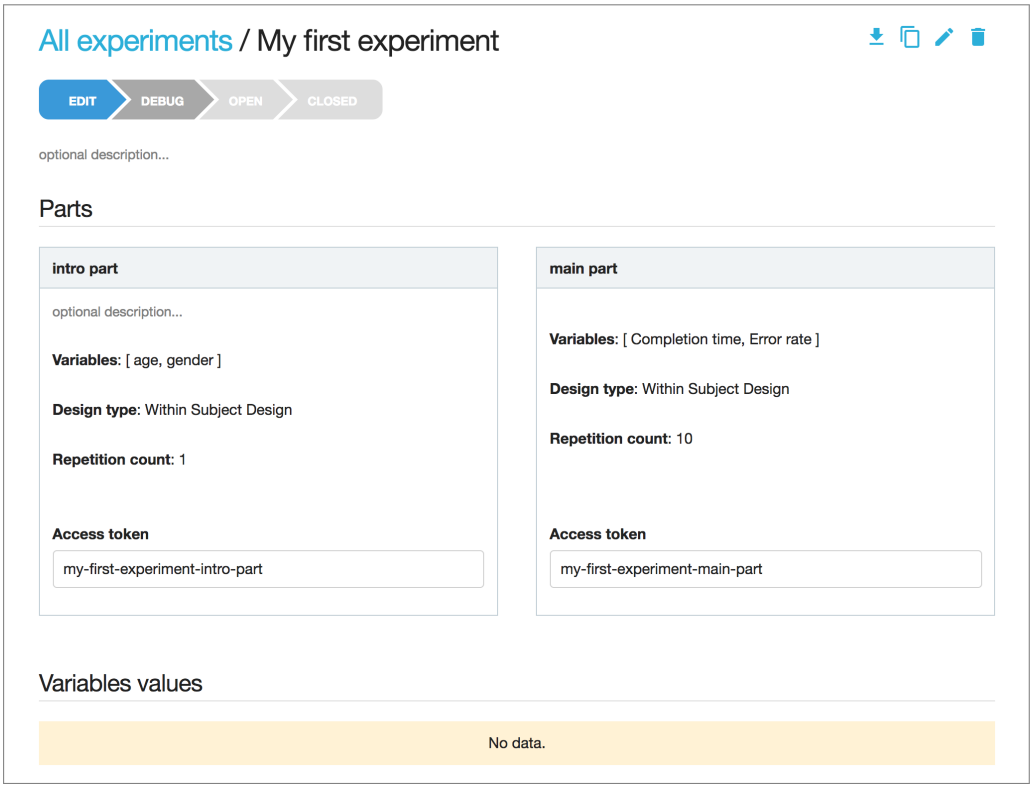


Figure 8. Experiment detail.

At the top there is its name and also a link to the list of all experiments in the breadcrumb navigation; the actions that can be performed with this experiment are

located to the right of the name: export the data as a JSON file, copy this experiment, edit it, or delete (after this a standard dialog appears to confirm an action).

Then the description of the experiment and the block with experimental states is shown where the current one is highlighted in blue, available for a transition in dark gray, and inaccessible in light gray.

The next part of the page holds the description of the experiment's parts; there is the information specified by the user during an experiment creation and the generated access token as well.

The rest of the page is dedicated to the analysis and the visualization of the collected data. In this example they are not yet obtained, and, therefore, the user sees a yellow bar indicating this.

Charts

Figure 9 shows an example of a histogram chart that is designed to visually check the distribution of collected values across variables.

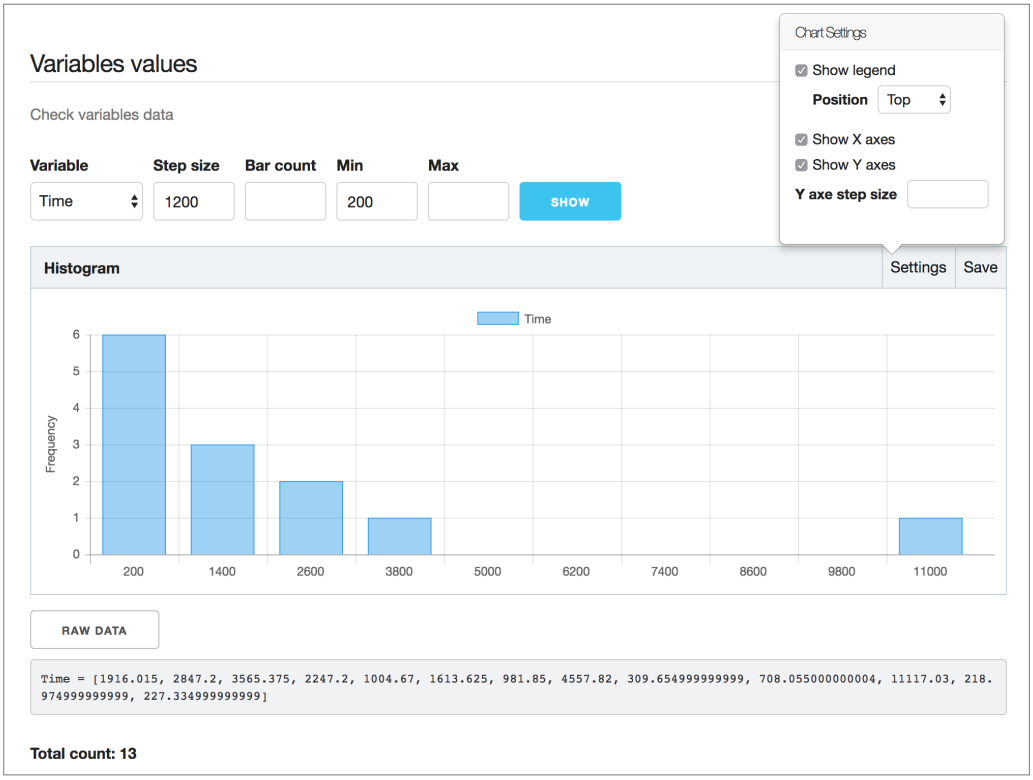


Figure 9. Histogram sample.

Using the form located on the top, the user can select a particular variable whose values are shown in the histogram. Also, for the user's convenience, I added the ability to configure the histogram by such parameters as a step size, the total number of bands, minimum and maximum values.

Values located under the bars on the horizontal axis represent the interval between the previous value and the current one. In the screenshot the intervals will be **[200, 1200)**, **[1200, 2200)**, **[2200, 3200)** and so on.

In the upper right corner the block of the chart settings is visible. It appears by clicking the "Settings" button. All changes occur immediately, so the user can set up a

chart by himself in a few seconds and download the resulting image by clicking on the "Save" button.

The "Raw data" button is placed under the graph, which shows or hides the list of variable values that are displayed in the histogram. The user can easily copy these values and use them at his own discretion.

The next very important chart is the boxplot (**Figure 10**). This is the visualization of confidence intervals. On the experiment detail page, the user can add his custom boxplot using a special form that is located above all the boxplots. But to make the text more compact I arranged the form and the boxplot next to it.

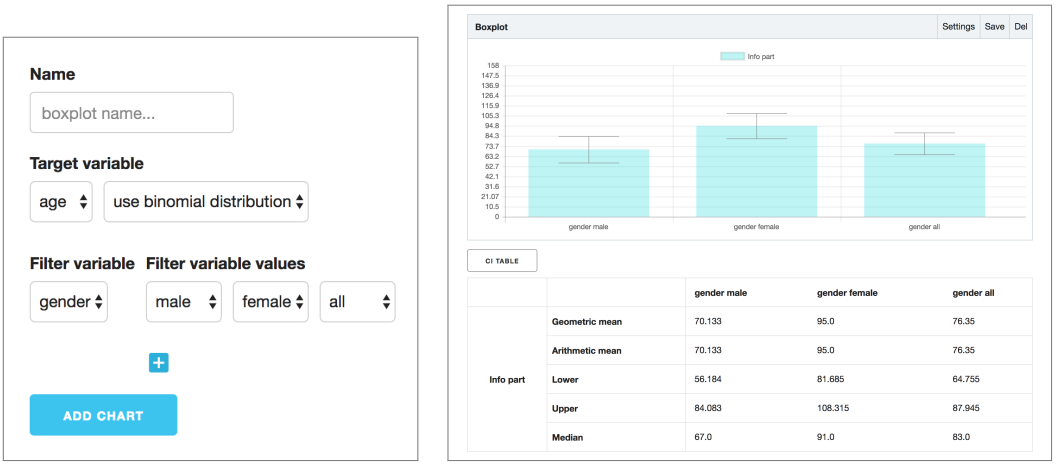
In this form the user needs to specify the name of the boxplot, to select a variable whose values will be calculated, and to choose a method that will be used for calculations (log transformation or distribution).

The next step is to select a variable that will filter the values of the target variable. A filter variable is selected from string type variables in order to separate the analyzable variables from the informative ones. After selecting a filter variable, the user needs to specify its values by which the data itself will be filtered. The "+" icon adds a subsequent value, and the number of these values corresponds to the number of compartments on the horizontal axis.

It is also possible to use the "all" value that does not take the filter into consideration and calculates all values of the target variable.

This graph also has the "Settings" and the "Save" button, the same as the histogram in **Figure 9**. The new "Del" button removes the definition of this chart and it no longer appears on the page.

The boxplot as well as the histogram has a button under the chart to display confidence interval values in the form of a table.

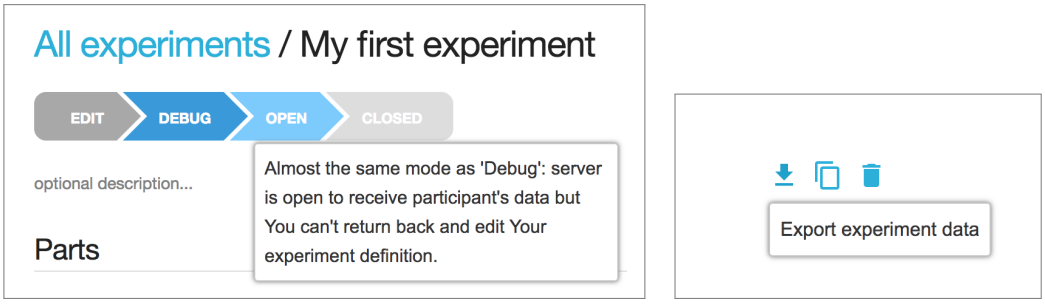


a) Form for a new boxplot. **b)** Example of boxplot.

Figure 10. Boxplot examples.

Tooltip sample

Further examples of clarifying titles to some UI elements will be presented. (**Figure 11**). In this particular case descriptions of the statuses of the experiment are shown in **Figure 16a**. The tooltip to action icons (**Figure 16b**) and the tooltip for some fields in the experiment form (**Figure 12**) are also present.



a) Tooltip on experiment's "Open" state. **b)** Tooltip on "Export" icon.

Figure 11. Examples of tooltips.

Experiment form

Another equally important part of the **"Web service"** is the form for defining a new experiment which is partially visible in **Figure 12**. The *"Experiment"* object contains only the `name` and `description` properties, but it is also important to define the *"Part"* and *"Variable"* object and link them together. Therefore the *"Experiment"* form contains a nested form for *"Part"*, and *"Part"*, in turn, contains a nested form for *"Variable"*.

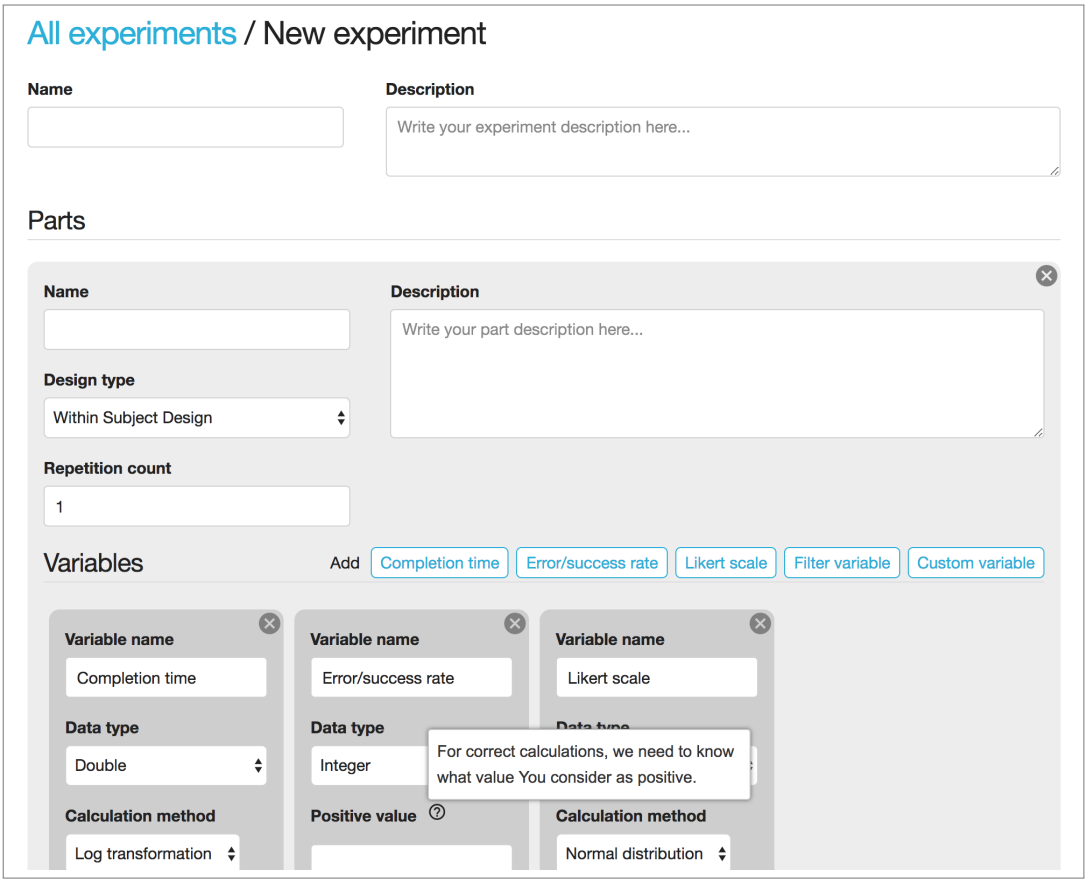


Figure 12. Cut from experiment creation form.

Parts can be added with the **"add part"** button and deleted using the **"x"** icon in the upper right corner of the gray block. The same applies to adding/removing variables with one difference - some variables are already predefined, so the user just

needs to press, for example, the **"Completion time"** button, and a variable with prefilled fields will be added. Those prefilled fields, of course, can be changed. The **"Custom variable"** button creates a variable with empty fields that the user should fill in.

5. Testing

This chapter describes testing the system as a whole with real users. The target application will be explained in detail as well as the setup of the experiment and its results. In the end, the calculations will be compared with the results from public sources.

5.1. Target application prototype

To check the operation of the entire system all three of its components should be present: **"Web service"**, **"Transfer library"**, and **"Target application"**. For testing purposes I implemented a simple client application for testing the labeling methods.

The underlying idea of this application is that the participant is provided with a picture of the scheme of some object or system with signed labels. Each time one of the parts of the system/object is highlighted, and the participant must determine which label refers to the selected part and click on the corresponding label.

Figure 13 is an example of such picture. This schema shows a drill with a tip highlighted in green color which corresponds to the **Label_4**.

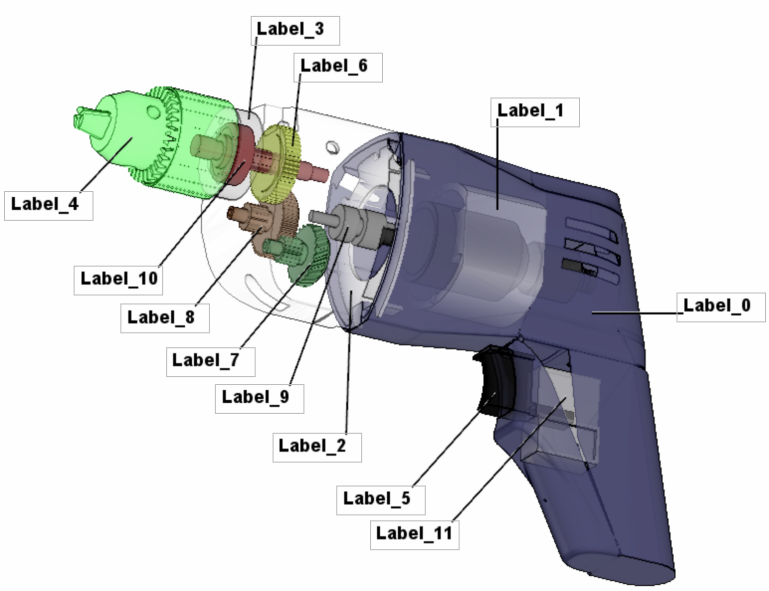


Figure 13. Model Drill.

The purpose of the experiment is to test which method of the label location is more understandable and effective for users. I chose 2 *methods* for testing and 4 *models* (schemes) on which they will be applied.

With this in mind, the experiment was designed and setted up in the **"Web service"**. The experiment consisted of three parts: an introductory part in which participants get acquainted with the rules of the experiment, and in addition to this they are asked about their age and gender, and two identical parts for each labeling method.

It should be noted that it is very important to think about the experiment split, how

many parts will be there, which variables will be gathered, and what will be compared, since the parameter that will be compared (method, UI solutions) should be defined as a separate part.

In the main part every participant will see pictures with labels and one highlighted part of the scheme. As soon as the participant presses the *"start"* button, one of the parts will turn green and from now on the time will count down for finding the corresponding label.

This time will be sent to the server, along with the error code: `0` - the answer was correct, `1` - not a valid label was clicked, `2` -*"I cannot decide"*, `3` -*"No correct label"*. In addition, in conjunction with the time and the error code, the name of the current model will also be sent, so that later it will be possible to filter the data by the *"Model"* variable.

This (Figure 14) is how the definition of the experiment in the **"Web service"** system looks like for this particular target application.

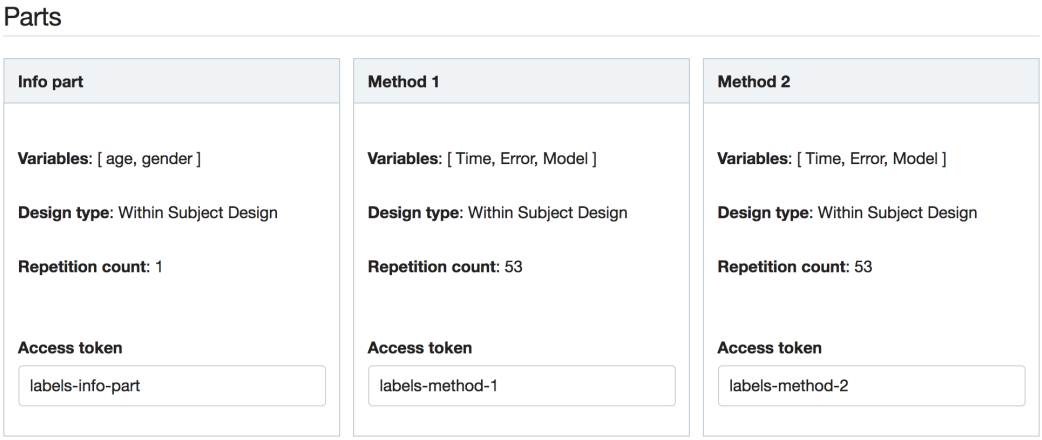


Figure 14. Experiment details.

The *"repetitions count"* parameter displays the number of clicks from the one participant. Taking into account that all **4** models will be shown to all the participants, but with different methods, **12** labels for model *"digestive"* + **12** labels for model *"drill"* + **17** labels for *"fork"* + **12** labels for *"head"* = **53**.

The entire target application was written using HTML, CSS, Javascript and the jQuery library. The code is included in a file together with the *"Transfer library"*.

Ten people have participated in the experiment. Since the experiment had the within-subject design, that is, one participant passed the test with both methods, I used the counterbalancing technique - from the model to the model the methods were alternated as shown in **Table 9**.

	digestive	**drill**	**fork**	**head**
P1	M1	M2	M1	M2
P2	M2	M1	M2	M1
P3	...			

Table 9. Counterbalancing table for labeling experiment.

The upper row describes the sequence of models, and the subsequent rows describe the alternation of methods (**M1**, **M2**) for the models for each participant (**P1, P2**..).

After the experiment, I got the following results. **Figure 15** shows the boxplots for the *Time* variable of the two methods filtered for each model and also the boxplots without any filter.

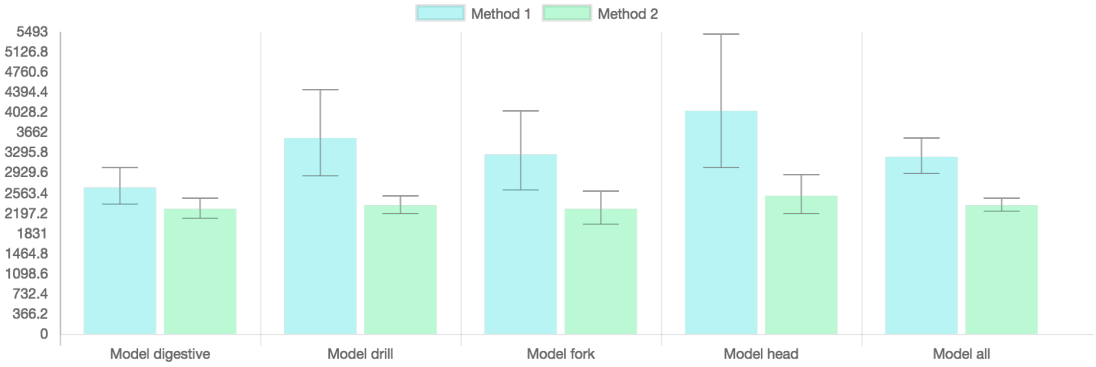


Figure 15. Boxplot for task times.

On this chart it is very clearly visible that the second method(*"Method 2"*) is more effective than the first(*"Method 1"*) because the participants spent on average less time to find a proper label. Another proof that the difference between theese two methods is statistically significant is that their Confidence Intervals do not overlap (It was described in Section 2.3.1). In addition, this theory (that the second method is more effective) confirms the number of mistakes made: the first method has 11 errors, the second method has none.

5.2. Check the accuracy

In addition to testing with real users, I compared the results of my calculator with other public online calculators and also with examples from the book *"Quantifying the User Experience"* [6]. Let us look at one example from this book for a *t*-confidence interval with the given data:

```
data = [90, 77.5, 72.5, 95, 62.5, 57.5, 100, 95, 95, 80, 82.5, 87.5]
```

Results from the book:

```
Mean: 82.9
Standard deviation: 13.5
Standard error: 3.9
Margin of error: 8.6
Confidence interval: 74.3 to 91.5
```

My results:

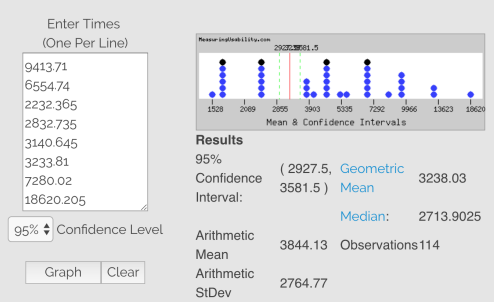
```
Mean: 82.917
Standard deviation: 13.519
Standard error: 3.903
Margin of error: 8.589
Confidence interval: 74.327 to 91.506
```

Since the **"Web service"** does not work in the same way a public calculator does, there is no possibility to load the given data using the UI. In order to check the data from the book, I used the interactive project mode in the command line, the command `rails c`, and a manual. This works because Ruby is a dynamic interpreted language.

Looking at the results it is clear that my values differ only in the precision, owing to the fact that my *CiCalculator* returns values rounded to a certain number of decimal places (default value of the precision is 3).

Further I compare my results with the online calculator "MeasuringU"[1] to calculate the Confidence Intervals for the task time. Now the sequence of actions is reversed. I take my raw data and load them into the calculator with the following settings:

- Confidence Level = 95%
- Log transformation = true



a) "MeasuringU" calculator results.

		Model all
Method 1	Geometric mean	3238.025
	Arithmetic mean	3844.134
	Lower	2927.667
	Upper	3581.285
	Median	2713.903

b) CI table of my results.

Figure 16. Examples of tooltips.

My results are again almost identical to the values obtained from the online calculator. For verification I used the values of the variable *"Time"* for the *"Method 1"* part of the experiment without filtering by the *"Model"* variable, this means that all calculations were made using all values.

Time is measured in milliseconds, and, therefore, the obtained results (means, CI bounds, etc.) are in milliseconds, too. In this regard, I consider the differences in `0.167 - 0.215` milliseconds insignificant.

[1] https://measuringu.com/time_intervals/

6. Conclusion

In this part, I would like to summarize the whole thesis. The aim was to create a system for collecting and evaluating data from user tests, which would facilitate a routine work for researchers.

In order to understand the needs of researchers, I conducted an analysis of the variations of the testing, existing designs of experiments with users (BSD and WSD), examined the pros and cons of these designs as well as several techniques that solve some of their problems. It was equally important to study the possible types of variables that the researcher is going to collect and analyze and also the methods of the statistical analysis depending on the properties of these variables. Between two methods of the statistical data processing, I chose the confidence intervals. Because they are better visualized and more suitable for comparative purposes. The analytical part brought clarity to this problem and prepared the basis for the next part - Design.

Further, the future system was described in greater detail, the ways of its usage together with the needs of the researchers were analyzed. Two research parts of this thesis helped to develop a list of requirements which made a basis for the system.

In the practical part (Implementation), I tried to describe the main points of the created system where the web service made the largest component of the whole system. Besides that, the implementation part included key code snippets providing valuable insights. In the end of the practical part, I conducted a short overview of the resulting web application.

The very last part was the Testing. In this part, I described the prototype of the tested application, the final component of the whole system, as well as the results of this testing. In addition, I compared the accuracy of the results with one of the available online calculators and with an example from the book *"Quantifying the User Experience"*.

Working on this thesis was definitely beneficial to me. I learned a lot of new and useful information. Developing the whole project independently was an interesting experience since one person must go through all the stages of the project development and also have a complete understanding of all its connecting parts. During the implementation, I was faced with tasks and problems that required non-standard solutions. Undoubtedly, everything above mentioned vastly improved my programming skills.

6.1. Future work

The **"Web service"** is a complete product with some additional functionality. But this does not mean that this project has nowhere to grow. There is an unlimited number of ideas and additions that could improve this product. For example, the "Web service" could send emails and give suggestions to the user. It could calculate p-value and effect size for each of the confidence interval.

Bibliography

[1] Elizabeth Goodman, Mike Kuniavsky, and Andrea Moed. *Observing the User Experience (Second Edition)*. Second Edition. Boston: Morgan Kaufmann, 2012. ISBN: 978-0-12-384869-7. URL: https://www.sciencedirect.com/science/article/pii/B9780123848697000012.

[2] Jeffrey Rubin and Dana Chisnell. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. Wiley Technical Communications Library. Wiley, 2008. ISBN: 978-0-47-038608-8. URL: http://ebookcentral.proquest.com/lib/cvut/detail.action?docID=343716.

[3] John J. Shaughnessy, Eugene B. Zechmeister, and Jeanne S. Zechmeister. *Research methods in psychology*. New York: McGraw-Hill, 2006. ISBN: 978-0-07-803518-0. URL: https://steladhima.files.wordpress.com/2014/03/john-j-shaughnessy-eugene-b-zechmeister-jeanne-s-zechmeister-research-methods-in-psychology-2012.pdf.

[4] Martyn Shuttleworth. *Counterbalanced Measures Design*. 2009. URL: https://explorable.com/counterbalanced-measures-design.

[5] *Hypothesis testing: Confidence intervals, t-tests, ANOVAs, and regression*. 2010. URL: https://my.vanderbilt.edu/joshuabazuin/files/2011/08/HOD-2990-Stats-21.pdf (visited on 01/03/2018).

[6] Jeff Sauro and James R. Lewis. *Quantifying the User Experience*. Boston: Morgan Kaufmann, 2012. ISBN: 978-0-12-384968-7. URL: https://www.sciencedirect.com/science/article/pii/B9780123849687000011.

[7] David M. Lane and Heidi Ziemer. *Distributions*. 2013. URL: http://onlinestatbook.com/2/introduction/distributions.html.

[8] I. Scott MacKenzie. *Human-Computer Interaction. An Empirical Research Perspective*. Elsevier Science, 2012. URL: https://ebookcentral.proquest.com/lib/cvut/detail.action?docID=1110719#.

[9] John Mueller. *Understanding SOAP and REST Basics And Differences*. 2013. URL: https://blog.smartbear.com/apis/understanding-soap-and-rest-basics/.

[10] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. Irvine: University of California, 2000.

[11] Rolf Timmermans. *Entity-Relationship Diagrams for Ruby on Rails*. 2010. URL: https://voormedia.github.io/rails-erd/ (visited on 12/14/2017).

[12] *Ruby on Rails*. URL: http://rubyonrails.org/ (visited on 12/22/2017).

[13] *Slim*. URL: http://slim-lang.com/ (visited on 12/22/2017).

[14] PostgreSQL. *PostgreSQL*. 2010. URL: https://www.postgresql.org/docs/9.4/static/datatype-json.html.

[15] *Chart.js*. URL: http://www.chartjs.org/ (visited on 01/05/2018).

[16] *Heroku platform*. URL: `https://www.heroku.com/platform` (visited on 12/28/2017).

[17] *Getting Started with Rails*. URL: `http://guides.rubyonrails.org/getting_started.html` (visited on 12/20/2017).

[18] *Rails Routing from the Outside In*. URL: `http://guides.rubyonrails.org/routing.html` (visited on 12/20/2017).

[19] *Promise*. URL: `https://developer.mozilla.org/cs/docs/Web/JavaScript/Reference/Global_Objects/Promise` (visited on 12/29/2017).

Appendix A.

T-distribution table example

α	0.05	0.025	0.01	0.005	0.0025	0.001	0.0005
df							
1	6.3138	12.7065	31.8193	63.6551	127.3447	318.4930	636.0450
2	2.9200	4.3026	6.9646	9.9247	14.0887	22.3276	31.5989
3	2.3534	3.1824	4.5407	5.8408	7.4534	10.2145	12.9242
4	2.1319	2.7764	3.7470	4.6041	5.5976	7.1732	8.6103
5	2.0150	2.5706	3.3650	4.0322	4.7734	5.8934	6.8688
6	1.9432	2.4469	3.1426	3.7074	4.3168	5.2076	5.9589
7	1.8946	2.3646	2.9980	3.4995	4.0294	4.7852	5.4079
8	1.8595	2.3060	2.8965	3.3554	3.8325	4.5008	5.0414
9	1.8331	2.2621	2.8214	3.2498	3.6896	4.2969	4.7809
10	1.8124	2.2282	2.7638	3.1693	3.5814	4.1437	4.5869
11	1.7959	2.2010	2.7181	3.1058	3.4966	4.0247	4.4369
12	1.7823	2.1788	2.6810	3.0545	3.4284	3.9296	4.3178
...							

Table 10. My caption

Appendix B.

F-distribution table example

DF_2/DF_1	1	2	3	4	5	6	7	8	9	10	...
1	39.86346	49.50000	53.59324	55.83296	57.24008	58.20442	58.90595	59.43898	59.85759	60.19498	
2	8.52632	9.00000	9.16179	9.24342	9.29263	9.32553	9.34908	9.36677	9.38054	9.39157	
3	5.53832	5.46238	5.39077	5.34264	5.30916	5.28473	5.26619	5.25167	5.24000	5.23041	
4	4.54477	4.32456	4.19086	4.10725	4.05058	4.00975	3.97897	3.95494	3.93567	3.91988	
5	4.06042	3.77972	3.61948	3.52020	3.45298	3.40451	3.36790	3.33928	3.31628	3.29740	
6	3.77595	3.46330	3.28876	3.18076	3.10751	3.05455	3.01446	2.98304	2.95774	2.93693	
7	3.58943	3.25744	3.07407	2.96053	2.88334	2.82739	2.78493	2.75158	2.72468	2.70251	
8	3.45792	3.11312	2.92380	2.80643	2.72645	2.66833	2.62413	2.58935	2.56124	2.53804	
9	3.36030	3.00645	2.81286	2.69268	2.61061	2.55086	2.50531	2.46941	2.44034	2.41632	
10	3.28502	2.92447	2.72767	2.60534	2.52164	2.46058	2.41397	2.37715	2.34731	2.32260	
...											

Table 11. My caption

Appendix C.

CD content

- CD
  - `source` — the directory of source codes
    - `server` — web service implementation sources
    - `library` — library sources
      - `transfer.js` — library in JavaScript
      - `transfer.java` — library in Java
    - `target_app` — target application implementation sources
  - `thesis` — the thesis text directory
    - `src` — the directory of LaTeX source codes of the thesis
    - `thesis.pdf` — the thesis text in PDF format