Concept and Prototypical Re-Implementation of a Web-Based Card Sorting Application with Responsive Design

Can Card Sorting be Done on a Smartphone?

By

KAILAASH BALACHANDRAN

Submitted to the Human-Computer Interaction Research Group in Partial Fulfillment of the Requirements for the Degree of Master of Science

Supervisor

PROF. DR. GERNOT SZWILLUS
and

PROF. DR. GITTA DOMIK

March 2016
The ubiquity of digital information has changed the way how users consume data on various devices. On the web, usability has become a necessary condition for survival of any application. One of the crucial factors in usability is that the information presented don’t get haphazard and chaotic as it could negate user experience. Taking an example of a web application, the most common problem encountered by users was in locating and navigating to desired content. To overcome this, usability experts emphasize on the need to adapt to a user-centered design approach to better understand user expectations. In this thesis, the popular design technique known as Card Sorting is discussed and reimplemented as a web application. The design of the application is made mobile-friendly in order to flex to multiple displays on various devices and the result exported can be fed to Casolysis or similar tools for evaluation.
declare that the work in this thesis was carried out in accordance with the requirements of the University’s Regulations and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate’s own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .......................................................... DATE: .............................................
# Table of Contents

| List of Figures | vii |
| List of Tables | ix |

1 Introduction  
1.1 Motivation .............................................. 2  
1.1.1 Domain Statement ........................................ 2  
1.1.2 Objective ............................................. 2  
1.2 Overview ............................................... 4  

2 Card Sorting  
2.1 Organizational Challenges .......................... 6  
2.1.1 How does Card Sorting help? ....................... 7  
2.2 Steps in a Card Sort ................................... 7  
2.3 The Variants ........................................... 8  
2.3.1 Open Card Sort ........................................ 9  
2.3.2 Closed Card Sort ...................................... 9  
2.3.3 Hybrid Card Sort ..................................... 10  
2.4 Sorting Structures .................................... 10  
2.4.1 Hierarchical Structure ............................. 10  
2.4.2 Flat Structure ......................................... 11  
2.4.3 Multiple Insert ...................................... 12  
2.5 Running an experiment .............................. 12  
2.5.1 Team-based Sort .................................... 12  
2.5.2 Individual Sort .................................... 13  
2.5.3 Further Analysis .................................... 13
## Table of Contents

3 Related Work 15

4 Design 19
4.1 The Home Page 19
4.2 The Dashboard 20
4.3 Experiment View 22

5 Technical Walkthrough 25
5.1 Technologies Used 25
5.1.1 Laravel 5.1 25
5.1.2 JQuery 29
5.1.3 MySQL 30
5.1.4 Other Libraries 31
5.2 Implementation 33
5.2.1 Database Structure 33
5.2.2 Plugins and Components 35

6 Testing 59
6.1 Test Plugin 59
6.2 SQL Map 60
6.3 Beta Test 61

7 Summary and Future Work 73
7.1 Summary 73
7.2 Future work 74
7.2.1 Image and Video Sort 74
7.2.2 Hybrid Sort 74
7.2.3 Evaluation of Hierarchical Sort 74
7.2.4 Integrating Casolysis 75

Bibliography 77
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 A Custom Organized Book Shelf</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Participants sorting cards into groups [1]</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Exploring and analyzing results [1]</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Use what is learned in the project</td>
<td>8</td>
</tr>
<tr>
<td>2.5 An Open Card Sort</td>
<td>9</td>
</tr>
<tr>
<td>2.6 A Closed Card Sort</td>
<td>10</td>
</tr>
<tr>
<td>2.7 A Hierarchical Tree Structure</td>
<td>11</td>
</tr>
<tr>
<td>2.8 Table with a Flat Structure</td>
<td>11</td>
</tr>
<tr>
<td>3.1 User Sort Session in UXSort</td>
<td>16</td>
</tr>
<tr>
<td>4.1 Homepage View</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Dashboard View</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Welcome message in Experiment View</td>
<td>22</td>
</tr>
<tr>
<td>4.4 Sort page in Experiment View</td>
<td>24</td>
</tr>
<tr>
<td>5.1 Popular PHP Frameworks</td>
<td>26</td>
</tr>
<tr>
<td>5.2 A MVC architecture</td>
<td>27</td>
</tr>
<tr>
<td>5.3 Project structure</td>
<td>28</td>
</tr>
<tr>
<td>5.4 JQuery DOM Wrapper</td>
<td>30</td>
</tr>
<tr>
<td>5.5 Database structure</td>
<td>34</td>
</tr>
<tr>
<td>5.6 Login View</td>
<td>36</td>
</tr>
<tr>
<td>5.7 Registration view with Recaptcha</td>
<td>36</td>
</tr>
<tr>
<td>5.8 Dashboard view</td>
<td>37</td>
</tr>
<tr>
<td>5.9 Create Experiment</td>
<td>38</td>
</tr>
<tr>
<td>5.10 Create Experiment</td>
<td>40</td>
</tr>
<tr>
<td>5.11 Edit Experiment</td>
<td>41</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.12</td>
<td>Show Analytics</td>
<td>42</td>
</tr>
<tr>
<td>5.13</td>
<td>The control flow that renders visitor map</td>
<td>43</td>
</tr>
<tr>
<td>5.14</td>
<td>Export Data</td>
<td>45</td>
</tr>
<tr>
<td>5.15</td>
<td>A Hierarchical Sort</td>
<td>45</td>
</tr>
<tr>
<td>5.16</td>
<td>Profile Settings</td>
<td>47</td>
</tr>
<tr>
<td>5.17</td>
<td>Change Password</td>
<td>47</td>
</tr>
<tr>
<td>5.18</td>
<td>Language Switcher</td>
<td>48</td>
</tr>
<tr>
<td>5.19</td>
<td>Manage Languages</td>
<td>49</td>
</tr>
<tr>
<td>5.20</td>
<td>Edit Language</td>
<td>49</td>
</tr>
<tr>
<td>5.21</td>
<td>Process diagram illustrating language translations</td>
<td>51</td>
</tr>
<tr>
<td>5.22</td>
<td>Welcome Message</td>
<td>52</td>
</tr>
<tr>
<td>5.23</td>
<td>Experiment Questionnaire</td>
<td>53</td>
</tr>
<tr>
<td>5.24</td>
<td>Experiment Instructions</td>
<td>55</td>
</tr>
<tr>
<td>5.25</td>
<td>Modal to add category for open sort experiments</td>
<td>55</td>
</tr>
<tr>
<td>5.26</td>
<td>Open sort experiments with cards being sorted</td>
<td>56</td>
</tr>
<tr>
<td>5.27</td>
<td>End Experiment Modal</td>
<td>56</td>
</tr>
<tr>
<td>5.28</td>
<td>Acknowledgement View</td>
<td>57</td>
</tr>
<tr>
<td>6.1</td>
<td>Penetration test being initiated in SQL Map</td>
<td>61</td>
</tr>
<tr>
<td>6.2</td>
<td>The Welcome Message</td>
<td>63</td>
</tr>
<tr>
<td>6.3</td>
<td>Questionnaire</td>
<td>64</td>
</tr>
<tr>
<td>6.4</td>
<td>Instruction Set</td>
<td>65</td>
</tr>
<tr>
<td>6.5</td>
<td>Modal dialog to add categories in an open sort experiment</td>
<td>66</td>
</tr>
<tr>
<td>6.6</td>
<td>Cards sorted into categories</td>
<td>66</td>
</tr>
<tr>
<td>6.7</td>
<td>Modal dialog to end an experiment</td>
<td>67</td>
</tr>
<tr>
<td>6.8</td>
<td>Google Form to report critical bugs</td>
<td>67</td>
</tr>
<tr>
<td>6.9</td>
<td>Acknowledgement View</td>
<td>68</td>
</tr>
<tr>
<td>6.10</td>
<td>Export sort data to .csv format</td>
<td>68</td>
</tr>
<tr>
<td>6.11</td>
<td>Importing sort results in .csv to Casolysis 2.0</td>
<td>68</td>
</tr>
<tr>
<td>6.12</td>
<td>Allocating columns in .csv to corresponding holders</td>
<td>69</td>
</tr>
<tr>
<td>6.13</td>
<td>The database view after import</td>
<td>69</td>
</tr>
<tr>
<td>6.14</td>
<td>Calculated distance matrix</td>
<td>70</td>
</tr>
<tr>
<td>6.15</td>
<td>Single link view</td>
<td>70</td>
</tr>
<tr>
<td>6.16</td>
<td>MDS Evaluation</td>
<td>71</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Laravel folder structure.</td>
</tr>
<tr>
<td>5.2</td>
<td>MySQL and Eloquent ORM Select Syntaxes.</td>
</tr>
<tr>
<td>5.3</td>
<td>User Management Components.</td>
</tr>
<tr>
<td>5.4</td>
<td>Dashboard Components.</td>
</tr>
<tr>
<td>5.5</td>
<td>Create Experiments Components.</td>
</tr>
<tr>
<td>5.6</td>
<td>Extended Manage Experiment Components.</td>
</tr>
<tr>
<td>5.7</td>
<td>Analytics Components.</td>
</tr>
<tr>
<td>5.8</td>
<td>Sample exported results with path structure</td>
</tr>
<tr>
<td>5.9</td>
<td>Export Data Components</td>
</tr>
<tr>
<td>5.10</td>
<td>Account Management Components</td>
</tr>
<tr>
<td>5.11</td>
<td>Components used in Language Handler</td>
</tr>
<tr>
<td>5.12</td>
<td>Components used in Experiment Handler</td>
</tr>
</tbody>
</table>
In recent times, the World Wide Web has created a significant impact on data accessibility through the Internet. Billions and trillions of websites and web-based applications are interconnected and are made available to users providing access to information and services. To make an application stand out, information has to be organized in a way that facilitate better navigation and accessibility. The success of any web-based application depends on how well structured the content is put and how much acceptable the layout is to the users in terms of its usability. Recently, usability has been receiving great attention, being recognized as a fundamental property for the success of Web Applications [2]. To understand better, let us evaluate the official website of the University of Paderborn. To get a quick glance on possible research opportunities in the university, a user would scan for the ‘Research’ page from the menu and would not expect the page to appear subcategorized under ‘Study Programs’. Simply put, nested information needs to be relevant that users should be able to comprehend and navigate accordingly. It is important that every item on the website is placed under semantic parent categories for better user experience. Organizing data semantically is a great challenge imposed on professional data experts to ensure better usability. But the results from data experts need not be the final call to achieve better usability, as the perception of users can be very different from that of a data expert. Users might classify the same data differently from experts and differences can arise between different groups of users, and the same users can create different taxonomies for different goals [3].

In approaching this endeavor to improve semantic categorization and data findability,
the user-centered technique to organize data in categories known as Card Sorting is re-implemented as a web application with much focus is given to enable mobile-friendliness and responsiveness to the user interface, thus making the application adaptable on various digital displays.

1.1 Motivation

1.1.1 Domain Statement

For any given application type, there are several up-front guidelines available that require careful consideration during the phase of design and development. On top of all given guidelines stands 'Usability'. Guidelines on usability include setting clear and precise goals for users, determining appropriate user requirements that also meet user expectations, thereby ensuring user intended goals are met in short time. From developing a simple static web page to a billion dollar software, usability matters much more than anything else in this digital age. To make a product likable to target audience, the initial and primary emphasis should be given to usability, rather than on aggregation of features. The current research suggests that the right approach to begin the construction of an application is to have many different people propose design solutions (i.e., parallel design), and then to follow up using an iterative design approach [4]. From the point of achieving a completed and satisfying design, features could always be reverse-engineered to meet the initial requirement specifications. Card sorting is one such user-centered technique that helps in the design and evaluation of information architecture of an application. It is a quick, inexpensive and reliable method, which serves as input to any information design process. Card sorting generates a clean overall structure of semantically connected information, navigation, suggestions and other taxonomies. In the method, data items known as Cards are distributed among participants and the goal for each participant is to sort the cards into groups.

1.1.2 Objective

In traditional card sort approach, the researcher is required to meet test participants individually, hand out a deck of cards and monitor how the cards are placed in groups. This manual process is tiresome and time-consuming as all steps need to be iterated all over again for each participant. Today, in this digital age, organizing the information architecture has become complex and researchers often feel the need to conduct experiments
on larger audiences. To cater such demand, the card sorting technique is digitalized. Web-Based Card Sorting experiments help in the study of usability especially when the number of participants is considerably large [5]. By creating a sorting experiment in the web, multiple users from different locations could participate with the requirement of needing only a browser to access. Such web-based experiments increase user participation exponentially and play a vital role in understanding the mental model of multiple users. Analyzing multiple and varied sorting results from a web-based experiment help data experts to implement better contextual navigation thus achieving better usability.

While conceding to the need of having web-based card sort experiments, it is important that the design of such applications be responsive on all devices. The advent of mobile devices has paved the way for the need to have mobile-friendly interfaces. Surveys taken in 2015 show that the usage of mobile devices has surpassed PC usage three-folds, thus making responsiveness of web applications a must-have for better usability. This change in the way people consume information on multiple devices means that the service that is offered has to be compatible on all platforms. In context to card sorting application, responsive sorting experiments could enhance user engagement as it can be performed on any device. The experiments are accessible with just a click on the link and the application flexes itself to fit the display of the device. Moreover, participants don’t feel the need to use desktop computers or laptops to perform sorts. They could also personalize the progress of the experiment by taking incremental steps to sort at their own pace without having to perform the entire experiment in one go. These web-based responsive experiments have a huge impact on user engagement, which in a way will lead to increase in the number of test participants thereby aggregating more results data for further analysis.

The idea of developing an all new web-based application is obsolete as multiple services are already available online. But what is not obsolete is the need for integration of better features and enhancements to the tool. The first version of Wecaso [6] is considered as the primary reference application for this thesis. On evaluating the missing enhancements and bugs in the tool such as responsive design, language inconsistencies and outdated versions of front end technologies (for example, sort feature is not supported on touch devices), it is evident that the tool required an upgrade in multiple aspects. Also, the trend in web technologies is not static. They evolve or change with time; so should the application that uses those technologies. The objective of this thesis is to re-implement a complete responsive version of Wecaso with modern design elements and enhanced user features that comply with current technology trends.
1.2 Overview

An outline of the remainder of the thesis is described here. Chapter 2 gives an introduction and motivation to the thesis domain and also briefly discusses the technique called card sorting. Then it’s followed by Chapter 3 which discusses the related work done in the area of the thesis research. After this, Chapter 4 gives an overview of the graphical user interface and its associated views. Every page links that are identified as the pages to go from the navigational panel of the application and related design decisions are covered in this chapter. Then follows Chapter 5, which extends chapter 4 to discuss in detail about the technical aspects of the application. Chapter 6 then proceeds to discuss various test components used such as the unit test plugins, SqlMap and beta test. Finally, the discussion is concluded in Chapter 7 which summarises the entire discussion and also points out about the possible future work which can be performed in this area.
Card sorting is all about organizing information into groups. The term 'information' need not be computing specific but can also mean any generic data. From articles on newspapers, recipes of a cookbook to personal contact list on mobiles, everything has to be organized into relevant groups. Taking the example of phone contacts, one could add a contact to 'Family list' and in the case of a news magazine, an article on Stock trends could be sorted under 'Economy' category. Though sorting data may sound straightforward, if disorganized, it could negate user experience and make them mundane. One other aspect to ponder while sorting data is to consider user preferences. Every user thought process is subjective and unique in their own way. To make information look appealing, it is crucial to get user's perspective on sorting data, analyze results and take the final call on making the information architecture balanced based on those results. Card Sort precisely helps in generating an optimal sort data from results. It is a technique where items to be sorted are written on cards and participants sort them into groups that makes the most sense to them.

On analyzing the card sort results based on supermarket groceries, the sort perspective of users could differ variably. Some may sort items by cuisine, by cooking method, by type (canned or snack), by color or texture or even by their frequency of usage and taste. Such an example experiment clearly demonstrates how the resulting user data could prove to be inconsistent with the original sort idea. A card sort is particularly useful to unveil new organizational schemes to items and also to understand the user for whom the system is built for.
2.1 Organizational Challenges

The primary intent of a sorting experiment is to organize information. But grouping data is something humans have been doing for centuries, from ancient textbooks to the latest fashion magazine, the idea of categorization is antique. Since this method is quite prevalent, it is not surprising that there exists more than one approach to sort items. Fig 2.1 shows an image of a bookshelf. The books can be organized in multiple ways, some by fiction and non-fiction, sub-genres, by author, by size, by numbering while books being read is placed at first or last or even by its ease of access. These different schemes mentioned make sense but the problem is when multiple people prefer different schemes. People think differently and so they may not favor one particular approach.

The case of digital items is different from that of physical items. In the previous example discussed, every book is a physical item and there is an advantage that it can be arranged in only one way at a time i.e they cannot be redundant or exist at two racks in the shelf. But in the digital world, data is easily prone for duplication and multiple organizational schemes can coexist which in a way could distort user experience. Change in usage context should also be taken into consideration. Taking the example of a university official web page, users may do different tasks at different times on the site. The usage context can range from finding contact information of a faculty member to ensuring application deadline for a degree course offered in that semester. Such changes
2.1.1 How does Card Sorting help?

Card Sorting can help you find what classification schemes are available for a set of information, whether people think in similar ways, and how usage context changes the way they describe groups [7]. Even if an existing scheme may seem to work for an application, a sorting experiment performed on the same, need not be consistent with the original scheme, and the results gathered may lead to question initial assumptions on categorization. The results produced from sorting does not tell the right scheme to go with, but will provide a deep insight on what the users are looking for, that way it is easier to cherry-pick the best of all results and to keep the final scheme balanced and suited to all users.

Card Sorting is best understood as a collaborative method to group information and also as a tool that helps to understand user perspectives. Moreover, the sorting experiment itself is relatively small and cheap, nevertheless, the results produced does make sense and could enhance user experience to a greater extent.

2.2 Steps in a Card Sort

The method to perform an experiment is fairly straight forward. A set of cards, with items to be sorted (often written on a piece of paper), are given to participants. Depending on the type of card sort (which will be discussed in following sections), participants are asked to pile cards into relevant groups. The core steps in a cart sort experiment are as follows.

1. Define the goal of experiment
2. Pin down the right experiment type
3. Choose content
4. Recruit participants
5. Run experiment and record all data [Fig 2.2]
6. Analyse results [Fig 2.3]
7. Actualize project [Fig 2.4]

### 2.3 The Variants

There are different ways to perform card sort that yields different results. Among the initial assessments to make before running the experiments is to choose the right type and it is crucial as it would affect the outcome of the project. The major variants, such as open, closed and hybrid are discussed in this chapter.
2.3. THE VARIANTS

2.3.1 Open Card Sort

In an open type [Fig 2.5], participants create labels for groups and sort cards in them accordingly. This method helps to get an overall picture of user’s expectation as the created custom labels unveil varied patterns in classification and grouping taxonomies. Also, depending on the goal of the experiment, participants could be asked to focus on one particular area. For example, participants could be asked to think about

- Target audience group
- The tasks they are likely to do and prioritize them

and sort cards in the way that suits them.

2.3.2 Closed Card Sort

In a closed card sort [Fig 2.6], participants are provided with predetermined group names and they slot items into relevant groups. When compared to open type, result from a closed card sort is less informative as participants are restricted from creating custom names, this hinders getting complete user perspectives. This variant is useful when

- There is an existing structure that works and sort results help to explore detailed item placements.
- Set of categories cannot be modified but items in it could be changed.
- Amount of change needed to an existing structure is small.
CHAPTER 2. CARD SORTING

2.3.3 Hybrid Card Sort

Hybrid sort is a mix of closed and open types. Here, names of categories are predetermined but participants can also create additional categories or modify existing category names. This approach is helpful when the partial structure exists and to allow participants to validate and complete the rest of the structure. This variant is not implemented in this thesis but listed out in section 7.2.2 as a possible future enhancement to the tool.

2.4 Sorting Structures

The structure of the card sort indicates how the information is organized during an experiment. This section discusses major structure topologies that could be used and depending on the chosen topology, the resultant sort structure is determined.

2.4.1 Hierarchical Structure

In a hierarchical structure as shown in Fig 2.7, multiple groups are nested under parent groups. Such structure is more comprehensive intuitively and is common as the file system of any operating system resembles hierarchy structure, also the site map of websites on the Internet has this nested structure. This approach works great in a card sort as participants create bundles of subgroups and nest semantic information, thereby expediting better analysis of user expectations.
2.4. SORTING STRUCTURES

2.4.2 Flat Structure

Flat structure, also known as database structure, is a simple bucket-like structure where cards are stacked on top of each one. There is no middle level or sub-level groups. Flat structure is best suited when the content to be sorted belongs to one context. For example, the architecture of a conference meeting could contain categories of title, presenter and venue. In such an example, cards corresponding to venue name can be sorted inside ‘venue’ and cards with the title can be sorted to the relevant category respectively. A flat structure is illustrated in Fig 2.8
2.4.3 Multiple Insert

Though not a direct structural variant, the multiple insert describes the possibility of having items in more than one category. There are instances where an item can have 1:m relationship with other categories. Considering the following example where the movie names are taken as cards and the genres it can belong to as categories, participants can get unsure as a movie can belong to multiple genres. In such uncertain cases, participants duplicate items and place them in as many number of categories they think the card fits in.

2.5 Running an experiment

This section discusses different ways to run a card sorting experiment. Choosing the right type directly affects the quality of data and also the participant involvement during the sort.

2.5.1 Team-based Sort

In a team based experiment, a group of participants work together sorting cards into groups. This is the most promising approach to get participants perspective as they discuss and argue while performing the sort. In a team, every move to sort a card is argued upon with fellow members and the resultant data represents the perspective of all participants in the team. Moreover, the discussions among team members during the sort contain deep insights than the actual sort results. The down-side of this approach is that participants might tend to make compromises rather than working on their differences. In a team, one can expect members to have different personality traits, these traits range from being submissive to assertive in nature. Both these mentioned traits have adverse effect on the sorted results. Considering cases of teams with dominant participants, the dominant member might become assertive by taking the lead in the decision-making process and the player who is submissive gets a passive role to play. This could monopolize the team’s viewpoint and the team-represented results might become one-sided perspective of the dominant player. In connection to this thesis, the application designed does not accommodate team-based sort as Wecaso is not a computer-supported cooperative work (CSCW) tool.
2.5.2 Individual Sort

Individual sorts has the benefit that of gathering large number of responses and it is much easier to coordinate individuals than teams. The disadvantages of individual sorts is that the sort lacks rich discussions that happen in a team sort. Also, if the experiment has to done face to face, the entire process becomes time-consuming. In individual sorts, the experiment can be done with a sorting software or using physical set of cards.

2.5.3 Further Analysis

For a physical sort, it is important that the sort data is recorded as soon as possible. If a card sorting tool is used, recording of data is handled by the application itself and evaluation can also be conducted depending on the features offered in the application. Once the resultant data is generated from the tool or manual listing on a spreadsheet, the results are then evaluated based on exploratory or statistical analysis. These analyses identifies key patterns in the data and derives useful insights for the project. In the context of this thesis, the sort data can be exported to any spreadsheet format and further evaluation is carried out by inputting the file to Casolysis. The various software based tools available are discussed in the Chapter 3.
In this faster paced world, usability experts work on complex data architectures and feel the need to reach out to more participants to perform sorting experiments and gather maximum data as possible in shorter time. Traditional card sorting technique is conducted using a physical set of cards as individual sorts for each participant or as a team-based sort. This approach is time-consuming and low-tech. There has been a lot of related work in the area of digitalizing card sorting technique. These software-based sorting experiment looks quite similar to physical sort - the cards that look like real cards can be moved around into categories. The card sorting tools that are available can be classified into two types; system-based and web-based applications. The tool UXSort [8] is a system application that is installed locally on a computer and runs only on one system node for the entire set of gathering experimental data from participants. This means each participant should be given access to the computer to perform experiments and the process needs to be repeated for all participants. Though UXSort supports multiple level deep hierarchical sorting of cards and provides cluster analysis on results, as the tool is system based, the steps needed to perform experiment are the same as performing a sort with physical cards.

On the other hand, sorting experiments created on web-based platforms can be accessed by anyone with just a click on the link. Some of the popular web based tools available are OptimalSort [9], Simple Card Sort [10], and User Zoom [11]. All of the mentioned sorting tools have integrated analysis mechanisms that provide insightful card sort result analysis with comprehensive similarity matrix and dendrograms. In
spite of their user friendliness and embodied analysis mechanisms, it is important to note that the available web-based tools don’t support hierarchical sorting and multiple insertions of cards.

Consider an example experiment to create navigational menu for a news magazine and by assuming a magazine can have varied sections and subsections, it is expected that the number of cards can be higher. In such cases, the feature to nest cards creating categories and sub categories is necessary to create semantically relevant navigational menu for the portal. Single level or simple stack based sorting may work well with simple applications but with complex web portals such as news listings or classifieds, a hierarchical navigational structure is absolutely necessary given the size of digital content they have. Also, during the sort, participants could relate a card to more than one category. In such scenarios, a feature like multiple insert helps users to duplicate cards and place them in as many number of categories they think the card might fit in. This way, cards in a sort can have 1:m relationship with categories.

As the world wide web has become ubiquitous, it is lack of feature that all of the available sorting tools are restricted for one language support only. In order to serve a web-based platform to a wider audience, it is crucial that the application supports multiple languages. The growth of digital devices is also at large. The advent of mobile
devices has paved the way for the need to have mobile-friendly interfaces. Survey taken in 2014 shows the usage of mobile devices has surpassed PC usage three-folds, thus making responsiveness of web applications a must-have for better usability. This change in the way people consume information on multiple devices means that the service that is offered has to be compatible on all devices.

Now that the setbacks and missing enhancements with existing tools are discussed, it is evident that the new card sorting platform need to close gaps with possible improvements. The first version of Wecaso [6] was developed by Andreas Vdovkin as a part of bachelor thesis some years ago. As with any other software, WeCaSo too required an upgrade and redesign as to meet with the mentioned enhancements and modern usability requirements. The contribution of this thesis is a complete responsive card sorting application with support to hierarchical nesting, multiple insert of cards and support for multiple languages.
there are two major approaches followed in the design of web-based systems: responsive design and mobile templates. Responsive design requires you to have one website that is coded to adapt to all screen sizes, no matter what the device the website’s being displayed on. In contrast, a mobile template is a complete separate entity requiring you to have a second or more template designs for each device type. Mobile templates are also built for each specific site, not per screen size. The disadvantage with mobile templates is that the entire application needs to be redesigned to suit multiple devices. To avoid such redesign cycles, developers opt to go the responsive way. By considering the current design trends, mobile templates are no more a viable option. Hence, the design of the application incorporates material responsive elements to provide a modern look to the user interface. The graphical user interface incorporates Bootstrap [12] and also uses other modern design elements to enhance user experience and mobile friendliness. The structure of the application is divided into three major views as discussed below.

4.1 The Home Page

The home page, as seen in fig 4.1, serves as the default index root of the application where a visitor arrives at. The information contained in the homepage include generic details about card sorting and the application. The page also acts as the parent view that holds links to access all other sections of the application.
4.2 The Dashboard

The dashboard view, as in fig 4.2 is the default site view a user sees after logging in. When considering to deliver modern look to the application, three specific needs were recognized i) to give a positive visual appeal to users, ii) to enable user friendliness, and iii) aggregation of add-on features such as analytics, map view etc. All of the three aspects are particularly concentrated in the dashboard view as it acts as the control panel for users to create and manage experiments. The static menu panel on the left is the primary navigation menu that renders multiple views and sub views for every intended function. The main content area displays custom information of the user’s activity and other experiment stats such as total experiments created by the user, total visitors and reach, map view and quick links for easy access to other areas of the site. An
4.2. THE DASHBOARD

OVERVIEW OF THE MAIN PAGE LINKS WHICH ARE IDENTIFIED AS THE PAGES TO GO FROM THE ADMIN PANEL IS DISCUSSED BELOW.

- **Create Experiment:** The form in this view inputs needed data to create an experiment. The URL of the experiment needs to be unique and to facilitate better error handling in naming URLs, the text entered in the URL field is checked dynamically while being typed by the user. In case of non-availability of the URL, the corresponding error message is displayed.

- **Manage Experiments:** The option is split into open and closed views depending on the type of the experiment. This view allows all experiment data such as project details, items, categories and questionnaire to be read, edited or deleted.

- **Analytics** For any active experiment, given that the experiment has at least 1 test participant, statistics report is generated dynamically based on current data from the database. Analytics is a statistical add-on to keep track of the progress of an experiment.

- **Export** The feature offers option to export data to various spreadsheet formats including .csv, .xls and .xslt. The form to export data takes the format as input from the drop-down menu and also the content to be exported from database.

- **Language Switcher** The language switcher from top menu lists all available language translations in the application. When user picks a language, the selected language is set as default language for the user. User can toggle between languages.

Figure 4.2. Dashboard View.
CHAPTER 4. DESIGN

4.3 Experiment View

The public URL created at the backend identifies the experiment and the respective experiment view is generated based on the experiment details. Care has been taken to make sure the experiment is responsive on all devices and the URL is readable. Responsiveness of the experiment, particularly sort view, where the sort elements need to be touch friendly, is thoroughly implemented and tested. Also, the structure of the URL is standardized for all types of experiment as seen in the format below:

http://wecaso.de/experiment/[experiment-name]

Here, the parameter [experiment-name] is replaced with the actual URL of the experiment and this helps the application to identify the experiment data from the database. The flow of the experiment follows the wizard structure and the order of the flow correspond to the order of the accordion menu as seen in 'Manage Experiments' page at the backend. The order of the flow of an experiment is discussed below.
1. **Welcome message** The welcome message set at the backend serves as the homepage for the experiment [fig 4.3]

2. **Questionnaire** The second view of the experiment holds the questionnaire. Based on the questions set at the dashboard, question items are rendered in this view. The application supports three answer types; single choice, multiple choice and text field.

3. **Instructions** Similar to welcome message, the data set as instructions for the experiment is rendered in this view.

4. **Sort view** The sort view, as seen in fig 4.4, is divided into open and closed subviews. Depending on the type of experiment, the respective view is rendered. All items or cards are listed on the left. The handlers allow the item to be moved around and placed into other categories. In open sort types, the Bootstrap modal allows users to create categories during the experiment run-time. Options to record user email and comments are displayed when the button to end experiment is clicked.

5. **Acknowledgement** This final view renders the thank you message as set for the experiment.

Now that the overview on the design structure of the application is discussed, the next chapter focuses on the technical aspects of every implemented feature, the frameworks used and respective dependency adaptions to the application.
**Figure 4.4.** Sort page in Experiment View
The variants of open and closed card sorting are implemented as a web application with options of multiple insert and hierarchical sort. This chapter discusses in-depth about technical aspects of the application such as the system architecture, frameworks used, the dependencies and components involved.

5.1 Technologies Used

The application has taken the full advantage of some of the latest standard web technologies as of 2016. The combination of all these frameworks gave the app a brand new cutting edge look which is throughout consistent, provides multi-language provision and support for mobile devices. Some other benefits also include a single page design and only parts of this page is getting changes as per the data requested from the user. Moreover, the floating windows are also persistent even after the page is getting refreshed. Some of these technologies are discussed in the following sections.

5.1.1 Laravel 5.1

Laravel [13] is a full-stack PHP framework as it handles everything right from page generation to database management. It comes with its own command-line utility called Artisan that manages the entire project environment. From generating skeleton code to database stubs, this command-line utility comes in handy for most functions. Laravel is
one of the popular web frameworks used, Fig 5.1 shows the popularity graph as of 2015. Some of the top-notch features that Laravel offers are discussed below.

5.1.1.1 Standard Convention

Laravel imposes strict constraints on how the application is structured. Surprisingly, such restricted rules have its own benefits and makes the app development easier. The difference with other vertically integrated environments is that Laravel gives less precedence to configuration than to convention. With Java, Python and other PHP frameworks, XML config generation is required whereas Laravel requires none.

5.1.1.2 Model-View-Controller

Model-view-controller (MVC) is a software architectural pattern that separates business logic from presentation logic i.e the views. Laravel is built on the MVC pattern providing
5.1. TECHNOLOGIES USED

Figure 5.2. A MVC architecture

a clear divide between the database, application logic and HTML front-end. Fig 5.2 depicts the top-level control flow of a typical Laravel application.

5.1.1.3 The Project Structure

Laravel’s aversion to custom configuration files gives the possibility to have a consistent project and code structure across multiple apps. The framework provides folders where custom logic needs to be placed at designated places. The structure of a laravel project is shown in [Fig 5.3]. The basic rundown of some of the top level folders are discussed in the table below.

<table>
<thead>
<tr>
<th>Folder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/app/</td>
<td>Contains routes, controllers and models. Most of the code logic is written inside the folder.</td>
</tr>
<tr>
<td>/vendor/</td>
<td>The designated folder for all 3-rd party dependencies and other plugins. Plugins can be managed using composer.</td>
</tr>
<tr>
<td>/database/</td>
<td>Contains migrations to create data models and seeds. Migrations is created using Artisan.</td>
</tr>
<tr>
<td>/config/</td>
<td>Contains the parent configuration files of the application and also its vendors that are added in the cascade.</td>
</tr>
</tbody>
</table>
5.1.1.4 Template Engine

To avoid inconsistencies during page-loads, Laravel comes with its own template engine called Blade. To ease front-end development, Blade supports template inheritance where a master layout is defined that has elements common to all pages and is called to render subpages. All Blade views are compiled into plain PHP code and cached until they are modified, meaning Blade adds essentially zero overhead to your application. Blade view files use the .blade.php file extension and are typically stored in the resources/views directory [13].

5.1.1.5 Data Models

Database migrations are integrated within the project environment using Artisan. The migrated scheme generates skeleton data for the database and the corresponding database is used to create the model. Laravel uses Object-Relational-Mapping interface
called Eloquent. All of the queries in this application uses Eloquent ORM. The complete database structure of this application is discussed in Section 5.1.3.

5.1.2 JQuery

JQuery [15] is a powerful library built on top of Javascript. Ever since its debut in 2006, JQuery has become the to-go framework in the area of front-end development. It permits easy manipulation of HTML elements, thus simplifies scripting that involves complex changes to DOM (Document Object Model). Some of the reasons for choosing JQuery are discussed below.

5.1.2.1 Abstraction

On top of its simplicity, JQuery makes the code interoperable on different browsers. The manipulation and navigation of a HTML page are tailored automatically to suit the browser engine that renders the page. For example, Firefox renders whitespaces in the DOM whereas IE6 does the opposite. In such cases, developers had to recode the same functions for every browser but, with JQuery, DOM manipulation is automated.

5.1.2.2 AJAX

AJAX (Asynchronous Javascript and XML) is a paradigm used to send and receive data without having to refresh or load the page. JQuery handles Ajax efficiently and provides many convenient event handlers to handle data. In context to this application, JQuery AJAX is used throughout, especially in the experiment view where the sorting is performed. Though XML could be used as the standard data format for AJAX, this application passes data in JSON (Javascript Object Notation), the standard format used in modern web applications. One of the reasons for the success of JSON over XML is because the former is simpler to serialize and takes up less space than the latter. Also, many libraries and quick functions exist to decode and encode JSON objects for other programming languages.

5.1.2.3 Event Selectors

JQuery doesn’t require context switching to refer to the same elements in the DOM, particularly CSS. Unlike other javascript libraries, JQuery keeps CSS selectors in the forefront. It implemented CSS3 much before it was incorporated in popular browsers. This meant all of the JQuery events could be applied to all front-end elements that
require spaghetti code otherwise. To alter DOM elements, JQuery wraps them into instances of JQuery object and provides access to those objects as javascript elements. Thus, DOM is changed without altering the core DOM features. Fig 5.4 illustrates a basic Jquery DOM wrapper with associated functions.

5.1.3 MySQL

The application uses MySQL [16] as the database system. As a database with relational model, the data is stored in tables and optimized for speed. While choosing the best open-source databases for this project, MySQL was preferred as it was found to match the requirements of this application. Some of the criteria that favored the decision are discussed below.

5.1.3.1 Relational Property

In relational databases, the data is stored only once. Connected data is split into relational tables and stored individually to avoid duplication. Powerful queries help to concatenate multiple data and show refined results. As the data is split into tables, some table data can be made confidential. For example, in this application, the data table 'role' is made confidential that it doesn't approve read or write access to it.
5.1.3.2 Scalability

For any web application, the database that serves users need to be robust and handle extreme load of complex queries. One of the exceptional property of MySQL is its performance and scalability. A unique storage engine architecture allows database professionals to configure the MySQL database server specifically for particular applications, with the end result being amazing performance results [16].

5.1.3.3 Reliability

MySQL has been battle-tested by QA testers to have a check on its trust-worthiness in a wide variety of environments and the result generated exhibits MySQL's high fault tolerance and up-time, rapid restart, dynamic adaptation of varying load and easy maintenance.

5.1.3.4 Security

Because the data is stored in relational tables, MySQL offers remarkable security features for data protection. A table can be marked confidential and hidden from users, ensuring only authorized users are granted access to it, that way, users only see things that they should.

5.1.3.5 Ease of use

Though PostGRE offers similar features as MySQL, but in its ability for ease of use and maintenance, MySQL wins. It offers a comprehensive quick-start ability, self-management feature once configured and ease to use tools for maintenance. Administrator tools such as MySQL Workbench and phpMyAdmin facilitate trouble-free database management.

5.1.4 Other Libraries

In today’s web development paradigm, it is recommended to use a range of different frameworks to make the app more intuitive, modern and flexible on multiple devices. Depending on the requirements, the best frameworks were cherry-picked for this application and are discussed in the section below.
CHAPTER 5. TECHNICAL WALKTHROUGH

5.1.4.1 Bootstrap

Developed by Twitter Inc, Bootstrap [12] is the most popular open-source framework used by web designers. It is elegant, intuitive and a powerful kit for developers to create cross-browse web applications. It offers support to many other libraries such as JQuery UI to provide good looking interfaces and UI components.

5.1.4.2 GeoIP

GeoIP [17] is a laravel package that records the IP address of the user performing the experiment. It takes the location of the user’s ISP to determines the location, then computes the latitude and longitude based on the location and feeds the data to JVector Map.

5.1.4.3 JVectorMap

JVectorMap [18] is an efficient alternative to Google Maps. Built on Javascript, unlike GMaps, this library does not require special API Key privileges. Data from GeoIP is fed to JVectorMap to add markers to the map view.

5.1.4.4 MetisMenu

MetisMenu [19] is a javascript library used to create the menu panel in the dashboard view. The library is auto-responsive and also provides interface for collapse and expand.

5.1.4.5 Nestable

This application uses Nestable.js [20], a Jquery library, to sort cards and create categories in the experiment view. It comes with features to drag and drop, create nestable lists and touch compatibility.

5.1.4.6 AnimateCSS and Livicons

Livicons [21] and Animate CSS are cross-browser libraries of CSS animations and vector icons. This package is retina perfect, means the elements are presented perfect on high-resolution displays.
5.2 Implementation

As previous chapters gave a top-level overview of the application and the technologies used, this chapter discusses in detail about the internal technical architecture and its components involved. The main components of the database architecture are described in the following section. The next few sections are covers the code logic and flow control of each component.

5.2.1 Database Structure

The application uses MySQL as the primary database and implements Laravel’s Eloquent ORM to connect to the database. The recommended laravel convention is followed throughout the application right from rule migration to querying the database. Eloquent ORM comes with a powerful syntax to connect to model and retrieve data. Table 5.2 shows the difference between a simple MySQL query and an eloquent query to select data from a model. The complete structure and relationship schema of the database is shown in Fig 5.5. The models available in the framework for each schema are discussed below.

<table>
<thead>
<tr>
<th>Simple Query</th>
<th>Eloquent ORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select * from testuser where user = 50</td>
<td>Testuser::where('user','=',50)</td>
</tr>
<tr>
<td></td>
<td>&gt;get()</td>
</tr>
</tbody>
</table>

Table 5.2: MySQL and Eloquent ORM Select Syntaxes.

User

This auto generated model from migration contains all user data that includes name, password, email, biography, profile image and language.

Role

A user can belong to only one of the two roles; basic and admin. Users with admin roles are given special privileges to add or remove experiments and handle languages.

Experiment

Experiment model contain data of all experiments created by users. Experiments are associated with users using the user id as primary key. Items and categories are concatenated and stored as string in their respective rows.
Figure 5.5. Database structure.
Questions

Questions and predefined answers set by users while creating an experiment are stored in this model. Experiment id in the model identifies the experiment to which a set of questions belong to.

TestUsers

This model contains the sorted results of an experiment. Answers to questions, comments, email and all categories and cards that have been sorted are stored in this model.

Language

Contains all available languages and the associated translations. Users with the role of admin have access to manage languages.

5.2.2 Plugins and Components

This section discusses extensively about different features in the application, extends it to the framework layer with detailed description about the functionality. A comprehensive approach to cover the control flow and components of the Model-View-Controller is explained in a table structure for each feature.

5.2.2.1 User Management

The login and registration components use Laravel’s inbuilt authentication system for user management. The remember me option stores user session in Sessions model and is made available to the application for the next 120 minutes, after which the session will expires. The registration component validates and allows only one occurrence of given email address, thereby avoiding redundancy. To control spam-bots from abusing user registration system, it is essential to have a check during registrations, hence the popular Google Recaptcha is used to validate if the entry is provided by a real user. Recaptcha requires a public and private key for the domain, thus such validation is made unique for the application associated to the domain. Passwords are hashed using bcrypt, the popular key derivation encryption algorithm.

Furthermore, a provision to change password is provided in the dashboard view that verifies the old password given against the one stored in the database. The class Validator::() performs all validation checks for login, registration and forgot password components.
Adaptations to Wecaso:

<table>
<thead>
<tr>
<th>Model</th>
<th>User, Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Login, Register, Profile</td>
</tr>
<tr>
<td>Controller</td>
<td>LoginController@getAuth, UsersController@store</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Google ReCaptcha, Auth::(), Validator::()</td>
</tr>
<tr>
<td>Access</td>
<td>Public</td>
</tr>
</tbody>
</table>

Table 5.3: User Management Components.

### 5.2.2.2 Dashboard

The landing page after user login renders multiple subviews and menu links to navigate throughout the website. The static menu bar on the left and the top menu, i.e language selector and account settings drop-down menu, extends master layout from views. Addi-
5.2. IMPLEMENTATION

Additionally, the dashboard view displays custom information on user's activity, map view and quick links for easy access to other areas of the site.

Note that the information shown in the dashboard view is the total aggregation of data of all experiments created by the user. Individual data on statistics of each experiment is available in analytics view which is discussed in section 2.2. The map area in this view uses GeoIP [section 5.1.4.2] to discover the latitude and longitude of test participants and feeds the same to JVectorMap [section 5.1.4.3] to render location markers in the map.

**Adaptations to Wecaso:**

<table>
<thead>
<tr>
<th>Model</th>
<th>User,Experiment,TestUsers,Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Dashboard</td>
</tr>
<tr>
<td>Controller</td>
<td>UsersController@showDashboard</td>
</tr>
<tr>
<td>Dependencies</td>
<td>GeoIP,Auth::():,JVectorMap,Datatables</td>
</tr>
<tr>
<td>Access</td>
<td>Authentication required</td>
</tr>
</tbody>
</table>

Table 5.4: Dashboard Components.

### 5.2.2.3 Create Experiment

The form in this view inputs primary details needed to create an experiment. AJAX based precondition validates uniqueness of the project URL to avoid redundancy. Jquery catches the text as entered in the field, sends an AJAX request to the server in order to check if the entered URL is available and is unique. If yes, the text 'OK' is displayed or else 'URL Exists' is printed. The language field in the form is in sync with the master language controller of the application.
Any changes to the language handler made by admin such as publishing or republishing is also updated in Language field select menu. After an experiment is created, the user is then redirected to manage experiments view wherein further details can be added to the experiment.

**Adaptations to Wecaso:**

<table>
<thead>
<tr>
<th>Model</th>
<th>User,Experiment,Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>CreateEx</td>
</tr>
<tr>
<td>Controller</td>
<td>CreateExController@index, CreateExController@store</td>
</tr>
<tr>
<td>Access</td>
<td>Authentication required</td>
</tr>
</tbody>
</table>

Table 5.5: Create Experiments Components.

**5.2.2.4 Manage Experiments**

Manage Experiments holds two views i.e the table and edit View. Created experiments (open and closed) are added to table view that displays options to edit experiment details and delete experiment.
Table view uses DataTables, a JQuery table library to add advanced interaction controls to table structure. The table view is split for both open and closed experiments which are filtered based on the user selection from menu panel. Fig 5.10 shows the table view for open sorting experiments. Eloquent’s destroy() and delete() functions soft-deletes selected experiment data in the order of its relationship precedence. The code snippet below shows the experiment destroy() controller method in 'ExperimentController' that handles deletion of an experiment. The deletion is performed carefully that only the owner of the experiment or the admin are given access for deletion.

```php
public function destroy(Request $request)
{
    $ex = Experiments::find(Input::get('eid'));
    if($ex->fk_user==$user->id || Auth::user()->fk_roles==1 )
    {
        $test_users = Results::where('fk_experiments','=',$exid)->get();
        if($test_users->count()>0)
        {
            foreach($test_users as $eachuser)
            {
                $eachuser->delete();
            }
        }
        $questions = Questions::where('fk_experiments','=',$exid)->get();
        if($questions->count()>0)
        {
            foreach($questions as $eachquestion)
            {
                $eachquestion->delete();
            }
        }
        $ex->delete();
    }
}
```
The Bootstrap modal dialog handles the delete confirmation pop-up when the delete button is clicked. When a delete request is initiated for an experiment, the authenticity of the request if first checked, i.e the access control of the user that initiated the request is verified, then the associated experiment data is deleted in the following order of table relationships that holds the primary key.

1. All TestUser::() data that holds sorting results and answers of participants are deleted.

2. Questionary data, if set, is deleted for the experiment.

3. Lastly, the experiment itself, that includes the name, url, items and categories, is deleted.

4. AJAX returns delete()->success flag to update table view.

The edit view of an experiment renders two forms that hold project details and experiment settings. Fig 5.11 shows the screen shots of two forms from the same view. Each form data is handled by different controllers and routes, except that, the view is shared.

The project details form has the accordion structure to make the flow of experiment order comprehensive for users. It uses JQuery AJAX to post data to controller and expect success flag when the data is saved to database, after which the view is updated. Each items and categories (for closed sorting experiment) are separated by semi-colon(;) and is stored as a single string entity at the database level. To ease data entry operation, users are allowed to enter one text per line, ending each text with a semi-colon. All created
white-spaces are removed at the server side the data is saved to database. Experiment settings form in edit view replicates the Create Experiment view as discussed in Section 5.2.2.3

**Adaptations to Wecaso:**

<table>
<thead>
<tr>
<th>Model</th>
<th>User, Experiment, Languages, TestUsers, Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Accordion, Extable</td>
</tr>
<tr>
<td>Controller</td>
<td>ExController@resource</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Accordion.js, DataTables.js, DatePicker.js</td>
</tr>
<tr>
<td>Access</td>
<td>Authentication required</td>
</tr>
</tbody>
</table>

Table 5.6: Extended Manage Experiment Components.
5.2.2.5 Analytics

For any active experiment, given that the experiment has at least 1 test participant, statistics report is generated dynamically based on current data from database. Analytics is a statistical add-on to keep track of the progress of an experiment. The page when loaded renders two sub views which are discussed below.

Visitor Map
Map view calls the static JVectorMap [Section:5.1.4.3] function to generate the map with markers. These markers correspond to the location of the test participants based on the latitude and longitude data provided by GeoIP [Section:5.1.4.2]. GeoIP identifies the IP address of a test participant and then locates the latitude and longitude of the location based on IP. This data is stored in TestUsers model. When the map view is rendered, JVectorMap fetches all location data of test participants from the model and populates the map view. Fig 5.13 illustrates the control flow of components involved to display the map view.

Statistics
Statistics generates a top-level overview report on the performance of the experiment. The report calculates the following parameters as discussed below.

- **Test Users**: The data corresponds to the total number of test participants who has performed the experiment.
- **Visitor Count**: When a user perform an experiment, the user is added to the visitor list, irrespective of whether he/she has completed the experiment. The
major difference between Test Users and Visitor Count calculations is that the former includes only those users who completed the experiment, whereas, the latter includes both values.

- **Total Reach**: The calculation on the average of users who performed the experiment versus visitor count, provides an interesting insight on the reach the experiment has got.

  \[
  \text{Visitor Count} - \text{Test Users} = \text{Non-Submitted User Data}
  \]

- **Sorting Complete**: The data shows the number of test participants who has sorted all cards of the experiment.

- **Comments Recorded**: Filters the test participant count based on comments, if given.

- **Email Recorded**: Shows the count of test participants with email addresses, if given.

**Adaptations to Wecaso:**

<table>
<thead>
<tr>
<th>Model</th>
<th>User, Experiment, Languages, TestUsers, Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Analytics, Extable</td>
</tr>
<tr>
<td>Controller</td>
<td>AnalyticsController@resource</td>
</tr>
<tr>
<td>Dependencies</td>
<td>JVectorMap.js, DataTables.js, GeoIP</td>
</tr>
<tr>
<td>Access</td>
<td>Authentication required</td>
</tr>
</tbody>
</table>

Table 5.7: Analytics Components.
5.2.2.6 Export

The application utilizes Laravel Excel package, the eloquent way to export data from database with the power of PHPExcel library. The library offers options to export data to various spreadsheet formats including .csv, .xls and .xslt. The form to export data takes the format as input from the drop-down menu and also the content to be exported from database.

The architecture of the application, specifically at the database level, is structured in a way considering scalability in mind. The sorted results that contain huge chunks of data, such as multiple cards and categories, are consolidated and stored as strings, rather objects. The aggregated sorted results are first stored in arrays, which are later encoded to JSON and stored as a single entry in the database. This approach saves space and doesn’t complicate entries to the database. At the export point, the stored JSON data is fetched from TestUsers::model() and decoded with json_decode() function, which de-serializes the data to its original format. Once decoded, Laravel Excel package loads data and exports file in the requested format. The available column headers are discussed below.

- **user_id**: The data corresponds to the test participant id in TestUsers::model().
- **email**: Email as provided by the test participant. Default is NULL or empty.
- **card_label**: The card that was sorted into a category.
- **card_id**: For each card item, a unique integer id is generated. If two card items are identical, the card id is shared with two copies.
- **category_label**: Denotes the category to which the card is sorted into.
- **category_id**: Similar to card id’s, numerical id is generated for each unique category name. If redundancy is found, the id of the original category is assigned for this new category.
- **comments**: Comments are shown if given by the participant.
- **complete**: This column can take only two values; 'yes' or 'no'. The value 'yes' is generated if all cards were sorted by the user, otherwise the value is 'no'.
- **time**: Indicates the time when the experiment was performed.
• **q\_responses:** The count of this column with question responses increments dynamically depending on the number questions asked in the questionnaire. For example, if the questionnaire of an experiment contains three questions, then three columns with numbered labels, three in this case, are generated with user responses.

For experiments where hierarchical sorting is enabled, the hierarchy of sorted cards is displayed as a path. For example, Fig 5.15 shows a screen shot of a hierarchical sort being
performed. The structure of the exported results takes the path structure as illustrated in the table below.

<table>
<thead>
<tr>
<th>card_label</th>
<th>category_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technische Unterstützung / Studienabbruch / Praktikum / Vom Bachelor zum Master / Studienstart</td>
<td>Studium</td>
</tr>
<tr>
<td>Prüfungen</td>
<td>Studium</td>
</tr>
<tr>
<td>koaLA</td>
<td>Studium</td>
</tr>
<tr>
<td>Kosten im Studium</td>
<td>Studium</td>
</tr>
<tr>
<td>Exmatrikulation</td>
<td>Studium</td>
</tr>
</tbody>
</table>

Table 5.8: Sample exported results with path structure

**Adaptations to Wecaso:**

<table>
<thead>
<tr>
<th>Model</th>
<th>User, Experiment, Languages, TestUsers, Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Export _data _form, Extable</td>
</tr>
<tr>
<td>Controller</td>
<td>ExportController@resource</td>
</tr>
<tr>
<td>Dependencies</td>
<td>DataTables.js, Laravel Excel</td>
</tr>
<tr>
<td>Access</td>
<td>Authentication required</td>
</tr>
</tbody>
</table>

Table 5.9: Export Data Components

### 5.2.2.7 Account Management

The application offers the ability to add or edit profile information for logged in users. The view uses bootstrap tabs to add profile information and to change password. Profile images are stored in /img/userimages/ folder with the user id as file name. Once the image the stored, the relative path generated from root of the application to the image is saved in User::model().

**Adaptations to Wecaso:**

<table>
<thead>
<tr>
<th>Model</th>
<th>User, Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Profile</td>
</tr>
<tr>
<td>Controller</td>
<td>UsersController@resource</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Laravel Auth::() and Storage::()</td>
</tr>
<tr>
<td>Access</td>
<td>Authentication required</td>
</tr>
</tbody>
</table>

Table 5.10: Account Management Components
5.2. IMPLEMENTATION

The user email once created is made non-editable as the system uses the email to identify the user during login. All other data from the profile settings page such as bio and addresses are saved in User::model() identified by the user email and are editable. To change password, the Laravel Auth::() and Hash::() checks user and password authenticity of the existing password before altering the value with new data.

5.2.2.8 Language Handler

The languages from top menu as shown in Fig 6.16 lists all available language translations in the application. When user picks a language, the selected language is set as the default for the user. User can toggle between languages at any point and whenever a language change is detected, the default language for the user is also changed. This ap-
CHAPTER 5. TECHNICAL WALKTHROUGH

Figure 5.18. Language Switcher

The approach is intended because when a registered user logs in to the system, the application is loaded with the default language setting as selected by the user. In case of new users, English is set as the default fallback language.

Users with the role of admin have the access to add, edit and remove languages. Option to manage languages is hidden from regular users other than admin. To avoid inconsistencies while handling multiple languages, English is set as the base language for the application. Language management is comprised of two views; one to add or remove languages and the other to edit translations and set publish or un-publish modes. Fig 5.19 shows the view to add or remove languages. To avoid break-down during translations, the fallback language cannot be deleted from application but is made editable for future extensions or changes to text. The edit language view, shown in Fig 5.20, generates table with three columns as explained below.

- **Placeholder**: Placeholder is the unique text that identifies and bridges text-area to respective language translation.

- **English**: This column help translators to understand textual semantics from English language and enter the new language’s equivalent for the same.

- **<Selected Language>**: The value for the column name changes dynamically depending on the language selected. For example, in Fig 5.20, the third column
corresponds to the translation for 'German'. The column contains form elements where translated texts are entered and stored in Language::model().

The ‘Save’ button allows to add translations to the language in patches and continue later on. Publish and unpublish toggles depending on flag status of the language. When a new language is published, all translations of the language are aggregated to Language::model() and is listed in the language switcher menu for all users. Figure
5.21 illustrates how placeholder connects to language text stored in the database. Every placeholder text is passed as an argument to the translate(string) helper function. The following is the code snippet for the helper function that runs on every page load.

```php
static function translate($string)
{
    if($user->fk_language==NULL || $user->fk_language=="")
    {
        $user->fk_language =1;
        $user = User::find($id);
    }
    $lang = Language::find($user->fk_language);
    if($lang->published=="no")
    {
        $user->fk_language->save() =1;
        $lang = Language::find($user->fk_language);
    }
    if($lang->$string=="" || $lang->$string==NULL)
    {
        $eng = Language::find(1);
        if($eng->$string=="" || $eng->$string==NULL)
        {
            return $string;
        }
    }
    else{
      return $lang->$string;
    }
}
```

The function checks the default language that is set for the user, connects to Language::model() to fetch the translation for the placeholder text. If no translations are found for the language, English is set as the default language. As a further enhancement to control erroneous exceptions while handling language translations, the placeholder text is displayed when no translations are found in any languages.
Figure 5.21. Process diagram illustrating language translations
5.2.2.9 **Experiment Handler**

Experiment handler models the experiment view, where the actual experiment is performed by test participants. The URL for respective experiments are generated at the dashboard, after creation of an experiment, and the link is publicly accessible to all users. The structure of experiment URL is readable by the experiment name and is standardized for all types of experiment. The following code snippet generates the URL for the experiment welcome message. Here, the variable for experiment name dynamically takes in the name of the experiment.

```php
$root = 'wecaso.de/';
$url = '{! $root !}/experiment/{! $experiment_name !};
```

The experiment views follows a wizard structure and each page of the wizard is associated to individual views. The flow of the experiment is ordered as per the accordion structure in the dashboard; it starts with the welcome message, questionnaire, instructions, sort data and acknowledgement. Fig 5.22 shows the screen shot of the welcome message in the experiment view and the view also serves as the homepage for the experiment.
5.2. IMPLEMENTATION

Figure 5.23. Experiment Questionnaire
The second view of the experiment, as seen in fig 5.23, holds the questionnaire. Based on the questions set at the dashboard, question items are rendered in this view. If a question requires an answer, the HTML attribute 'required', along with Laravel's form validate class checks if the question was answered. Some users tend to answer the questionnaire and submit them without performing sorting of cards. Such submissions are irrelevant as they do not contain sort results. To avoid such cases, answers posted from the questionnaire are not saved in the database immediately, but they are stored in Laravel sessions temporarily. After completion of card sorting, the sort data along with the answers stored in sessions are then retrieved and concatenated into strings. The instruction view in fig 5.24 has the same template pattern as the welcome message.

The sort view, as seen in fig 5.26, is divided into open and closed sub views. Depending on the type of experiment, the respective view is rendered. JQuery Nestable plugin is called during the page load and it plays a crucial role in sorting of elements. All items or cards are listed on the left. The handlers allow the item to be moved around and placed into other categories. Bootstrap modals are used to display the instruction set to end the experiment and to add categories in case of open sort experiments. The modal dialog to add category checks the submitted category name for any special characters. It is important to note that the sort results are stored in JSON with underscore as the delimiter for category names. In order to avoid data mix up during export to .csv, users creating category names are restricted from using underscore in the text. The instruction set in the sort view displays the same information as the instruction view. The add category button opens a form where a text can be entered as category name. The given text is fed to Nestable() plugin to load a category list where cards can be placed. All dynamically created cards and categories are stored in the DOM to avoid immature submissions of data.

The button ‘End Experiment’ opens a modal, as presented in fig 5.27, where user can optionally provide the email and comments if any. Once the user confirms to end experiment, all DOM elements are manipulated once again and are sent via AJAX to experiment controller in JSON format. Only then the answers to questionnaire are fetched from Sessions() and stored to TestUsers::model() along with the sort data, email and comments. Once all data is successfully stored, Session() data is flushed and the success ajax flag is received at the experiment view, after which the acknowledgement view (fig 5.28) is rendered.
5.2. IMPLEMENTATION

**Figure 5.24.** Experiment Instructions

**Figure 5.25.** Modal to add category for open sort experiments
CHAPTER 5. TECHNICAL WALKTHROUGH

FIGURE 5.26. Open sort experiments with cards being sorted

FIGURE 5.27. End Experiment Modal
Adaptations to Wecaso:

<table>
<thead>
<tr>
<th>Model</th>
<th>TestUsers, Languages, Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Welcome, Instructions, Questionary, Open-sort, Closed-sort, Acknowledgement</td>
</tr>
<tr>
<td>Controller</td>
<td>ExperimentController@resource</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Laravel Auth::() and Nestable.js</td>
</tr>
<tr>
<td>Access</td>
<td>Authentication required</td>
</tr>
</tbody>
</table>

Table 5.12: Components used in Experiment Handler
This chapter describes the features of several test components integrated for testing purposes such as test plugin, SQL Map [22] and beta test. Unlike unit test which is designed to verify the functionality of single part of the system individually, the test plugin is used to test the system as a whole. Additionally, SQL Map, the popular penetration test tool, is used to detect potential query based injection flaws. And beta Test, usually, is for end users instead of developers to use the system in order to find out potential bugs, was conducted and reported bugs were fixed accordingly. Last but not least, logging was enabled to collect the data generated during the testing and the log entries were monitored for the time period of beta test to verify the system's quality of service.

6.1 Test Plugin

Unit Test, as mentioned above, is used to test the correctness and robustness of a specific component of the system. For a certain functionality, all possible representative inputs need to be examined, and code coverage should be considered as well. And each unit test case is separate from another, which is good for finding out problems and fixing them. Ideally, every single functionality should have a corresponding unit test case created, but with Laravel test plugin, several test cases can be combined into single test bed. That way, the entire application is tested as a whole.
public function login_form()
{
    $this->visit('login')
        ->submitForm('login', ['email' => 'johndoe@me.com',
                              'password' => '12341234'])
        ->see('Welcome!')
        ->onPage('dashboard');

    $this->visit('login')
        ->submitForm('login', ['email' => 'johndoe@me.com',
                              'password' => '12341235'])
        ->see('LoginError!')
        ->onPage('login');

    $this->visit('login')
        ->submitForm('login', ['email' => 'johndoe@me.com',
                              'password' => '12341234'], 'remember_me' => 'yes')
        ->see('Welcome!')
        ->onPage('dashboard')
        ->db(sessions_timeout());
}

As an example, the code snippet mentioned above performs the unit test for all possible actions that can be performed on the login page. If any of the test function yields false, all tests for the page also yield false. Such integrated test functions for single functionality, login in this context, imitates a mix of both black and white box testing where the login component is tested with various actions and expected results. The test cases are iterated throughout the application. In summary, most functionalities of the system are tested by the test plugin.

### 6.2 SQL Map

As a full-fledged application hosted on the web, it is essential to perform penetration testing to check vulnerabilities if any exists. By definition, penetration testing is the process of gaining access to resources without obtaining permissions to do so.
6.3 Beta Test

As the definition of beta testing can be seen as “allowing the software to undergo usability testing with users who provide feedback, so that any malfunctions these users find in the software can be reported to the developers and fixed”, it is essential for the application to undergo beta testing. The challenge in beta testing is to reach-out the users who are willing to experience the product and more importantly, provide feedback or suggestions on possible improvements. The job of a beta tester is to do just that: to test run the product on all possible use cases and report bugs if found. There are two types in beta testing; open and closed. In the open type, the working application is accessible to all users to conduct tests and in the closed type, the developers limit availability of
the application only to certain number of users. The advantage of choosing an open beta test approach is that the application simulates real deployment scenarios where multiple users experience the application and this paves way for maximum collection of feedback data. Such tests deliver crucial data as each user has their own perspective and preference to test, which is significant to find out the potential bugs or defects to enhance the robustness of the application.

To take the full advantage of open beta tests, students in the university were asked to perform a card sort experiment. In the test, a demo experiment to organize website content for Universität Paderborn (UPB) was created and the link was sent to students by email. An open card sort experiment was chosen for the test purpose mainly because the open type has all functions that a closed card sort type can do and also provides additional requirements such as dynamic creation of categories, renaming of category names and verifying if the entered category name is redundant. Options to sort cards into a hierarchy and multiple insertions were enabled to cover all use cases. The language of the experiment was German; this has helped to test inconsistencies in the application while handling built-in translations and also to assess the performance of the sort feature (i.e drag and drop) when German special characters like umlauts exist in cards. The questionnaire in the experiment was intended to gather details from users about the device and browser that was used to perform tests. This data is crucial for a web application because if any inconsistencies in lay-outing was encountered on any device and reported, we could easily refer to the questionnaire responses to find the type of the device or the browser that was used. The experiment questionnaire contained the questions as shown in Fig 6.3.

Many students performed the experiment on various devices and browsers. Some of the bugs were reported as comments using the default comment feature for the experiment. Considering possible critical bugs, for instance if the user couldn’t save the question responses or sort cards, a provision to report bugs using Google Forms [Fig 6.8] was added to the header of the experiment page. As the form was hosted outside the application, users that encounter critical mishaps and couldn’t get past the instructions page to use the comment feature, could use Google Forms to report issues. The test ran for a week and by end of the duration, the application had recorded results from 22 participants and had 3 feed-back comments. On a positive note, all of those 3 comments were minor enhancements and no critical bugs were reported. Comments that were recorded are discussed below.

• **Special characters in category names:** This bug was encountered when user
wanted to use special characters in category names. While doing so, the application didn’t respond and so the category with the entered name was not created. This happened because the application uses special characters internally for text management. Status of this reported bug is fixed. Now the user is alerted with the message that special characters such as underscore and ampersand are not allowed in category names.

- **Experiment language:** A feature enhancement was requested to enable users to switch language of the experiment at the experiment view. This feature request is not considered because the researcher sets the language of the experiment at the dashboard. Only the researcher can switch languages and not the test participant.

- **The Handler button:** The handler to drag and drop cards displayed the letter ‘D’ in Firefox. This happened because the HTML special character was not rendered properly or supported by the browser. Status of the bug is fixed and tested.

After the test phase, the application was unit tested once again to validate all changes that were made before the final version was deployed to live server. The remainder of this section contains screenshots of the demo experiment followed by feeding the exported results in Casalysis for evaluation.
CHAPTER 6. TESTING

Figure 6.3. Questionnaire

Experiment: UPB

Questionary

1. How often do you visit UPB’s official website? *
   - Quite often
   - Once a month
   - Once a Semester
   - Not very often
   - Never

2. What device are you using for this experiment? *
   - Android
   - iPhone
   - Tablet
   - Netbook
   - Laptop
   - Desktop
   - Other

3. The browser you’re on right now? *
   - Chrome
   - FireFox
   - Opera
   - Safari
   - IE
   - Other

Save & Continue
Experiment: UPB

Instructions

Step 1
Take a quick look at the list of items to the left.
We'd like you to sort them into groups that make sense to you.
There is no right or wrong answer. Just do what comes naturally.

Step 2
To sort, drag the "Handler icon" on the left side of each card.
move them into appropriate categories and
drop card on the placeholder that appears as you move.

Continue

Figure 6.4. Instruction Set
CHAPTER 6. TESTING

**Figure 6.5.** Modal dialog to add categories in an open sort experiment

**Figure 6.6.** Cards sorted into categories
6.3. BETA TEST

If you have comments about this experiment you can record it here.

Optionally, you can enter your email address so that we keep you updated about the progress of this experiment.

<< your email >>

End Experiment

**Figure 6.7.** Modal dialog to end an experiment

**Figure 6.8.** Google Form to report critical bugs
CHAPTER 6. TESTING

**Figure 6.9.** Acknowledgement View

**Figure 6.10.** Export sort data to .csv format

**Figure 6.11.** Importing sort results in .csv to Casolysis 2.0
6.3. BETA TEST

**Figure 6.12.** Allocating columns in .csv to corresponding holders

**Figure 6.13.** The database view after import
CHAPTER 6. TESTING

**Figure 6.14.** Calculated distance matrix

**Figure 6.15.** Single link view
6.3. BETA TEST

FIGURE 6.16. MDS Evaluation
7.1 Summary

On the web, the amount of digital data present is in mind-boggling proportions. It is estimated that around 170,000 domain names are registered every day, out of those registered names, even if 10% of them grows into a web application that holds information, making the application stand-out among peers is indispensable. To favor achieving such an endeavour, usability needs to be taken into account as it has become the judgemental factor to weigh an application. With the growth of the Internet, along with various digital devices to render the web, it is not surprising that applications are being judged based on its usability. One of the top aspects of usability is data findability, that is in locating and navigating to desired content.

Organizing information to suit user expectations is a challenge imposed on design experts. Moreover, the structural taxonomy generated by design experts need not necessarily suit the expectations of the end user. Hence, it is vital to consider perceptions of users while structuring the content and menu navigation of an application. Card Sorting is one in the family of user-centric design techniques where cards, i.e the data, is sorted into groups and the experiment is repeated for each user, generating custom sort results. In the end, the sort results are evaluated to produce better structural taxonomies. The two major variants in card sorting, open and closed, with options of multiple insert and hierarchical sorting, are implemented in this thesis.

The rise of mobile devices has instigated the need for mobile friendly responsive
design technique for web based applications. In a responsive design, the template adapts to the screen size, rather than on the device. Having a mobile friendly design is no longer considered as a nice feature to have, but, a necessity. The card sorting web application implemented in this thesis is built with responsive design and can cater mobile audiences to perform sort experiments.

7.2 Future work

The thesis provides a web-based tool for Card Sorting experiments. Though all major features and enhancements are implemented in this version, the tool can be extended in several ways which are discussed below.

7.2.1 Image and Video Sort

The current version of the application provides text input as an entry for cards to be sorted. The feature can be extended to support images and video based sort. With Laravel's storage handlers, media files such as images or videos can be managed efficiently. Alternatively, YouTube and Flickr's embed API can be used to handle media files, that way the media is not stored within the application thus, the storage space is saved. In the exported results data, the title of media files can be fed as card labels.

7.2.2 Hybrid Sort

As skeleton templates for open and closed sort experiments are already set in the application, extensions to support other hybrid variants in card sort such as semi-open and semi-closed are possible.

7.2.3 Evaluation of Hierarchical Sort

In hierarchical enabled experiment results, the nested cards are illustrated as a path with '/' as the delimiter between cards. As hierarchy based sort is newly implemented in this version of Wecaso, evaluation schemes for nested card labels to generate taxonomies can be expanded.
7.2.4 Integrating Casolysis

As part of modern software practices, resource based routes and controllers are implemented in the application. The benefit of such resource is that RESTful API can be built at the application’s end. Similarly, REST API can be extended in Casolysis tool such that both applications can exchange data via RESTful service. By this approach, Casolysis can be easily integrated within Wecaso and evaluation results from Casolysis can be rendered in the dashboard view of Wecaso.
https://www.flickr.com/photos/rosenfeldmedia/.

Web usability: Principles and evaluation methods.

Card sorting, category validity, and contextual navigation.  

Research-Based Design and Usability Guidelines, 1, 2006.

Information design using card sorting.

http://cardsort.cs.uni-paderborn.de/.

Card sorting - designing usable categories.


BIBLIOGRAPHY


   https://laravel.com/docs/5.1/.

[14] SitePoint.  


[16] MySQL.  
   https://www.mysql.de.

[17] GeoIP.  
   https://github.com/pulkitjalan/geoip.

[18] JVectorMap.  


[22] SQLMap.