Front-end Migration and User Experience Improvements in the DecSpace MCDA Framework

Rúben André Paulino Garcia Rodrigues

Thesis to obtain the Master of Science Degree in Information Systems and Computer Engineering

Supervisor: Prof. José Luís Brinquete Borbinha

Examination Committee

Chairperson: Prof. Alexandre Paulo Lourenço Francisco
Supervisor: Prof. José Luís Brinquete Borbinha
Member of the Committee: Prof. Sérgio Luís Proença Duarte Guerreiro

May 2019
Acknowledgments

I would like to thank my parents, sister and girlfriend for their unconditional support throughout my academic journey in many different ways. They all made it possible by never doubting my capabilities and restlessly encouraging me even in the hardest times.

I would also like to extend my gratitude to my dissertation supervisor Prof. José Borbinha for giving me the opportunity to work on this project, for all the support and sharing of knowledge, and for giving me the freedom to work at my own rhythm. Additionally, a special thanks goes to Ana Sara Costa, André Xiang and Prof. José Figueira, for cooperating with the project and giving outstanding feedback.

Last but not least, to my close friends and colleagues that helped and travelled along with me during the extension of my studies at Instituto Superior Técnico: thank you for everything.
Abstract

Multiple Criteria Decision Aiding (MCDA) is a field of study dedicated to developing methods and techniques that help decision-makers in complex decision scenarios involving multiple criteria, often conflicting. DECSPACE is a web-based Decision Support System (DSS) - a software tool that implements modular MCDA methods in a framework, which can be composed and connected freely inside projects. The framework allows users to upload their own data and create and save persistent projects containing workflows with multiple method modules and data modules.

The main objective of this work was the migration of the front-end of the DECSPACE application into the Vue.js JavaScript framework, while simultaneously implementing several usability improvements. Some relevant existing DSS tools were analyzed in order to extract their greatest strengths, contributing to the development of a new DECSPACE prototype. This new version is stable and future-proof, ready for further improvements and integrations with other MCDA projects.

After the development was finished, the resulting prototype was evaluated through tests with users, both familiar and unfamiliar with MCDA. The issues raised by the evaluation were fixed, resulting in an improved new version of the prototype.

Keywords

Multiple Criteria Decision Aiding; Decision Support System; Web Application; Framework; Front-end Migration; User Experience.
Resumo

Apoio Multicritério à Decisão (AMCD) é uma área de estudo dedicada a desenvolver métodos e técnicas para ajudar os decisores em cenários de decisões complexas envolvendo múltiplos critérios, muitas vezes conflitantes. O DECSPACE é um Sistema de Suporte à Decisão (SSD) online - uma ferramenta de software que implementa métodos AMCD modulares numa framework, que podem ser compostos e conectados livremente em projetos. Esta framework permite aos utilizadores fazer o upload dos seus ficheiros de dados e criar e gravar projetos persistentes, que contêm workflows com múltiplos módulos de métodos e módulos de dados.

O principal objetivo deste trabalho foi a migração da interface frontal da aplicação DECSPACE para a framework de JavaScript Vue.js, ao mesmo tempo implementando várias melhorias de usabilidade. Algumas ferramentas de SSD relevantes foram analisadas de forma a extrair os seus pontos fortes, contribuindo para o desenvolvimento do novo protótipo do DECSPACE. Esta nova versão é estável e “à prova do futuro”, pronta para futuras melhorias e integrações com outros projetos AMCD.

Após o final do desenvolvimento, o protótipo resultante foi avaliado através da realização de testes de usabilidade com utilizadores, tanto experientes como inexperientes com AMCD. As questões surgidas durante a avaliação foram resolvidas, resultando numa nova e melhorada versão do protótipo.

Palavras Chave

Apoio Multicritério à Decisão; Sistema de Suporte à Decisão; Aplicação Web; Framework; Migração de Interface Frontal; Experiência do utilizador.
4.2.2 Source Code Structure ........................................ 38
4.2.3 Completed Goals ............................................. 39
4.2.4 Web Application Pages ........................................ 42
  4.2.4.A Toolbar ..................................................... 42
  4.2.4.B Home Page ................................................ 42
  4.2.4.C My Projects & Public Projects ............................. 43
  4.2.4.D Methods .................................................... 43
  4.2.4.E Workspace ................................................ 43
  4.2.4.F Other Pages ............................................... 43
4.2.5 Development Milestones ...................................... 44
4.2.6 Method Modularity ........................................... 49
4.2.7 Communication in the Workspace ............................. 51
4.3 Demonstration ................................................... 53

5 Evaluation ......................................................... 57
  5.1 Preparation ...................................................... 59
  5.2 Results ......................................................... 60
  5.3 Conclusions ..................................................... 61

6 Conclusions and Future Work .................................... 63
  6.1 Conclusions ..................................................... 65
  6.2 Future work .................................................... 65

A User Guide ......................................................... 69
B System Evaluation Guide ......................................... 79
C User Evaluation Survey ......................................... 83
D User Evaluation Results ......................................... 89
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>General overview of the PriEsT desktop software</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Input for the PAPRIKA method from the 1000minds web application</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>A radar chart as data output from 1000minds</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>The diviz workspace and its different panels</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>General overview of the Analytica software</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Workspace view of DECSPACE 1.0</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Use case diagram of the DECSPACE 1.0 service</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>UML domain model diagram of the DECSPACE 1.0 service</td>
<td>24</td>
</tr>
<tr>
<td>3.4</td>
<td>Deployment diagram of the DECSPACE 1.0 service</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>The JavaScript-based MEAN stack</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Use case diagram of the DECSPACE 2.0 service</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>UML domain model diagram of the DECSPACE 2.0 service</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>Deployment diagram of the DECSPACE 2.0 service</td>
<td>36</td>
</tr>
<tr>
<td>4.4</td>
<td>Directory structure of the DECSPACE source code</td>
<td>38</td>
</tr>
<tr>
<td>4.5</td>
<td>A typical workflow with the DCM-SRF method</td>
<td>40</td>
</tr>
<tr>
<td>4.6</td>
<td>Examples of user feedback messages</td>
<td>41</td>
</tr>
<tr>
<td>4.7</td>
<td>Home page and Public Projects page of DECSPACE</td>
<td>42</td>
</tr>
<tr>
<td>4.8</td>
<td>Methods page and Workspace page of DECSPACE</td>
<td>44</td>
</tr>
<tr>
<td>4.9</td>
<td>Demonstration method created for the 15th DDWS</td>
<td>45</td>
</tr>
<tr>
<td>4.10</td>
<td>Backlog of the project in Wunderlist</td>
<td>46</td>
</tr>
<tr>
<td>4.11</td>
<td>An interactive table for the criteria of the DCM-SRF method</td>
<td>47</td>
</tr>
<tr>
<td>4.12</td>
<td>DCM-SRF method interactive modal</td>
<td>48</td>
</tr>
<tr>
<td>4.13</td>
<td>Components of a method</td>
<td>49</td>
</tr>
<tr>
<td>4.14</td>
<td>Single file component structure</td>
<td>51</td>
</tr>
<tr>
<td>4.15</td>
<td>The publish-subscribe software pattern</td>
<td>53</td>
</tr>
</tbody>
</table>
4.16 The restaurants' example in DecSpace ............................... 54
5.1 Example of new user feedback message ............................... 61
D.1 Distribution of MCDA familiarity of the users ........................ 90
List of Tables

2.1 Feature comparison of analyzed MCDA tools ........................................ 16
4.1 Description table of DECSpace 2.0 use cases .................................... 34
4.2 Degree of completeness of goals ....................................................... 39
4.3 Restaurants’ example data set .......................................................... 55
D.1 Degree of difficulty and usefulness of features .................................... 90
D.2 Usability questions ........................................................................... 90
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHP</td>
<td>Analytic Hierarchy Process</td>
</tr>
<tr>
<td>ANP</td>
<td>Analytic Network Process</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BMSD</td>
<td>Business Modeling and Software Design</td>
</tr>
<tr>
<td>CAPSI</td>
<td>Conferência da Associação Portuguesa de Sistemas de Informacão</td>
</tr>
<tr>
<td>CAT-SD</td>
<td>CATegorization by Similarity-Dissimilarity</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DCM-SRF</td>
<td>Deck Cards Method: Simos-Roy-Figueira</td>
</tr>
<tr>
<td>DDWS</td>
<td>Decision Deck Workshop</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DSS</td>
<td>Decision Support System</td>
</tr>
<tr>
<td>ELECTRE</td>
<td>ELimination Et Choix Traduisant la REalité</td>
</tr>
<tr>
<td>EWG-MCDA</td>
<td>EURO Working Group on MultiCriteria Decision Aiding</td>
</tr>
<tr>
<td>FAQ</td>
<td>Frequently Asked Questions</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>LAMP</td>
<td>Linux, Apache, MySQL and PHP</td>
</tr>
<tr>
<td>MACBETH</td>
<td>Measuring Attractiveness by a Categorical Based Evaluation Technique</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MCDA</td>
<td>Multiple Criteria Decision Aiding</td>
</tr>
<tr>
<td>MEAN</td>
<td>MongoDB, Express, Angular and Node.js</td>
</tr>
<tr>
<td>MEVN</td>
<td>MongoDB, Express, Vue.js and Node.js</td>
</tr>
<tr>
<td>NPM</td>
<td>Node Package Manager</td>
</tr>
<tr>
<td>OOP</td>
<td>Object-Oriented Programming</td>
</tr>
<tr>
<td>PAPRIKA</td>
<td>Potentially All Pairwise RanKings of all possible Alternatives</td>
</tr>
<tr>
<td>PROMETHEE</td>
<td>Preference Ranking Organization METHod for Enrichment of Evaluations</td>
</tr>
<tr>
<td>PriEsT</td>
<td>Priority Estimation Tool</td>
</tr>
<tr>
<td>QOL</td>
<td>Quality Of Life</td>
</tr>
<tr>
<td>XMCDA</td>
<td>Extensible Multi-Criteria Decision Analysis</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
# Introduction

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Problem Description</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Motivation and Objectives</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Document Structure</td>
<td>4</td>
</tr>
</tbody>
</table>
Making decisions is a staple of human nature, being rational beings. Multiple Criteria Decision Aiding (MCDA) is a branch of Operational Research dedicated to analyzing and solving problems characterized by multiple criteria. Its main objective is to support decision-makers during the decision process by producing algorithms that can aid them in these situations (like choice, ranking and sorting problems). An MCDA method serves a specific type of problem, appropriately processing certain input parameters and returning some output, which is supposed to help the decision-maker with the decision. As more and more of these MCDA methods are developed, several software solutions have come up. These Decision Support Systems (DSSs) help the users by running hard computations required by MCDA methods, many times with very large datasets.

DecSpace is a web application that intends to be a multipurpose DSS by providing a framework for composing and executing several MCDA methods. The project was originally developed by Master’s students from Instituto Superior Técnico, with its main objective being to be a web-based, open-source, easy to use application with a large catalogue of MCDA methods as well as connections to other existing open-source software. However, the application has some problems which make it necessary to conduct changes to it on the technology level, particularly on the front-end side. This dissertation documents those problems and the work that was done to solve them and improve the application.

1.1 Problem Description

As MCDA evolves and is getting more popular, with the advent of new methods every year, so does the number of software solutions to support these processes. These DSSs exist in a wide variety of styles, serving different purposes and entities. Most of them, however, either serve a single very specific purpose (i.e., implement a single specific MCDA method) or are tremendously outdated, particularly concerning usability and user experience. None of these tools are particularly friendly to users that are not familiar with MCDA, even though some MCDA methods could be easily understood and applied in casual contexts, precisely by casual users.

DecSpace was originally developed to meet these needs, but some problems had arisen: the front-end implementation of the project was unmaintainable and several new features were required. Besides, DecSpace should have been better integrated with other ongoing MCDA projects.

1.2 Motivation and Objectives

Due to the inherent problems of the original version of DecSpace, a major task needed to be done: a full migration of the front-end technology. This implied a full rewrite of the front-end, which could be done while simultaneously implementing new features and usability requests. The new prototype should
be maintainable by upcoming developers, by providing a clean and modular system to implement new methods for the framework. Also, the groundwork for future integrations with other MCDA projects should be established.

### 1.3 Contributions

The final solution is a robust prototype of the DecSpace application implemented with new front-end technologies. The framework is stable and many desired features were implemented into it, with a significant amount of work dedicated to making it maintainable and flexible for future implementations of new MCDA methods. A module generation system was developed, separating the logic of methods into module, modal and service, which will be crucial for the near future of the project, as multiple new developers get on board.

During the development, many new features were implemented, making the new version an improvement from the previous one. The new DecSpace version competently allows users to check information about MCDA methods, experiment with them by generating modules on the workspace, and to create complex workflow scenarios by composing multiple modules, connecting them via the adequate connection points. Each project has also some very useful features, such as duplication, workflow importing and exporting (to local files), module annotations, sharp incompatibility detection when connecting modules and loop detection when executing workflows. Finally, some interfaces were specifically made from scratch for the supported MCDA methods, which directly answer the needs of their users.

New iterations of the prototype were frequently reviewed by an MCDA specialist and potential future user, which contributed to a better application overall, catering to both experienced and inexperienced users.

Furthermore, a set of papers focused on DecSpace were accepted and published during this time, in Conferência da Associação Portuguesa de Sistemas de Informação (CAPSI) 2018, in the 88th meeting of the EURO Working Group on MultiCriteria Decision Aiding (EWG-MCDA) and in the International Symposium on Business Modeling and Software Design (BMSD) 2019 [1–3].

### 1.4 Document Structure

This document is structured into several different chapters:

1. **Introduction**: The current chapter, where a brief introduction to MCDA and the DecSpace project is given, as well as a description of this document. The motivations of this Master’s thesis are exposed, directly correlated to the existing problem, and the contributions that resulted from it are listed.
2. **MCDA Overview:** A general definition and overview of MCDA is given, including the types of problems and the different groups of MCDA methods that exist. In addition, an analysis is made on the state of the art of MCDA software tools. For that, four different tools are presented, including a list of their features, advantages and disadvantages. At the end of the chapter, some conclusions about the tools are drawn from this analysis.

3. **Problem Analysis:** An overview of the DecSpace framework and its problems is presented. A description of the domain is given, as well as the different use cases, architecture and technologies, all supported by relevant diagrams. At the end of the chapter, the requirements that a potential solution should fulfill are presented and justified.

4. **Solution Design and Implementation:** This chapter describes the work that was done from several different points of view, starting from the differences in architecture, domain and technologies from the previous version. A detailed technical description of the project, of methodologies used, milestones of the development and concluded objectives follows. Some of the most significant development obstacles and how they were solved are also presented, closing with a practical demonstration of the framework.

5. **Evaluation:** Following the solution details, Chapter 5 describes what was done to evaluate the work done on it. The rationale behind the chosen evaluation method is stated, as well as what the results were and what they might mean in the context of the project.

6. **Conclusions and Future Work:** The final chapter wraps up this dissertation, with a summary of the work that was done and what can be taken out of it - the knowledge generated by the Master’s Thesis. Finally, some notes about future work possibilities are given.

This dissertation is also paired with four appendixes, containing relevant documents that were produced during the Master’s thesis, including a user guide and the evaluation guide, survey and results.
MCDA Overview

2.1 Introduction to MCDA ................................................. 9
2.2 Examples of MCDA Tools ........................................... 10
This chapter presents a general introduction to MCDA, its different methods and how they can be used to solve real-life problems. Afterwards, four different DSS tools are analyzed and their advantages and disadvantages are weighted and related to the DecSpace project.

2.1 Introduction to MCDA

Decisions are a heavy part of the everyday life of humans. As rational beings, we make decisions for everything, all the time: what do we want to eat for breakfast or lunch; what TV channel should we tune in to; which car brand and model should we purchase? While some of these decisions could be more head-scratching than others, none of them were particularly hard - humans are perfectly capable of solving them on their own. However, sometimes situations arise where human beings cannot properly weight the different criteria, and thus become poor decision-makers, failing to see the best options.

For instance, if one wants to purchase a house, some of the relevant criteria could be: price, number of rooms, location, safety of the neighborhood, energetic efficiency, etc. Obviously, we would want to minimize the price, but that comes at the cost of reducing the quality of most of the other criteria. Therefore, one must reach a trade-off where all of these criteria are balanced in an acceptable way. Most of the times, humans do this implicitly, with more or less thought behind it, but for complex scenarios with too much in stake, this is not always possible [4].

MCDA is a term that is used to describe a range of mathematical techniques and algorithms that help make decisions in complex scenarios [4]. These techniques can be extremely useful in situations where different stakeholders, with different criteria and objectives (which may conflict) need to reach a common agreement [5]. Along the years, several researchers have proposed methods and techniques for aiding decision makers to handle decision situations effectively and efficiently [6]. The versatility of MCDA methods makes them ideal to solve problems in multiple areas of knowledge, from business to everyday life, with different kinds of MCDA methods available [7].

Since MCDA is for problem-solving, it is worth mentioning that problems can be split into four different main categories [5]:

- **Choice problem**: Select some of the most favorable actions available such as it is then possible to choose the best possible option;

- **Sorting problem**: Assign each option to a relevant, predefined category, effectively grouping them by some characteristics;

- **Ranking problem**: Rank the options, ordering them by certain criteria;

- **Description problem**: Describe actions and their respective consequences.
Regarding **MCDA** methods, there is a myriad of different ones. Some of these methods are more similar between each other than others, though. According to [5, 7], **MCDA** methods can be classified into three different groups:

- **Multiattribute Utility and Value Theories:** The classic approach, where to each action is assigned a utility value (which is a number that indicates the preferability of this action - many times a combination of the different criteria, weighted accordingly), resulting in global "scores" for each one. Some of the major methods from this type are: Analytic Hierarchy Process (AHP), which adds expert judgment to evaluate qualitative criteria, and more recently Analytic Network Process (ANP). Another method from this category is Measuring Attractiveness by a Categorical Based Evaluation Technique (MACBETH), which focuses more on qualitative judgments about differences of values of attractiveness between actions.

- **Outranking Methods:** They are based on the pairwise comparison of actions, taking into account the decision-maker’s preferences, the quality of the performances of the actions and the nature of the problem. These result in binary relations $S$ defined on the set of potential actions $A$ such that $aSb$ if there are enough arguments to decide that $a$ is at least as good as $b$. All in all, it means that there is, then, a preference relation between every given action. Some examples of methods from this group are ELimination Et Choix Traduisant la REalité (ELECTRE), which adheres to the definition very strictly, and Preference Ranking Organization METHod for Enrichment of Evaluations (PROMETHEE).

- **Non-Classical MCDA Approaches:** Other methods that do not fit into the previous categories are included here. Some examples are fuzzy set approach, fuzzy integral approach and verbal decision analysis.

### 2.2 Examples of MCDA Tools

Most of the times, **MCDA** methods cannot be processed simply by "pen and paper" - they imply heavy computations - so they must be supported by software tools that can do the heavy lifting while presenting the user with an enjoyable user interface. As of 2018, there is a myriad of software tools that implement **MCDA** methods, some with more features than others. Most of the existing tools, however, are traditional, desktop software applications, as opposed to the modern trend of web applications, which offer a lot more in terms of interoperability, scalability and portability [7, 8].

Below is an analysis of four relevant **MCDA** tools, which were selected for their unique approaches and relevant advances in the field. However some of these tools have usability issues, which will be highlighted to better learn from them.
2.2.1 PriEsT

Priority Estimation Tool (PriEsT) is an MCDA software tool that supports AHP-type decision-making. As said in the previous section, AHP is a decision-making technique that combines both quantitative and qualitative, objective and subjective types of judgment. PriEsT follows on the footsteps of some already existing AHP-based tools, like Expert Choice and HIPRE (both developed in the 1990s), while doing a lot of things differently [9].

In the first place, PriEsT offers multiple equally-good solutions, as opposed to the single solution provided from the precedent tools. You can easily create your own project, add your available options, define the criteria, and then PriEsT does all the work, leaving you to analyze the results, for which it provides a range of different methods. The criteria are weighted between each other (for instance, the cost criterion may be twice as important as the benefit criterion), the pairwise comparisons that are characteristic of AHP. For that, you fill in your judgments in a table (a \( n \times n \) table, where \( n \) is the number of criteria), as it can be seen in the bottom left corner of Figure 2.1. This input method is very simple, but surprisingly powerful and error-proof.

![Figure 2.1: General overview of the PriEsT desktop software](https://sourceforge.net/projects/priority/)
PriEsT also offers useful additional features, like some data visualization (right side of Figure 2.1) and Extensible Markup Language (XML) data exporting. There is also a basic mobile version, which widens the availability of the application, also available for Windows, Mac and Linux. A negative point, however, is that the tool was abandoned, given that the last version dates to the end of 2015.

Regarding the technology, PriEsT was implemented in Java, and it follows a Model-View-Controller (MVC) architecture. The MVC architecture makes it easy to separate the information itself (Model), from the way it is shown to the user (View) and from the way the user manipulates it (Controller). The data is stored in a relational database, allowing for easy access from other endpoints if needed. The graphical user interface is written on top of the Qt framework, which is cross-platform, available for Java and C++.

2.2.2 1000minds

1000minds \(^2\) is a commercial collection of decision-making tools by 1000minds Ltd, including also prioritization and conjoint analysis tools. 1000minds offers these tools in a web application. The official website claims their product has been used widely for teaching and research, as well as in many different areas of expertise (health, government, business, environment, etc.).

Regarding MCDA methods, 1000minds is powered by the in-house developed Potentially All Pairwise RanKings of all possible Alternatives (PAPRIKA) method. PAPRIKA is used to determine weights of criteria, and consists of the users rating the importance of individual criteria by pairwise comparing them \([10]\). 1000minds supports this with a modern, easy to use web application with ratings calculated via user-directed questions, an example of which can be seen in Figure 2.2.

![Figure 2.2: Input for the PAPRIKA method from the 1000minds web application](https://www.1000minds.com/)

\(^2\)https://www.1000minds.com/
1000minds also offers solutions for conjoint analysis, where large amounts of people contribute to the decision process, effectively what is called group decision-making. For that, they offer many different options, like preference surveys, online voting, ranking surveys, categorization surveys, etc. 1000minds still offers many different ways of outputting and analyzing the information. From simple tables to more advanced charts, like the radar chart exemplified in Figure 2.3.

Overall, 1000minds offers methods from the AHP and MAVT (direct weighting - user adjusts weight bars himself) categories, as well as appropriate ways to visualize and analyze the data, including written reports [8]. It does this on a relatively modern web application with multiple panels, which, of course, supports all the features that were previously described. It is also adequate for handling multiple stakeholders with its group voting and decision survey functionalities.

2.2.3 diviz

diviz is one of the best MCDA tools available because of the modern UI it presents as well as a reliable back-end and variety of MCDA methods supported. Developed by Decision Deck, a project for collaboratively development of open-source MCDA software tools, diviz is one of their flagship projects and it is supported by a couple of other Decision Deck tools. One of these tools is Extensible Multi-Criteria Decision Analysis (XMCDA), a standardized XML format for representing any MCDA methods and data, for interoperability between different tools [8, 11]. Furthermore, Division Deck provides a range of XMCDA web-services, which can be remotely called from other web applications.

diviz is one of the few tools that aim to support as many methods from different categories as possible
in a centralized, free of charge solution. The user interface is based on a workbench, with a toolbar for available methods, a tree with all workflows and, of course, the central panel where the methods, represented by boxes, are configured, set up, and executed. A screenshot is provided in Figure 2.4 where the diviz workspace can be seen. Using a drag-and-drop approach, the tool has a small learning curve and is very appealing to new users. The data to use as input must only be provided with external files, with diviz lacking the functionality to directly enter the data, which may be an obstacle for smaller projects. Also, diviz does not support written reports as output.

This tool supports numerous different MCDA methods including the ELECTRE family of methods, PROMETHEE and ACUTA. One of the problems with diviz is that it is a standalone desktop program, which lacks many of the benefits of a web application. However, diviz can be considered unique in the market of MCDA tools as it is one step ahead of most of them. Instead of supporting a single method or a single category of methods, focusing on a niche, diviz has higher ambitions and tries to be the go-to MCDA tool that obsoletes the others. Additionally, diviz is much more than a simple tool for a job - it provides a highly flexible interface which itself can almost be considered a visual programming language [8].

In regard to visualization techniques, diviz, like other tools, includes a myriad of them. From pie charts and star graphs to simpler $x$-$y$ graphs, diviz even offers the flexibility for the user to develop
his/her own additional visualization techniques, if necessary. On another note, diviz does not currently support any group setting decision features, which it probably should, on their quest to support as many MCDA methods as possible.

2.2.4 Analytica

Analytica is a "visual software environment for building, exploring, and sharing quantitative decision models", according to the official website.\(^4\) Developed by Lumina Decision Systems, Analytica is their flagship product - a quantitative decision-support environment that extends the capabilities of a traditional spreadsheet to help with decision-making \([12]\). It can even be said it is a combination of Excel and MATLAB with an extended graphical user interface \([8]\). The product has been used in a variety of different environments, including finance, healthcare, energy, telecommunications, etc. To support this claim, Lumina Decision Systems has a sole section of their website dedicated to publishing case studies that emerged from these different applications.\(^5\) Analytica is a full package, including many different tools which serve different purposes - Lumina Decision Systems even offer a variety of different distributions and packages for their clients.

Lumina Decision Systems claims one of the benefits of using their product is the speed, both when building and running. They even go as far as saying that Analytica runs 5 to 10 times and builds 2 to 4 times faster than traditional spreadsheets containing equivalent data, while exhibiting a much smaller

---

\(^4\)http://www.lumina.com/why-analytica/what-is-analytica1/
\(^5\)http://www.lumina.com/case-studies
footprint in terms of size. One of the interesting features that differentiate Analytica from classical spreadsheet software, like Excel, is its Intelligent Arrays. These allow for much better handling of multidimensional data, as it has the flexibility to even add and remove dimensions on the go. Analytica also supports the Monte Carlo methods for uncertainty analysis.

Like diviz, Analytica supports the custom development of new methods. Furthermore, it provides support for influence diagrams and has a dedicated MCDA module. Unfortunately, also much alike diviz, Analytica has no support whatsoever for group decision scenarios.

The user interface is not particularly current, but it does its job well, taking advantage of high contrast and sober positioning of panels and toolbars (see Figure 2.5). Analytica does not support a large variety of different visualization methods, but the ones it does support work really well, like influence diagrams and tornado diagrams.

2.2.5 Conclusions

After the analysis of the previously mentioned MCDA tools and frameworks, some conclusions can be made. One particularly obvious observation is that very few, if any, of these applications have current user interfaces, which makes it much harder to connect with modern potential users of the tool, which are expecting something totally different in regard to interfaces (especially non-experts). Only some of these tools are web applications, with most of them being desktop-based. An ideal MCDA framework should be used online, via a browser. A general comparative analysis of the relevant features of these tools can be consulted in Table 2.1. DecSpace should avoid the flaws presented here and embrace the strong points from the table.

PriEsT presents a nice evolution from its predecessors, and does its job well in the AHP category. However it only represents that niche, and even though its user interface works well, it is quite outdated, with Windows XP styling.

1000minds is a bit more extensive, offering scenarios for group decision-making, and doing it also in a modern fashion with a web-based approach. Additionally, it offers plenty of variety in terms of

<table>
<thead>
<tr>
<th>Feature</th>
<th>PriEsT</th>
<th>1000minds</th>
<th>diviz</th>
<th>Analytica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web-based</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Open-source</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Cross-platform</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Modern UI</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Large catalogue</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Develop new methods</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Data import/export</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Data visualization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
interesting visualization options.

diviz is the most complete solution, offering a whole platform with flexibility of methods and solid user experience. It suffers from the flaw of being a desktop standalone application, though.

Finally, Analytica is a very powerful tool, although with a very dated interface, also suffering from the fact of being a desktop application. It limits itself a bit in terms of visualization, if spreadsheets are not what the user is looking for.

Overall, a newly developed tool should extract some of the best features of all the analyzed tools. Preferably, it should do it on a web application using state-of-the-art technology in terms of user interface and back-end. One of the major conclusions from analyzing these tools is that the web interface should be able to appeal to less skilled potential users, while keeping the modernity in a clean way. Also, another thing to note is to try not to clutter the workspace too much, something some of these tools suffer a bit from.
Problem Analysis

Contents

3.1 DECSpace 1.0 ......................................................... 21
3.2 Requirements ....................................................... 27
In this section, a conceptual and technical description of the original prototype of DecSpace (referred to as DecSpace 1.0) is presented. From this definition, the associated problems and limitations that motivate this Master's Thesis are drawn. Finally, a list of goals that directly answer the stated problems of DecSpace 1.0, which were the objectives of this work, are explored.

3.1 DecSpace 1.0

DecSpace is an MCDA framework that aims to make life easier for the users, providing a simple and straightforward way to consult multiple MCDA methods, create, modify and publish both simple and complex MCDA projects [13–15]. DecSpace ambitions to be that and much more, aggregating many of the features from the solutions that already exist into a single, standalone, web application. The center core of the framework is a workspace, essentially a sandbox where users can compose multiple methods and data, represented as modules, linked by connections (Figure 3.1). The framework provides registration and login services to the users, which are then able to save their projects and edit them later, as well as means for the developers to add additional MCDA methods easily in the future. Furthermore, one of the objectives is to have a centralized framework that is able to solve a range of different problems, with different methods.

Figure 3.1: Workspace view of DecSpace 1.0
3.1.1 Use Cases

The use case diagram for DecSpace can be seen in Figure 3.2. There are four actors: Anonymous User, Registered User, Developer and Administrator. The anonymous user is a generic user which is not logged in with a DecSpace account. He/she is still able to create anonymous projects. The registered user is a user who is logged in with a DecSpace account. He/she has access to all the features of the anonymous user, but can also save his/her work persistently and privately. The Developer is able to add new MCDA methods for the regulars to use in their projects. The Administrator can manage the totality of users and projects.

Below is a brief description of each use case.

- **Access Public Projects:** The anonymous user consults the list of public projects, seeing the project name, its creator, creation date and date of the last update.

- **Manage Temporary Projects:** The anonymous user consults, creates, edits, removes or publishes his/her temporary projects.

- **Build Temporary Workflows:** Builds the temporary workspace where the anonymous user is able to create and edit methods, connections between methods, as well as input and output data.

- **Manage Own Projects:** The registered user consults, creates, edits, removes or publishes his/her own projects.
• **Build Workflow:** Builds the workspace where the registered user is able to create and edit methods, connections between methods, as well as input and output data.

• **Manage Users:** The administrator is able to manage the users, meaning he/she may delete or consult the information about any of the users.

• **Manage Catalogue** The administrator can manage the catalogue of available methods. This includes managing local methods and synchronizing diviz methods.

• **Synchronize diviz methods:** The administrator can synchronize the diviz methods that the framework offers. This involves fetching a fresh list of the XMCDA methods available in diviz.

• **Manage Local Methods:** Both the developer and the administrator can manage local methods, meaning adding or removing a MCDA method from the catalogue, making it available, or not available to be used by the users.

### 3.1.2 Domain Model

The domain model of the DECSPACE project can be seen illustrated by the UML diagram in Figure 3.3. The following paragraphs further describe each of the classes and their relations.

A user has a unique, valid email address, which is used to identify him/her. A user also has a password, name and privacy setting. For every user, the sign-up date and last login date are recorded, as well as if that user is currently logged in or not. A user can own multiple projects and consult multiple public projects.

A project has a unique ID, used to identify it, as well as a name. The creation date and the date of the last update are both recorded. A project can have multiple workflows.

A workflow is also identified by a unique ID. It may contain a description produced by the user, and the last save date is recorded. A workflow is a collection of modules and connections between modules. A connection is identified by its name and has both a source module and a target module.

A module is also identified by a name and has a top and left position, which are used to position it spatially. A module can be of two types: either a method module or a data module. A data module consists either of data imported by the user (input data module) or data generated by the method modules (output data module).

A method module contains one, and only one method. A method has a name and some parameters. The parameters are specific for each implementation of the different methods. A method can be either a local method or a remote method. A local method contains its implementation. A remote method is an XMCDA method from Decision Deck called using SOAP requests. A catalogue contains several methods, which can be chosen by the user to execute.
In DECSPACE version 1.0 there was the concept of local methods, which only had five different implementations, namely CATegorization by Similarity-Dissimilarity (CAT-SD), Inquiry, OrderBy, Sort, Deck Cards Method: Simos-Roy-Figueira (DCM-SRF) and AdditiveAggregation. However, DECSPACE later also supported a prototypal concept of remote methods (via Decision Deck web services).

### 3.1.3 Architecture

DECSPACE was designed following a three-tier architecture - a type of client-server architecture where the project is split into three fundamental components: presentation tier, application tier and data tier. This is a proven methodology for software development, with many benefits, mainly related to the modularity of the solution, making partial upgrades/modifications much easier. The three layers are described below, and can also be better understood by studying the deployment diagram for DECSPACE, presented in Figure 3.4.

- **Presentation Tier**: The presentation tier is, essentially, what we call the user interface. The user interface runs client side only, typically in a web browser in whichever machine and operating system the user is using. Its function is mainly to expose DECSPACE’s features and services to the user in a user-friendly way. It communicates with the application layer sending user requests (HTTP requests) and receiving a response, which is then processed and translated into feedback...
the user can understand.

- **Application Tier**: The application tier is where the business logic is located. It acts as a server, as in the client-server architecture, handling the requests received from the possibly multiple clients. This is where all the computations are made, including everything necessary to fulfill the user needs - particularly the local MCDA methods. For persistence, the application tier communicates with the data tier, via an appropriate connector. This is all implemented in a single, standalone web server, which is deployed on server hardware.

- **Data Tier**: The data tier is the database, which is where all the information that must be persistent is stored. The database software is responsible for storing data properly sent from the application tier only, as well as retrieving any entries that are requested, also from the application tier. The communication is made via a connector.
3.1.4 Technology

The original DECSPACE version 1.0 was developed on top of a MongoDB, Express, Angular and Node.js (MEAN) stack, which is a very popular JavaScript-based open-source framework for developing both websites and web applications. It is called a full stack, since it includes every component necessary for creating and running a web application from the ground up - from the data layer to the client. MEAN makes life easier for developers because every component is manipulated with JavaScript, which allows them to easily build robust, cohesive applications.

Each of the components of the MEAN stack is briefly described below, along with some context about where they are relevant in the specific DECSPACE architecture.

- **MongoDB**: MongoDB is a NoSQL, open-source, document-oriented Database Management System (DBMS). Being a non-relational database, MongoDB offers a lot more scalability and flexibility than traditional relational databases, allowing for use without a predefined database schema. MongoDB uses JavaScript Object Notation (JSON) based files for the database’s storage.

- **Express.js**: Express.js is a Node.js framework to build web applications and Application Programming Interfaces (APIs). It is one of the most popular Node frameworks, adding a lot more functionalities to the vanilla Node.js and including many necessary tools that are critical for web applications development - middleware.

- **Angular.js**: Angular.js is a front-end web application framework which makes development easier. It provides support for model-view-controller architectures and two-way data-binding, extending traditional HTML and DOM manipulation for faster development of single page applications.

---

1. [http://mean.io/](http://mean.io/)
2. [https://www.mongodb.com/](https://www.mongodb.com/)
3. [https://expressjs.com/](https://expressjs.com/)
4. [https://angularjs.org/](https://angularjs.org/)

---

![Diagram](image.png)

**Figure 3.5**: The JavaScript-based MEAN stack
Angular.js is currently maintained and supported by Google.

- **Node.js**: Node.js is an open-source, cross-platform, runtime environment based on JavaScript, that runs server-side.\(^5\) It is an alternative to more traditional server approaches (e.g. Apache), allowing for the full development to be done exclusively in JavaScript. It offers great performance, scalability and portability, as well as a very active, growing community. Furthermore, there is the Node Package Manager (NPM), which opens up a lot more possibilities with third-party modules for a myriad of different occasions.

These four layers constitute the MEAN stack, which is growing in popularity for its advantages mainly in scalability and performance, as opposed to classical, more conservative stacks, like the Linux, Apache, MySQL and PHP (LAMP) stack. An illustration of the stack itself can be seen in Figure 3.5.

### 3.2 Requirements

As referenced in the previous section, DECSPACE 1.0 front-end was developed using AngularJS technology. Unfortunately, it currently presents a range of problems, which highly affect the maintainability of the framework. AngularJS was abandoned by the developers, which instead focused their attention on a full rewrite of the framework, simply called Angular.

This triggered the discussion between the stakeholders and developers of DECSPACE of whether the front-end development effort should still be directed at AngularJS, or if we should switch to a different, more modern and maintained front-end technology. From this discussion and after a good amount of research, three different technologies emerged: Angular (the new version - Angular 2+), React and Vue.js. One of the reasons these three frameworks were selected was their JavaScript-based nature, just like the original Angular. This, and their compatibility with the remainder of the MEAN stack, made them ideal candidates to replace the AngularJS front-end while keeping the back-end implementation, which had no inherent problems (permitting a partial migration instead of a full one).

After deliberation, the final choice eventually fell on Vue.js, mainly picked because of the great performance it offers, as well as its simplicity and gentle learning curve. This meant that the migration of the application’s front-end to Vue.js would be the primary requirement for the duration of this work. Vue.js, or simply Vue, is an open-source, front-end JavaScript framework, just like AngularJS, which is suitable for building powerful and complex web applications and websites, especially single page ones, all while keeping a minimalist and clean design. With rising popularity over the last couple of years, Vue.js aims to be a lightweight alternative to AngularJS, while also offering very interesting features for developers, like hot-reloading, reactivity, routing, etc.

\(^5\)https://nodejs.org/en/
The solution, then, consists heavily on the process of migrating the old front-end from AngularJS to Vue.js. Since both technologies are totally incompatible, this implies a full rewrite of DecSpace’s front-end. Furthermore, since a number of reported bugs and necessary improvements already existed, these will be taken into account when developing the new user interface, resulting in a state-of-the-art, maintainable, and improved solution, relatively to DecSpace 1.0.

A list of goals was redacted by analyzing the preexisting notes on bugs and desired features and combining it with an observation of the current system. The development process was to be split into three phases. On the first phase (G1), the front-end migration to Vue would be concluded and several improvements on user experience would be implemented. Then, in the second phase (G2), focus would be on improving the management of local MCDA methods and figuring out ways to connect to more external projects. Finally, on the third phase (G3), improvements and bug fixes on already implemented methods would take place, and new local MCDA methods would be implemented from scratch. The detailed list of final goals can be consulted below.

• G1: Infrastructure and User Experience Improvements
  – G1.1: Front-end migration to Vue.js: The first major goal of the development process will be to finish the migration from AngularJS to Vue.js, which is already in motion. This implies a full rewrite of the framework’s interface, effectively starting from nothing. The old interface must be reverse-engineered and each module must be reimplemented in Vue.js.
  – G1.2: Usability Improvements: Implement the possibility of copying and pasting the modules inside the workspace, improving the usability of similar occurrences of the same methods. Implement drag-and-dropping of input files for direct upload. Improve the workspace by using a color scheme that clearly distinguishes between input modules, output modules and method modules. Allow the user to rectangle select multiple boxes, to copy, drag, or delete them.
  – G1.3: Error Reporting: Improve the error messages presented to the user. The error messages that occur when running a workflow should be much more specific, helping the user understand what exactly is wrong and how to fix it.
  – G1.4: Privacy Settings: Allow the users to change the privacy setting of a project (public/private) at any time. Implement the possibility of sharing projects with custom groups of users.
  – G1.5: Help and Tutorials: Implement tutorials of some kind, either static videos, interactive ones, or useful tooltips while filling in the parameters of a method. These should provide a higher sense of security to the user.
- **G1.6: Continuous Generic Minor Bug Fixing:** The development of the new version in Vue will undoubtedly spawn several minor bugs along the way. These generic small bugs should be continuously fixed as a process, during the development process, to avoid generating bigger problems in the future.

**G2: Catalogue and method management improvements**

- **G2.1: Management of Local Methods:** DecSpace should support more local methods to provide a cohesive and powerful framework for the users. Therefore, new MCDA methods should be implemented from scratch locally on the DecSpace framework. This includes theoretically implementing the method itself and implementing an easy to use, intuitive, interactive interface for it. This process must be repeated for each method that is to be added. Thus, this goal refers to the local method management process, how to optimize it and make it clearer.

- **G2.2: Management of External Methods:** Additionally to the already supported diviz XMCDA remote methods, DecSpace has the potential to be extended on this field. Some more remote methods should be set up, and additional external connections may be realized, particularly to the other Decision Deck projects. For instance, "The R initiative" is a package that uses the R statistical environment to enrich the MCDA process - it could be a candidate to link with DecSpace⁶.

- **G2.3: Methods Information:** Provide more information to the user for each method. For instance, more references, classification of the method and better descriptions, description of the necessary data, etc.

**G3: Local Methods Implementation and Improvements**

- **G3.1: DCM-SRF - Revisions and Improvements**
  - **G3.1.1: DCM-SRF - Ambiguity:** Resolve a situation where there is a certain degree of ambiguity in the DCM-SRF method. The situation where between criterion A and criterion B there are two rows with one blank card each is equivalent to the situation where there is a single row with two blank cards. This may confuse the users. Either the user should be warned about it or it should not possible at all.
  - **G3.1.2: DCM-SRF - Images on Cards:** Possibility to add images to the cards, making the identification of the criteria much faster. It should support a range of preselected images as well as user-uploaded custom ones. Alternatively, the cards could be filled with a color chosen by the user.

⁶https://www.decision-deck.org/project/
• **G3.1.3: DCM-SRF - Visualization**: Implement alternative ways to visualize the results with the weights of the criteria, like histograms, bar charts, pie charts, etc. This can also be applied in different ways to other methods.

– **G3.2: CAT-SD Tables**: Merge both "Criteria" and "Scales" tables for higher cohesion, since both have information about criteria.

– **G3.3: Implementation of the Choquet Integral Method**: Implementation of a new local MCDA method that has been requested called Choquet Integral Method [16], using the newly implemented foundations that makes adding new methods easier.

These goals were to be accomplished during the development phase of the Master’s Thesis and were planned accordingly. After the development phase, the end product should be a stable prototype of the DECSPACE service incorporating these features and potential others that would come up.
4 Solution Design and Implementation

Contents

4.1 Solution Design: DECSPACE 2.0 .................................................. 33
4.2 Implementation Challenges ......................................................... 36
4.3 Demonstration ................................................................. 53
This chapter refers to the execution of the objectives formalized in Chapter 3, essentially a detailed description of the work developed across six months. The new DecSpace prototype (DecSpace 2.0), a result of this work, is conceptually described to better understand the improvements made after this time, what was accomplished, how it was done, and why certain design decisions were made. A technical summary of the software that was developed is also given, highlighting used programming languages and libraries, and how they correspond to the end product, specifically certain elements of the user interface.

4.1 Solution Design: DecSpace 2.0

From the work developed during the period of this Master’s Thesis resulted a new prototype of DecSpace, which will be referred to as “DecSpace 2.0” from here on. This new prototype mainly maintains the back-end implementation of DecSpace 1.0 (with some adjustments for the new features), but has a brand-new front-end implementation in Vue.js.

For the new implementation some concepts were changed, which in turn prompted changes to the preexisting diagrams. Thus, in the following subsections are the newly-made diagrams adapted for DecSpace 2.0, with several corrections overall.

4.1.1 Use Cases

The UML use case diagram in Figure 4.1 models the DecSpace 2.0 use cases.

![Use case diagram of the DecSpace 2.0 service](image)

**Figure 4.1: Use case diagram of the DecSpace 2.0 service**
Some relevant use cases that were missing in the previous diagram are now modeled in the new one, some small mistakes were fixed and its generally simplified.

A New User is anyone that has just accessed the platform - the user can log in if already registered, sign-up with a new account or stay anonymous. The Anonymous User is a user that has not logged in - he/she is still able to see public projects, but can only work on temporary projects without logging in.

The Registered User is a user that has logged in with his/her DecSpace account and has access to all the features of the Anonymous User, plus can now also save his/her work persistently as a private project. The Developer can add newly-developed local MCDA methods to the DecSpace system to be made available. The Administrator can manage all the DecSpace users, public or private projects, as well as all methods (including the remote methods with diviz).

Each use case is briefly described in Table 4.1.

### 4.1.2 Domain Model

The domain model of DecSpace 2.0 that incorporates its behaviour and data is shown illustrated in Figure 4.2. Some attributes were added to the classes to reflect the new features and the overall diagram was compressed in a more sensible layout.
The first two main objects represented are User and Project. The User has a valid unique email address and a username that can be changed - the first one is used by the system to identify it while the second serves as an author name for projects - storing as well the encrypted password for login authentication. Every user object also has recorded the date it has created at, and the date of its last update. A user can own multiple projects, and consult multiple public projects.

The Project has a unique ID (used to identify it), the name of the project itself and the name and email of the project's owner. The creation date and the date of the last update are both also recorded, as well as the privacy state of the project. Each Project can have multiple Workflows which in turn can have multiple Modules.

The Workflow contains its save date and may also contain a description produced by the user. The Module entity (which can be either a Method Module or Data Module) has a unique id, used to identify it, a name, and attributes for top and left relative position on the workspace, which are recorded to position it spatially. Modules can be linked by Connections (source to target), which are identified by a unique name. A Data module consists either of data imported by the user or generated by the Method modules, while a Method Module contains one, and only one Method.

The Catalogue entity contains all the available methods that can be chosen by the user to be added and used. Each Method entity has a name and parameters, which are specific for each implementation of the different methods. Methods can be of two categories: Local method module which contains its own implementation and algorithms of execution; or Remote method module which contains a link to an XMCDA method from Decision Deck (called using SOAP requests).
4.1.3 Architecture and Technology

DECSpace 2.0 maintains the three-tier architecture described in Chapter 3, since the changes were made over the front-end. Due to the flexible nature of this architecture, it is possible to easily migrate the technology of one of the layers. For instance, one could also easily replace the data tier from MongoDB to any other DBMS (MySQL, for example). Nevertheless, some corrections were made to the original deployment diagram, which can be seen updated in Figure 4.3 - adequate connections between the layers, more details and mistakes fixed.

The technology stack is now a MongoDB, Express, Vue.js and Node.js (MEVN) stack, in which the front-end technology was migrated from Angular.js to Vue.js. Both stacks are very similar, being based on JavaScript, which made the migration possible.

4.2 Implementation Challenges

Development of the solution started in September 2018. At that stage, the project was passed over from the previous developer of the DECSpace project, in the context of his Master’s Thesis [15], which describes the development state of DECSpace at that time. His responsibilities involved starting the migration process from AngularJS to Vue.js, and so the bare-bones of the project were already coded in the new technology. The extent of the migration included the informational, support pages like the homepage, Frequently Asked Questions (FAQ), method catalogue and settings. Furthermore, rudimentary systems were put into place for user and project management, allowing for basic login and
sign-up, as well as project creation. Finally, a very basic workspace was implemented, only for visual demonstrations. It only included a non-executable version of the Sort method, simply to demonstrate the drag-and-drop nature of the workspace.

The immediate direction of the development was extending on what was already done to bring the project to the state it was before the migration, while at the same time implementing new features as needed and from the list of goals stated in Chapter 3. The different challenges faced during the development phase and what was accomplished are explained in the following sections, mainly focusing on the workspace (where most of the development time was spent).

4.2.1 Dependencies

One of the advantages of using Node.js and Vue.js is the great adaptability that they offer. One can take advantage of NPM to easily use many different packages that extend the functionality of the framework at the distance of a simple command. The following packages are effectively dependencies of the DECSpace project (some default Vue.js libraries used in the project are omitted here):

- **Vuetify**: A UI toolkit for Vue.js with multiple reusable components. Vuetify offers solutions for many of the most common UI design needs, namely: form controls, navigation, popups, data tables, etc. It is heavily used in almost every component of DECSpace, especially the form controls and data tables.

- **jsPlumb**: A JavaScript library that supports means for visually connecting elements on web pages. DECSpace’s workspace depends on jsPlumb for both the drag-and-drop functionality and connecting different modules.

- **JSZip**: A JavaScript library for creating, reading and editing .zip files. It is used in DECSpace for the “Export Workflow” functionality (to pack the multiple files into a .zip file).

- **Papa Parse**: A JavaScript library for CSV parsing. It is used in DECSpace for parsing and handling the data from uploaded .csv files.

- **FileSaver.js**: A JavaScript library for saving files on the client-side. It is also used in DECSpace for the “Export Workflow” functionality.

- **Find-cycle**: A Node.js package that detects cycles in a directed graph. This is used to detect invalid workflows before executing them and warn the user about it.

---

1https://vuetifyjs.com/en/
2https://github.com/jsplumb/jsplumb
3https://stuk.github.io/jszip/
4https://www.papaparse.com/
5https://github.com/eligrey/FileSaver.js/
6https://www.npmjs.com/package/find-cycle
• **Vue-sweetalert2**: A Vue.js wrapper for SweetAlert2, a JavaScript library for beautiful, responsive, customizable popup boxes. It is used everywhere in DECSPACE to provide visual, instantaneous feedback to the user.

• **Vue-smooth-dnd**: A Vue.js wrapper for Smooth-dnd, a lightweight and versatile drag and drop library for multiple scenarios. It is used in DECSPACE both in DCM-SRF and Sort methods (modals).

### 4.2.2 Source Code Structure

A high-level view of the directory structure of the project’s source code can be seen in Figure 4.4.

The back-end structure is rather trivial, being a very simple Node.js application. The node_modules folder is automatically generated by NPM when a new development environment is set up, as it contains all the modules which the project is dependent on. The src folder contains the source code of the server itself, including the Mongoose (a Node.js package for MongoDB object modeling) models for the two entities that are stored in the database - users and projects. The app.js file contains the logic of the server, mostly consisting of handling behaviour for received requests (to add, edit, delete users and projects).

---

7https://github.com/avil13/vue-sweetalert2
8https://github.com/kutlugsahin/vue-smooth-dnd
9https://mongoosejs.com/
The front-end code is structured optimally as a Vue.js application. The build and config folders are automatically generated by Vue.js. Inside the src folder, assets contains any images or other assets that are imported from the components. The components folder contains all the containers that make up DECSPACE - it is divided into subfolders for each of the pages of the web application. Each subfolder itself contains one or more Vue.js components, being App.vue the root component of the application. The bulk of the work was in components/Workspace/ which contains: Workspace.vue file, with all the general logic of the workspace; Menus folder, with some submenus used in the workspace that are complex enough to have their own component; Methods folder, with all the implemented MCDA modules and modal files, including the templates.

The router file contains a file for Vue Router. This file maps the existing components to certain URL routes (only some of the components should be accessible - other components may be auxiliary ones), effectively piecing the application together. The services folder contains the implementations of the available MCDA methods, each in its individual file. The store folder contains the necessary file for Vuex, a Vue.js official library for state management. Finally, the event-bus.js file is the custom-made event bus used for communication between components in the workspace (referred in Section 4.2.7), and main.js is the main JavaScript file where all the initializations for Vue.js are made (importing libraries, connecting to the back-end, etc.)

4.2.3 Completed Goals

The effective development endeavor ended up with some slight deviations from the original planning. One of the main reasons for these differences was the estimation of the state of the project in September 2018. In reality, the workspace was basically just a mockup at this point, thus a considerable amount of development time was spent at the migration process. In this section, the differences between planning and what was done are presented and explained.

Referring to the goals listed in Section 3.2, they can be split into three categories: fully done, partially done, and not done. In Table 4.2 are all the proposed goals sorted into the respective categories.

The development was very focused on the migration process, stability and maintainability of the platform, which still allowed for continuous usability improvements. However, this prevented the implementation of some other desired Quality Of Life (QOL) improvements planned in G1.2 (Usability

---

10https://router.vuejs.org/

11https://vuex.vuejs.org/

Table 4.2: Degree of completeness of goals

<table>
<thead>
<tr>
<th>Fully done</th>
<th>G1.1, G1.3, G1.4, G1.6, G2.1, G3.1.1, G3.1.2 and G3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partially done</td>
<td>G1.2 and G1.5</td>
</tr>
<tr>
<td>Not done</td>
<td>G2.2, G2.3, G3.1.3 and G3.3</td>
</tr>
</tbody>
</table>
Improvements), like module copy-pasting and rectangle selection. For the same reason, G1.5 (Help and Tutorials) was not fully completed, because very little work was done in the method catalogue. However, a user guide was created (Appendix A) as well as a couple of demo videos. Also, the migration itself improved the application with some help features, like tooltips on buttons with additional information to guide the user.

G2.2 and G2.3, which were originally planned and refer to external method listing and executing (particularly the Decision Deck web services) were scrapped at an early stage when the true extent of the migration progress was understood. Since the priority was the stability and flexibility for future changes on the platform, at some point it was agreed that the remote method execution would be done last. Eventually, this feature was moved to the roadmap of the project, to be picked up by the next developers.

Likewise, G3.1.3 (DCM-SRF - Visualization) and G3.3 (Implementation of the Choquet Integral Method) were also scrapped due to low priority. The platform was, however, developed taking into account multiple stakeholders (including method developers). As a consequence of this, the project is now (at the start of 2019), for the first time, welcoming more than one concurrent developer (1 platform developer and 2 method developers). So it made sense to push G3.3 to the new method developers and focus more on the platform itself. G3.1.3 would also be encapsulated into a larger, more generic task - a dedicated visualization module that would work with all the modules, not just DCM-SRF.

Overall, the completed goals were integrated into a robust prototype (referred to as pre-alpha version) that can be built upon from now on. The migration process to Vue.js is complete (G1.1), as it now has
Along with the migration, better error reporting was implemented, using a coherent system at the bottom of the screen, across all the pages of the application (fig. 4.5). Some examples of the different types of messages can be seen in Figure 4.6(a). This was implemented in a reusable way, making it possible to transmit messages to the user from any place (including from methods). For instance, one of the complaints from the previous prototype was the lack of specificity with error when executing workflows. Now, DecSpace tells the user that there is an error, states in which of the modules the error occurs, and what is actually missing from it that prevents the execution. The developers can take advantage of this system by color-coding their messages (red for error, blue for info and green for success).

Additionally, a method-specific error reporting system was implemented for the CAT-SD method, which can be extended to other applicable methods in the future. CAT-SD is a particular method implementation, since its parameters are very connected between each other. For instance, categories registered in the "Reference Actions" parameter must also appear on the "Weights" parameter. The method, thus, always verifies its input and warns the user if there is anything missing from any of the parameters. An example of an error scenario can be seen in Figure 4.6(b) where the data module connected to the "Weights" parameter is not compatible with the categories that exist in "Reference Actions".

Figure 4.6: Examples of user feedback messages
4.2.4 Web Application Pages

The main feature of the DecSpace web application is the workspace, where the users can explore MCDA methods. However, to work properly, the workspace needs a range of supporting pages. In this section, these pages are presented from a user perspective, with some notes from the development point of view.

4.2.4.A Toolbar

For navigation across the components of the web application, DecSpace relies on a toolbar and a navigation drawer. The toolbar is static and is always visible, containing links to: Public Projects, Methods, FAQ and Contact Us. On the right side of the toolbar are the user session buttons: when logged out, a sign-up and a login button; when logged in, a logout button. When a user is logged in, a menu button appears on the leftmost side of the toolbar, providing access to the navigation drawer. The navigation drawer has user-specific content - My Projects and Settings. Both of these navigation elements are always consistent across the application, making the user feel comfortable in knowing he/she can always access its content, anywhere.

4.2.4.B Home Page

The home page is the landing page of the DecSpace web application - Figure 4.7(a). Its main objective is to quickly give a new user a brief explanation of what DecSpace is and how to use it. Alternatively, the home page should also behave as quick access to users that have already experimented with DecSpace, quickly allowing them to access its main features. As such, the home page contains some information about the project and some general contact information. Furthermore, a highlighted “call to action” button encourages the user to easily try DecSpace via its anonymous workspace feature.
4.2.4.C My Projects & Public Projects

Both of these pages have a table listing DecSpace projects. My Projects refers to the projects owned by the user that is currently logged in (thus, it is only available when a user is logged in). The user can then open a project, change its name, change its privacy setting, check date and time of creation and last update, duplicate it, or delete it. On the other hand, Public Projects is always available for everyone, containing all projects on the platform marked as public. Users can open public projects, check its owner, check date and time of creation and last update or duplicate them (with ownership of the duplicate). The tables of both pages have built-in pagination controls, allowing the user to choose how many projects should be displayed per page, and to change pages.

4.2.4.D Methods

The Methods page is the catalogue of methods available in DecSpace. It lists each of the methods and displays a short description for them. If the user selects one of the methods, he/she will be directed to its respective description page, containing explanations, bibliographic references and examples. From inside the workspace, a similar version of the catalogue is also available, with the additional option of adding the selected method to the workspace.

4.2.4.E Workspace

The workspace is the core function of DecSpace. It consists of a minimalist sandbox area and a toolbar containing buttons to operate it. The buttons in the toolbar allow the user to execute the workflow, save the project and upload his/her own data files. There is also access to the method catalogue, with the ability to generate a new module of a specific method, which will appear on the workspace. If the user wishes, he/she can also clean the workspace by pressing the respective button and export the workflow to a local file on his/her computer, which can be later imported into any DecSpace workspace. The workspace can either be accessed by a registered user from My Projects, by creating and opening a new project, or by an unregistered user, by clicking the "Try it now" button on the home page. If the latter, the user will be constrained to an anonymous workspace and unable to save his/her project. He/she can, however, export the workflow and import it later.

4.2.4.F Other Pages

Finally, there are some more secondary pages that support the application and improve the user experience. The FAQ page, as the name implies, contains a list with some answers to the questions that the users might have the most. The Contact Us page allows users to easily and directly contact the
DECSPACE team via an email form with any suggestions or problems they might have. For session management, there is the Sign-up page and the Login page when the user is not logged in, and the Logout page when the user is logged in. In the latter scenario, the user also has access to a Settings page in the navigation drawer where he/she may change his/her username and/or password.

4.2.5 Development Milestones

Development started in September 2018. In a very early stage, a deadline was already coming up: the 15th Decision Deck Workshop (DDWS) on September 26th.¹² This workshop is organized simultaneously with the biyearly meetings of the EWG-MCDA, and the 2018 edition took place in Lisbon. Due to the potential interest that the DECSPACE project could raise with the EWG-MCDA, it was determined that a prototype of the new Vue.js should be demonstrated on the next meeting in Lisbon. As such, all efforts were directed towards making a preliminary prototype that could be used in a demonstration in front of the members of the EWG-MCDA.

In this period, the workspace was initially a static mockup. For instance, only the Sort method was implemented (module and modal only) and there was no concept of module generation - only one Sort module could be placed on the workspace at any given time. This was because the method was not generated, but simple visually toggled. Hence, the first thing to implement was the module generation system. To do this, and since the modules and modals were Vue components, they needed to be dynamically inserted in the document via JavaScript, since they could not be statically placed with HyperText Markup Language (HTML) (because of the unpredictable nature of the workspace there are many different possibilities). Here was also where single file components were used for the first time, encapsulating modules and modals in their own files. Until then, all the logic was inside the Workspace.vue file.

¹²http://mcda2018.idsswh.sysresearch.org/
For the demonstration, a working method was needed. The simplest method to implement was the Sort method, since it simply lets the user rearrange a set of criteria by dragging cards, and then outputs the new order. This was already partially implemented, but non-functional. Some work was done on its modal too. Since this method and its modal were very basic, an additional demonstration method was created specifically for the demonstration, which simply propagated its data to the following module. Its modal also showed what was possible to do with Vue.js with some demonstrative input controls and buttons (checkboxes, text fields, sliders, etc.). A screenshot of this demonstration method can be seen in Figure 4.9.

To demonstrate the potential of the framework, several more features were implemented. One of them was the “Execute Workflow” button’s functionality, which then allowed multiple Sort method to be instantiated, filled with their respective data and executed with a click of a button. In addition, the “Delete Workflow” feature was also implemented. Finally, the method selection window was developed, also listing methods that were not yet available, serving demonstration purposes. Since the only available method was Sort, a demo method was quickly developed, showcasing the multiple possibilities for data insertion in the modals of future methods.

After the conference, regular development could then resume. With the basic workspace working as expected with enough robustness, the implementation of the first MCDA method was started (CAT-SD) along with one of the other basic methods (Order By). These were the first methods developed from
scratch with the new module generation system, so it was a constantly adapting process. The original algorithms from DecSpace 1.0 could be salvaged after some clean-up, to be adapted as services for the new version. Most of the effort was into implementing the modals, especially CAT-SD, since it is a method with a lot of dependencies between data and parameters [17]. Several constraints existed between the CAT-SD inputs, which caused lots of bugs, unexpected behavior and overall complexity.

The development work on CAT-SD, from start to finish, took three weeks. When the module and modal already correctly handled input by hand, the need for a faster way of testing came up. Thus, the data file upload feature was implemented during this time as well. It allowed the upload of previously existing .csv files which would then be used to test the output of CAT-SD when executing it. This confirmed the correct execution of the method with three different case scenarios with large data sets.

After CAT-SD, the other MCDA method was developed - DCM-SRF. It took considerably less time due to the knowledge acquired from the previous one. At the same time, the first working prototype of the import/export workflow feature was finished. This was essential to be able to implement the save workflow feature, since it would take advantage of the same logic: exporting the workflow to a JSON, which is then stored in the database; fetching the JSON from the database and importing it. That feature came immediately afterwards, along with overall polishing work on the other pages (My Projects, Public Projects, etc.).

By this point the framework had continuously improved with new features, but some problems naturally arose with it. Some work had to be done by refactoring a lot of code, cleaning up, and solving recurrent bugs. These bugs were managed by keeping a to-do list in the Wunderlist web application, which served as the backlog of the project.¹³ Both bugs and features from the backlog were continuously

---

¹³https://www.wunderlist.com
put on the list, resulting in well-defined priorities and work estimation. In the short period when both the author and the following developer were simultaneously working on the project, Wunderlist also revealed itself as a powerful tool to distribute the work via its assignable tasks. The developers following this work will even inherit this to-do list for the future work.

At the end of November, the prototype was taking shape and continuous improvements were being implemented. For instance, the data modules, which were initially implemented as static (non-editable), became editable, so users could modify their own uploaded data modules, or other modules generated by the software. The work done on multiple methods also served to improve them iteratively, since more and more of the logic of modals and modules were transported to the templates, making the implementation of the methods themselves a lot simpler. For instance, at this time, it was decided that it would be better to have method modals with a toolbar with tabs, for navigating between parameters easily. This was immediately integrated into the previous methods.

By the end of the year, the anonymous workspace was implemented, allowing users to experiment in a sandbox without having to create an account, at the cost of being unable to save workflows, only exporting them. The consistency of the module logic was also improved: when a data module is connected to a parameter slot of a method module, trying to edit the data in the data module would then directly open the modal of the method module on the respective tab. Some changes were also made to make sure that when a user manually edited a parameter of a method module, a respective data module would be generated and connected to the former, making the visualization always consistent to the user - if a parameter is changed, a data module is connected to it.

![Figure 4.11: An interactive table for the criteria of the DCM-SRF method](image)
The color scheme and general aesthetics of the workspace were then worked on and simplified. Where before there were method modules, input modules and output modules, the latter two were merged into data modules, which is what they essentially are. This also made the colors easier to perceive: data modules are always light blue and method modules always light grey.

An example of the kind of work that was developed can be seen in Figure 4.11 and in fig. 4.12. This is one of the steps of the developed DCM-SRF method. After declaring the desired criteria with as much info as possible (step 1), the user can pick one of the available icons sourced from Google’s Material Icons set. ¹⁴ This is done through an icon picker that was developed to be reused throughout DECSpace - Figure 4.12(b). Then on step 2, the criteria are converted into playing cards which the user can distribute through a series of levels (Figure 4.12(a) [18]. Finally, when the method is executed, it will output the calculated weights of the criteria.

Until the end of the development period, an effort was then put into fixing bugs of different magnitudes, while also fixing cosmetic aspects by request of the potential users, which were constantly testing the new prototypes deployed. This included the implementation of multiple validations on the methods, making them as foolproof as possible while giving the user as much freedom as possible. Some smaller and simpler features were also added, like module duplication, module and project annotations, editable custom module names, and eventually even a new method (Additive Aggregation) which was ideal for demonstrations and tutorials.

¹⁴https://material.io/tools/icons/
After the main development period was finished, evaluation tasks were conducted (Chapter 5) with users. The results of the usability tests were then taking into account and minor fixes and improvements were implemented.

4.2.6 Method Modularity

Through the analysis of previous implementations of MCDA methods in DecSpace v1.0, some conclusions were formulated. Generally, a method can be split into three major components:

- **Module**: The representation of an instance of the method on the workspace. The module is a small, draggable box that can be moved freely inside the boundaries of the workspace. Inside the box is the name of the module, a delete button, input slots for the different accepted parameters of that method and an output slot.

- **Modal**: The interface of a method. The modal can be accessed via the respective module (by clicking the name of the module or one of the input slots). The modal is a window with a graphical interface specifically designed for the specific MCDA method. The window may contain multiple tabs, which correspond to the input slots that are in the module. For most methods, an interactive table should be adequate for manual data input, but more complex and esoteric data inserting ways can be implemented.

- **Service**: Effectively the algorithm of the MCDA method itself. This is an external component which is completely decoupled from the other two. It receives as input the parameters for that method and returns the output of the method.

These three components together are what a MCDA method is, in essence. Visual examples of each of these components can be seen in Figure 4.13. The module is the "physical" representation of the method, that makes the user easily understand that the box directly corresponds to the method that we intended. The modal is triggered when the user interacts with the module in a certain way - it is the bridge between the user and the method. In fact, it is what makes DecSpace useful: a user-friendly

![Figure 4.13: Components of a method](image-url)
way of inserting and manipulating data to be fed to MCDA methods. Finally, the service contains the
logic of the method, the algorithm itself, and is used to run it with the supplied data. It is invisible to the
user - just a piece of code that is called accordingly when the user executes the workflow - he/she only
sees the end result.

Taking these three components into account as well as their characteristics and requirements, an
infrastructure should be in place to make it as easy as possible to develop new MCDA methods for
DecSpace. For that, as much of the common logic of each of these components as possible should be
abstracted in a reusable module.

The most obvious model for the reusability problem stated in the previous section would be to im-
plement it by taking advantage of the Object-Oriented Programming (OOP) paradigm. By having pre-
developed generic classes that could be extended by future method developers, it would make the develop-
ment process itself tremendously faster, making available commonly reused parts of the methods. For
instance, in the case of the module concept, a Module class would be created. This class would contain
all the common characteristics of the modules. Then, subclasses DataModule and MethodModule could
be extended from the parent Module, further specifying the differences between modules containing
data, and modules containing method logic (e.g. data modules can be blue and method modules gray -
both are boxes). Finally, concrete methods would extend the MethodModule class, e.g. SortModule.

Unfortunately, JavaScript is not, at least natively, an OOP language. Therefore, a different solution
had to be found, constrained by the use of both JavaScript and the Vue.js framework. Vue.js, however,
supports two fundamental concepts which eventually contributed to the achievement of a fully-working
prototype. These two concepts are single file components and templates.

Traditionally, pure web development and front-end frameworks enforced separation of concerns. This
means that HTML, Cascading Style Sheets (CSS) and JavaScript code, the three main elements of a
web page, will be inevitably separated into different files with extensions .html, .css and .js. Vue.js,
however, supports the possibility of single file components. This means that the developer is able to join
the three parts into a file and develop self-contained, loosely-coupled components. These components
can then be composed into larger, more complex structures. An example of a single file component can
be seen in Figure 4.14.

The composition of components in Vue.js is possible with the use of templates. A template is a
Document Object Model (DOM) element which delimits a piece of code (a component) that can be
reused in other components. Single file components are implemented in files with the extension .vue.
More specifically, code that is inside a <template></template> block in its own single file component
(let's call it MyComponent.vue, can then be reused anywhere else where it is imported, by the auto-
matically generated DOM element <my-component></my-component>). This makes possible to create
components composed of smaller sub-components, which themselves can be composed of other com-
ponents. These concepts can be used to implement the framework for instantiating MCDA methods.

For the module and modal, files ModuleTemplate.vue and ModalTemplate.vue were respectively created. These are single-file components, containing Vue.js templates, per the definition exposed in the previous subsection. Each file is an entirely self-contained component, which can be placed anywhere on other Vue.js components. Both of these components serve as templates for potential new methods. Someone developing a new method, simply has to import the adequate template, and insert his/her own logic in a slot. A slot is a Vue.js feature which makes it possible to insert DOM elements into another component, which itself declares the place of that given, named slot.

Thus, both template files contain the bare-bones, the scaffolding of a method, making it so that the developer can easily focus on his/her part of the method, with the common logic encapsulated in the template file. In practice, the development of the different parts of a method could even be done by different developers.

4.2.7 Communication in the Workspace

The obvious obstacle of using an architecture based on loosely coupled, self-contained components is the communication between them. Since the logic is split into the different components mentioned in the previous subsection, it is not possible to have native, direct communication between them.
When new Vue.js components are instantiated, they must be mounted in a parent component. In DecSpace, both modules and modals are mounted, indexed and managed in the workspace component. For a variety of reasons stated hereafter, these modules and modals need to communicate with the workspace and between themselves. Vue.js actually supports a limited degree of communication in two different ways.

The first way is from the parent component to the child component, in the form of props. A prop is a property declared by a component which is essentially like a regular attribute, except its value is passed from the parent component. In a way, they are akin to the arguments passed to the constructor of an object in OOP. However, props are unilateral communication channels, and their values must not be mutated by the child component. The variable is effectively "owned" by the parent component, and its control belongs to it - the data flows dynamically from parent to child only, a concept called one-way-down binding.

For communicating from the child component to the parent component, asynchronous events are needed. The child component can emit an event, which has a name and payload, and is incorporated into the component itself. The only other component that has access to the child component is, of course, the parent component, because it is mounted there. Thus, the parent component may listen to certain events from the child component and handle them accordingly with callback functions. This is effectively a one-way communication channel from child to parent component.

Nevertheless, the native solutions are not enough for the DecSpace implementation. As said before, the components need to communicate in non-linear ways, mainly because:

- Modules need to communicate with the workspace when the user clicks the delete or duplicate button, or edits the module’s name or description;

- Modals need to communicate with the workspace to update it with the most recent data;

- Modules need to communicate with the respective modals when the user clicks one of the input slots, to make the modals appear.

Thus, an alternative solution had to be found. This solution came in the form of an event bus. An event bus is a communication channel that allows entities to add data to it via events, and other entities can listen to certain events to get access to that data. This is possible to implement in Vue.js by creating a component that will assume the role of event bus. This component is empty, since it will not be used visually for anything. Next, we give access to it from every component in the system that requires it (the workspace, the module components and the modal components). The components then communicate with each other as needed via events: by emitting an event with a certain name, which the recipient will be listening to with the exact same name. The recipient will then handle the event and its payload.
Accordingly. This is essentially an implementation of the publish-subscribe software design pattern, which can be seen illustrated in Figure 4.15.

As an example, when a user clicks an input slot on a module, the module emits an event called 'show-modal-<ID>' (where <ID> is the ID of the module, which corresponds to the ID of the modal) containing as payload a string with the name of the input slot. When the modal component is mounted for the first time, it registers a listener for the 'show-modal-<ID>' event, with a function that appropriately handles this behaviour. In this case, this means showing the window of the modal, and using the payload with the input parameter to open the respective tab inside the modal.

### 4.3 Demonstration

To better exhibit the capabilities of DECSPACE and what a typical use scenario for a regular user would be, in this section an example is provided. A potential user of DECSPACE could either already have his/her own prefilled data files, or take advantage of the interface provided by DECSPACE to fill in his/her data manually. For large data sets the best option is to use prefilled files and upload them to DECSPACE.

In our example, the user wants to make a decision regarding a list of restaurants. As input data, he/she has access to a list of criteria for restaurants and the respective weights that should be taken into account when deciding. Also, there is a list of options containing several restaurants and their score for each of the criteria (Table 4.3).

One of the simplest MCDA methods is Additive Aggregation, which is essentially a weighted sum [19]. For very basic decision-making and taking into account the fact that the user already has defined the weights of the criteria, Additive Aggregation is enough to judge the restaurants in a very primitive way. Thus, the user creates a new project in DECSPACE and uploads his/her two data files, generating two data modules. From the method catalogue, the user then selects the Additive Aggregation and connects
the data modules to the newly generated method module. DecSpace will warn the user if there is anything missing or if the data is not in the correct format, otherwise the input endpoints will be colored green.

Finally, the user can execute the workflow by clicking the Execute button, which will result in a scenario similar to that of Figure 4.16(a). A new data module is generated, containing the output (i.e. the result) of the executed Additive Aggregation method. The output data can be seen in Figure 4.16(b) - it displays the aggregated score of the options, sorted in descendant order. HappyPizza restaurant has the highest score (1530) based on the provided criteria and their weights. Therefore, the “best” restaurant, given the available data, is HappyPizza restaurant by a considerably large margin (with a score approximately 27% higher than the second highest).

The user is then able to save the project to DecSpace to review it later on any machine with access to the internet, after logging in to his/her account. The project can also be made public for anyone to see and make copies. If the user needs the data on a local computer file, the workflow can be exported in the form of a .zip file containing .csv files with the data modules’ content. These files can be imported into, for example, Microsoft Excel for further processing.
### Table 4.3: Restaurants’ example data set

(a) Criteria

<table>
<thead>
<tr>
<th>Name</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>50</td>
</tr>
<tr>
<td>Location</td>
<td>25</td>
</tr>
<tr>
<td>Schedule</td>
<td>10</td>
</tr>
<tr>
<td>Parking</td>
<td>10</td>
</tr>
<tr>
<td>Menu</td>
<td>2.5</td>
</tr>
</tbody>
</table>

(b) Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Location</th>
<th>Schedule</th>
<th>Parking</th>
<th>Menu</th>
<th>Variety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maria</td>
<td>7.8</td>
<td>2.0</td>
<td>12.5</td>
<td>30</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Roma</td>
<td>9</td>
<td>0.2</td>
<td>5</td>
<td>55</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Frutalmeidas</td>
<td>6.5</td>
<td>0.34</td>
<td>11</td>
<td>13</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Sakura</td>
<td>13</td>
<td>1.3</td>
<td>7.9</td>
<td>29</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Ni Hao</td>
<td>8.5</td>
<td>0.61</td>
<td>20</td>
<td>46</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Nipal</td>
<td>10.75</td>
<td>0.9</td>
<td>7.5</td>
<td>38</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>HappyPizza</td>
<td>8</td>
<td>1.5</td>
<td>15</td>
<td>89</td>
<td>21</td>
<td></td>
</tr>
</tbody>
</table>
5 Evaluation

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Preparation</td>
<td>59</td>
</tr>
<tr>
<td>5.2 Results</td>
<td>60</td>
</tr>
<tr>
<td>5.3 Conclusions</td>
<td>61</td>
</tr>
</tbody>
</table>
After the development work was finished and a stable prototype existed, a user guide was created, using as an example the DCM-SRF method. This guide was needed for an experimental use case of DecSpace - to be used in a university course with students, to explore the DCM-SRF method. The guide can be consulted in Appendix A. After briefly presenting the application, it goes through the basics, like setting up a user account, managing user projects and using the workspace. The general rules of the workspace are explained (method modules, data modules, connections, etc.), followed by a detailed step-by-step guide of the DCM-SRF method, along with a real example. The guide includes many screenshots of the steps and relevant links where applicable.

To validate the work that was done, some sort of evaluation had to be conducted. An adequate way of doing this is with usability tests, with potential users of the application [20]. Since one of the core objectives of DecSpace is to be easy to use by both experienced MCDA users and non-experts as well, it would be interesting to direct the tests towards both groups. Therefore, it was decided that the tests would be done with users with basic knowledge of English and computers, with half of them familiarized with MCDA and the other half not.

5.1 Preparation

To prepare the tests, a user evaluation guide had to be created. Since the developed work was mostly on the workspace (the support pages of the web application had already undergone usability tests when the platform was first developed, whose results can be consulted in [15]), the new tests should be focused on its features. Thus, a list of all the major features present in the workspace was made to begin with:

• Generating new methods;
• Drag-and-dropping of modules;
• Editing module name and description;
• Duplicating a module;
• Deleting a module;
• Manually editing module data;
• Deleting a connection;
• Deleting workflow;
• Executing workflow;
• Uploading data files;
• Connecting modules;
• Importing and exporting workflow.

The ideal tests would go through all these features so that the users could experience them and give feedback afterwards. Thus, a set of three tasks was produced. The first task is about the general mechanics of the workspace, serving as first contact for the user. The second task is more advanced, with a scenario for the Order By method which is properly executed. The third and final task includes user-uploaded data modules and exporting and importing of the workflow. All the tasks are done via the anonymous workspace, which can be accessed with one click from the home page, making the testing process all about the workspace and avoiding further problems.

The three tasks were compiled into an evaluation guide, which can be consulted in Appendix B. This was the guide that was supplied to the users when conducting the usability tests. The guide also includes a brief introduction and setup instructions. The user manual was made available through the evaluation guide, instructing the users to use it as they wished (they could read it all, skim through it, or not read it at all).

After doing what was stated in the evaluation guide, users were asked to fill a quick survey, rating their experience in multiple features as well as their overall enjoyment of the application and some other interesting metrics (Appendix C). The survey was deployed by using Google Forms. \(^1\) Since some of the referenced users with experience in MCDA were located all over the world, the tests could not be made in person. So, all the resources and instructions were sent to the users, which conducted the usability tests themselves. The processed results of the usability tests can be consulted in Appendix D (Table D.1 and Table D.2).

Due to the nature of the application, some care was taken to ensure some of the participants were familiar with MCDA, while others were not (about half of each, of a total of 20 participants). This resulted in a distribution of familiarity as seen in fig. D.1. To deem the tests a success, some objectives were previously set:

• Maximum time to complete the test (task 1 to 3): 30 minutes;
• Minimum average for each of the other metrics, (from 1 to 5): 3.

5.2 Results

By analyzing Appendix D, one can see that the tests can be considered a tremendous success. All the features tested (Table D.2) had scores between 4 and 5 (on a scale of 1 to 5). The average time

\(^1\)https://www.google.com/forms/about/
of completion of the test was about 13 minutes, with the longest test taking 20 minutes. The usability questions (Table D.2) were also very positive, with results in the same range.

The survey also included space for optional, additional feedback. When asked what they liked the most about DecSpace, 8 users reported that they really appreciated the simplicity of it, which is one of the main objectives of the project. Some other users also talked about the intuitiveness and user-friendliness of the framework.

When asked the opposite (what they liked the least), some interesting matters were brought up. A user complained that the workspace was "too empty at the start" - this is a direct consequence of the sandbox approach. Two more users referred that they really enjoyed the notification system implemented, and that it could be used more extensively, namely to notify the user when a new module is generated. This was quickly implemented, resulting in more frequent user feedback messages, like in Figure 5.1.

Another user mentioned some difficulties when connecting modules, due to the small size of the connector boxes - unfortunately, this characteristic is inherent of the chosen approach. A consistency issue with "OK" and "Close" buttons in different windows was discovered by one of the users, which was promptly corrected. The buttons from the workspace toolbar caused some confusion (the chosen icons), but users were able to quickly understand them because of their tooltips (an improvement from previous versions). Some other sporadic suggestions were: more methods, an undo button in the workspace (would require the implementation of a history tracker) and visualization features to better analyze the results, which are already in the roadmap for the near future (not in the scope of this Master’s thesis, however).

### 5.3 Conclusions

Overall, the users were very satisfied with the DecSpace experience. Since during the development the application was continuously tested, with weekly instantaneous feedback from a potential user (very experienced with MCDA), there was a continuous process of bug-fixing and constant improvements. This resulted in very successful tests, with few bugs discovered (and few immediate improvements to make). Thus, most of the suggestions of the users were about larger features that are in the roadmap.
of the project, but were already outside the scope of this thesis.

The users seem to notice the simplicity of the framework, which is indeed one of its main objectives. Also, the test results had no correlation with the degree of MCDA familiarity of the users (the tests did not require MCDA knowledge - only basic methods were used), which means that both experienced and inexperienced users seem to experience the DECSPACE framework similarly - there are not many obstacles for non-experts.
Conclusions and Future Work

Contents

6.1 Conclusions ................................................................. 65
6.2 Future work ............................................................... 65
This chapter presents the conclusions drawn after the work developed for the duration of this Master’s Thesis. A retrospective of what was done is given, along with the inherent relevant contributions for the MCDA community. Afterwards there are some notes on the possible future of the DecSpace project: what the next steps should be, what features could be implemented to improve it, and what direction should be followed to keep the project relevant and raise its weight in the community.

6.1 Conclusions

The importance of DSS is unquestionable, both at the academic level and at the inexperienced users level. These tools are what will make the field grow, by supporting both proven existing methods and new methods that will come up, which only benefit from having software support. Inexperienced users can also be captivated via satisfying and easy to use user interfaces, which will introduce them to MCDA.

DecSpace was presented in this dissertation as a solution to the previously stated problems, although it itself had its own challenges, mainly because of the maintainability of the front-end technologies. A new prototype of the application was developed using Vue.js and cutting-edge JavaScript technologies, which can be considered an improvement from the previous version. A considerable amount of effort was directed into making it maintainable and future-proof, simplifying method implementation so that the project could be scaled up in the near future, allowing the participation of multiple developers at the same time. The application development also took into account the perspective of inexperienced users, verified in the evaluation conducted which surveyed both experts and non-experts.

Several features and usability improvements were implemented, iteratively reviewed by an MCDA specialist. A user interface for an MCDA method (DCM-SRF) was tailor-made directly for its creators, resulting in an ideal and simple card-sorting process - which can be easily understood by non-experts. The user evaluation was conducted with 20 participants of different MCDA backgrounds and was extremely useful for gathering users’ perspective and suggestions, and fixing some small bugs.

6.2 Future work

The developed work is at a stable position and will be immediately continued by other students. The roadmap of the project is well-defined, as new important features will be continuously developed until a public release is possible. Some opportunities are already coming up for testing the framework on a large scale, namely academic partnerships with professors all over the world who want to use the experimental version of DecSpace with their students.

When talking about the biggest steps to take next, XMCDA integration is the obvious one. After the 88th EWG-MCDA, it was agreed that DecSpace would be integrated with XMCDA with close support
from the Decision Deck team, which is very interested in new perspectives towards their open-source technology, currently only interfaced via diviz’s desktop application. By implementing XMCDA support, DECSPACE will instantly gain access to more than a hundred web-services made available by Decision Deck, becoming a much more interesting framework. On the other hand, dedicated methods should still be developed exclusively for DECSPACE so that it does not depend on the XMCDA web-services. The new MCDA developers are already working on new method implementations (e.g., ELECTRE) to increase the number of available methods, taking advantage of the new modular method system.

In addition, the backlog of the project contains many interesting features and usability improvements that will need to be added to the project. Error validation should be perfected even more, to help users and avoid as many mistakes as possible, paired with robust user feedback notifications (one of the most referenced features during the usability tests). Data visualization is another crucial feature to implement, providing the user with alternative visualization forms besides the existing tables. This should be done with D3.js\(^1\), a data visualization library which will easily integrate with the JavaScript-based DECSPACE project.

Finally, before releasing the application to the public, it is essential to implement and improve the documentation of the project on the user side. This means standardizing the method catalogue and finding ways to make it as simple as possible, while also maintaining technical rigor (references to bibliography should be included, as well as practical use scenarios with examples of input, parameters and output. This can be paired with numerous usability and QOL improvements on the roadmap to make a solid and simple application.

A final note to the MCDA Method Selection Tool\(^2\), a tool that helps select MCDA methods tailored to the specific decision problem [21]. This could be an excellent addition to DECSPACE for non-experts that do not know which method is right for them.

---

\(^1\)https://d3js.org/

\(^2\)http://mcda.it/
Bibliography


User Guide
1. Disclaimer

DecSpace is currently in Pre-Alpha. This means that the platform is currently in development, and this is a very early release which still lacks thorough testing, major features and documentation. Taking that into account, please be advised that the platform may have some bugs and breaking issues. We kindly ask that, if you happen to find one of these issues, please let us know, either by using the Contact Us page, or by directly contacting us via email (decspace.dev@gmail.com). One of the missing features right now is Forgot Password, so be careful - if you forget your password, you will not be able to reset your account or create another one with the same email address.

Note: We recommend using the Google Chrome browser for DecSpace.

2. What is DecSpace?

DecSpace is a web-based platform that makes use of MCDA methods to support the user (or decision maker) during the decision process by giving the possibility of building decision models in a visually intuitive way. By having MCDA methods implemented into the platform, DecSpace allows the users to abstract from the implementation and focus on their input and output data.

3. Creating an account

3.1. Go to the DecSpace Pre-Alpha homepage and click SIGN UP on the upper right corner of the screen.
3.2. Fill in the required information and click **REGISTER**. The email field must be a valid email address, and the password must be at least 8 characters long and have at least one digit. You also need to confirm your password. If every field is accepted, you should see a green message confirming the creation of your new user account.

3.3. To login, click **LOGIN** on the upper right corner of the screen. In the login page, fill in the credentials that you just used to create your user account. If the credentials are correct, you should see a green message confirming your login.

3.4. You can now access your **My Projects** page, by clicking on the menu (≡) icon on the upper left corner of the screen, and then clicking **My Projects**. Below is the **My Projects** page. It lists your projects and their information, and allows you to create, duplicate and delete them. To create one, click the green circle with a plus sign, fill in the name of the project and its privacy setting. A private project can only be viewed and edited by you via your **My Projects** page. A public project will appear on the **Public Projects** page, and will be viewable by everyone that accesses it. However, only the creator of the project can edit it. Other users can open, but cannot save it - they can duplicate it as their own copy of the project. You can edit your own projects' name and privacy setting by clicking on the respective cell.

![My Projects](image)

3.5. You can also change your username and password on the **Settings** page, by clicking on the menu (≡) icon on the upper left corner of the screen, and then clicking **Settings**.
4. Workspace: general usage

4.1. After creating a project by following the previous instructions, you can edit it by clicking the respective open icon. You will be directed to the workspace.

4.2. The workspace has many different buttons and options (from left to right):

4.2.1. Execute Workflow: executes methods previously placed on the workspace, in the order that is conveyed by their connections.

4.2.2. Save Project: saves the current state of the project to the DecSpace server. This means that your progress will be saved, and can be accessed anywhere from your account. This button is disabled in anonymous mode and when opening public projects whose owner is not the user.

4.2.3. Import Data File: imports a previously created .csv file to be used as input for methods in the workspace. The header of the file should have the exact same format as the respective method displays in its interface.

4.2.4. Method Selection: chooses from the available MCDA methods a method to be placed in the workspace.

4.2.5. Delete Workflow: deletes everything (modules and connections) from the current workflow. Be careful: after saving the project, this cannot be undone.

4.2.6. Project Menu
4.2.6.1. Import Workflow: imports a previously exported DecSpace workflow. This function expects an unaltered .zip file generated by the Export Workflow function of DecSpace, described below.

4.2.6.2. Export Workflow: exports the current workflow to a .zip file that can be saved in your computer. This file can be sent, for instance, to other people, which can, in turn, import it to their own DecSpace accounts.

4.3. A DecSpace workflow consists of a collection of modules and connections. A module has a name, a delete button, input slots and an output slot. The delete button (X) deletes the module and all its connections. Clicking the name opens the method’s interface. Clicking an input slot opens the method’s interface in the respective tab. To connect a module’s output slot to another module’s input slot (typically we connect data modules to method modules), simply click the first module’s output slot (the white box) and drag it to the desired second module’s input slot. The white boxes should turn green and a connection will appear between the two slots. The data from the first module is now directed into the respective second module’s input slot.

4.4. When you don’t connect your own uploaded data modules to a method’s input slot, and instead edit it directly in the module’s interface, a data module containing your data will be generated and adequately connected for your convenience.

5. Workspace: the DCM-SRF method

5.1. The DCM-SRF method is one of the early available MCDA methods, supported by DecSpace. The Deck Cards Method (DCM) is mainly designed for determining the weights of criteria using the revised Simos’ procedure (“SRF method”). It consists of three steps and three additional parameters. Here is a simple demonstration video of the interaction with the method.

5.2. If you are trying to execute your own DCM-SRF method by manually inserting your data, first start with a clean workspace. Generate the DCM-SRF method by clicking the Method Selection button ( ). You are free to drag and drop...
the module anywhere you want, inside the workspace limits. To start entering your data, click the first input slot Cards, or simply click the module’s name.

5.2.1. In Step 1, start by entering your desired criteria information. A criterion can have a notation (e.g., g1, g2, g3, …), a small code that quickly identifies it (we recommend a 3/4 letter code), its full name, the scale unit, the direction, a description and an icon. The code and name are both required fields - you will not be able to proceed if you have criteria without one of these fields. For ease of use, you can pick an icon from our general selection of icons (Google’s Material Design Icons), which will then be placed on the respective card. You can add as many criteria as you want by clicking the blue NEW CRITERION button. If you made a mistake and need to remove a criterion, click the pink delete button ( ■ ) and it will be removed. When you are satisfied with the entered criteria, click the continue button.

5.2.2. In Step 2, you need to place the criteria cards you have previously created in your desired levels. To do that, drag the cards from the tray above labeled Criteria Cards to any of the levels below, given that the first level is the most important, and the last the least important. You can create and remove as many levels as you need, but remember that every level should have at least one card. To add a level click one of the green add buttons ( + ), and to remove one click the pink delete button ( ■ ). When you are satisfied with the placing of the cards, click the continue button. Any empty levels will be removed.
5.2.3. In Step 3, you will have a deck of blank cards, and can place them anywhere you want, between consecutive levels of criteria you placed before. To do that, drag blank cards from the deck to any of the empty levels that are placed in between the occupied levels. Please note that the criteria cards you placed before are now locked in place - if you need to change anything, click the BACK button. Also note that you are unable to place a blank card in levels that either a) already have criteria cards or b) already have a blank card. When you are satisfied with the placing of the cards, click the continue button. Again, any empty levels will be automatically removed.
5.2.4. Finally, you will be directed to the **OTHER PARAMETERS** tab, which contains three adjustable preference parameters. Ratio Z should be a real number greater than 1. Decimal Places allows you to choose the number of decimal places in the result (0, 1 or 2). Weight Type allows you to choose if you want the results non-normalized, normalized, or both. And you’re done! Whenever you are satisfied with the parameters, you can click **CLOSE** in the lower right corner of the window.

5.3. If, instead, you already have your data in a compatible format (.csv files, with the correct headers), you can make the process much faster. Generate a DCM-SRF module, as described in 5.2. Next, upload your data via the **Import Data File** button ( ). You can pick multiple files at the same time. Click **IMPORT**, and the files will be imported to the workspace as data modules. Place the modules wherever you want by dragging them. Then, connect the data modules to the DCM-SRF module, as described in 4.3. You can open the DCM-SRF module to confirm that the cards are placed as you need. The .csv file to be used as input to **Cards** should have the following format:

5.3.1. If the headers are: Notation, Code, Name, Scale Unit, Direction, Description and Icon, DecSpace will interpret it as being in Step 1.

5.3.2. If the headers are as in 5.3.1 but with an additional Position header, DecSpace will interpret it as being in Step 2.

5.3.3. If the headers are as in 5.3.2 but with one or more lines with the Name equal to “Blank”, DecSpace will interpret those lines as Blank Cards and as being in Step 3.

5.3.4. If in Steps 2 or 3, the Position column should contain an integer portraying the desired level for that criterion/blank card. A valid example of a .csv file in Step 3 can be seen below.
5.4. Finally, to execute the workflow, first double check if every parameter is correctly filled, and then click the blue **Execute Workflow** button ( ▶ ). If everything executes as expected, a data module named “DCM-SRF1 Output” will be generated on the right side of the method module. Else, if there is something wrong with the input, or something missing, you will see a red error message stating what is wrong.
6. Final Notes

Now you know exactly how to fill and execute a DCM-SRF method! We hope everything works out for you as expected! Again, if you encounter any bugs or unexpected behaviour, please take into account that the software is still in an experimental stage. We would really appreciate if you could forward us any problems, suggestions, or general feedback you have to our email address (decspace.dev@gmail.com). Thank you for using DecSpace!

The DecSpace Team
System Evaluation Guide
1 Introduction

DecSpace is a Multi-Criteria Decision Analysis (MCDA) framework currently being developed in Instituto Superior Técnico. This evaluation guide describes a list of tasks and steps for usability testing of the platform, to be used on a Master’s Thesis.

2 Setup

To participate in the test, you need a computer with Internet access and a web browser (Google Chrome is preferred, although you can use another one). Before starting the test, feel free to read the attached user manual, if you want (available here: https://goo.gl/A1iCjc). Please note that you can access it at any given time, even during the test.

First you will execute a set of small tasks, with detailed instructions. You will be using the DecSpace prototype deployed at https://goo.gl/hQN2j2. After finishing the tasks, please fill in the survey at https://goo.gl/18rfk3. For Task 3, you will need to download two files to your computer, available at https://goo.gl/Epc4aF.

3 Tasks

Please access the DecSpace prototype URL referred above. Then, click the blue "Try it now!" button to directly access, and easily use the workspace. Once on the workspace, follow the instructions for each of the tasks below consecutively.

3.1 Task 1

1. Generate an Additive Aggregation module via the Method Selection button on the top right and drag it around to any position you want;

2. Change the name of the module to "My first module" by clicking on the module’s name and editing it;

3. Change the description of the module to anything you want (for instance "The quick brown fox jumps over the lazy dog") by clicking on the second button of the top left corner of the module;
4. Duplicate the module by clicking the first button on the top left corner of the module, then change the name of the new module to "Another module";

5. Delete the original module you renamed "My first module" by clicking the button on the top right corner of the module and confirming;

6. Add a new criterion to the remaining module called "Price", with weight 10, by clicking the "Criteria" input slot and then clicking "New Criterion";

7. Verify that a data module called "criteria" was created and connected to the "Another module" module on the workspace;

8. Delete the connection by double-clicking it, and then verify the data disappeared from the "Criteria" input slot of the "Another module" module, but is still on the "criteria" data module (click the "Input" slot on the data module to open it);

9. Clean the workspace to prepare for the next task by clicking the pink button on the top right of the workspace and confirming.

### 3.2 Task 2

1. Generate an Order BY module via the Method Selection button on the top right and drag it around to any position you want;

2. Manually add the following data to the generated module, by clicking the "Criteria" input slot and then changing to the "Actions" tab:
   
   (a) On the Criteria tab:
   
      i. Unchecked, Price, Number, Ascendant;
      ii. Unchecked, Quality, Number, Descendant.

   (b) On the Actions tab:
   
      i. Apple, 90, 90;
      ii. Samsung, 80, 75;
      iii. Huawei, 50, 70.

3. Return to the workspace and verify that both data modules were generated (criteria and actions), then execute the workflow by clicking the "Execute Workflow" on the top left corner of the workspace;

4. An error message is displayed, because no criterion was picked to be ordered by - fix it by editing the Criteria data and checking the Price criterion on the first column;

5. Execute the workflow again and verify the data on the generated output data module - it should be ordered Huawei, Samsung, Apple;

6. Clean the workspace to prepare for the next task by clicking the pink button on the top right of the workspace and confirming.
3.3 Task 3

1. Generate an Additive Aggregation module via the Method Selection button on the top right and drag it around to any position you want;

2. Upload the two data files you downloaded before by clicking the yellow "Import Data File" button and selecting the files from your computer - two data modules will be generated;

3. Drag the modules to position them in convenient places to connect them to the Additive Aggregation module;

4. Connect the "criteria.csv" file to the "Criteria" input slot, and "options.csv" to the "Options" input slot, by clicking and dragging from the white squares to the respective white squares on the Additive Aggregation module - two green connections should appear;

5. Verify if the data is being correctly passed to the module by opening Criteria and Options tabs in the Additive Aggregation module, and then execute the workflow - it should generate an output data module;

6. Verify if the option with the highest score is "HappyPizza Italian Restaurant", with 1530;

7. Export the workflow by clicking on the blue "Project Menu" on the top right corner of the workspace, and then clicking "Export Workflow" - you can inspect the zip file that was downloaded to your machine;

8. Refresh your browser, which will clean the workspace, and then click once again on the "Project Menu" and "Import Workflow" and select the file you just exported to your computer - the workspace should be restored to the previous state.

4 Final Notes

Thank you very much for participating in this test. Please fill in the quick survey with your thoughts about the platform and the overall usability experience here: https://goo.gl/18rfk3.

5 Resources

User Manual https://goo.gl/A1iCjc
Live Prototype https://goo.gl/hQN2j2
User Evaluation Survey https://goo.gl/18rfk3
Files for Task 3 https://goo.gl/Epc4aF
DecSpace User Evaluation Survey

Thank you for participating. First start by reading and following the instructions in the System Evaluation Guide (https://goo.gl/zwEF6H). Once you are finished, please fill this survey as accurately as possible.

Resources:
User Manual: https://goo.gl/A1iCjc
Live Prototype: https://goo.gl/hQN2j2
User Evaluation Survey: https://goo.gl/18rfk3
Files for Task 3: https://goo.gl/Epc4aF3

* Required

Approximately how much time, in minutes, did it take you to finish the test (from starting Task 1 to finishing Task 3)?

Your answer
Rank the degree of difficulty of using these features: *

<table>
<thead>
<tr>
<th>Feature</th>
<th>Very Difficult</th>
<th>Difficult</th>
<th>Average</th>
<th>Easy</th>
<th>Very Easy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generating new methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drag-and-dropping of modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editing module name and description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duplicating a module</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deleting a module</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manually editing module data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deleting a connection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deleting workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Executing workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uploading data files</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connecting modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Importing and exporting workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rank the degree of usefulness of using these features: *

<table>
<thead>
<tr>
<th></th>
<th>Very Useless</th>
<th>Useless</th>
<th>Average</th>
<th>Useful</th>
<th>Very Useful</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generating new methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drag-and-dropping of modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editing module name and description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duplicating a module</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deleting a module</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manually editing module data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deleting a connection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deleting workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Executing workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uploading data files</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connecting modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Importing and exporting workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is your degree of familiarity with Multi-Criteria Decision Analysis (MCDA)? *

1  2  3  4  5

Not Familiar  Extremely Familiar
DecSpace User Evaluation Survey

How easy was it to learn how to use DecSpace? *

1  2  3  4  5
Very Difficult  ○  ○  ○  ○  ○  Very Easy

How would you classify DecSpace in terms of usefulness? *

1  2  3  4  5
Very Useless  ○  ○  ○  ○  ○  Very Useful

How satisfied are you with DecSpace? *

1  2  3  4  5
Very Unsatisfied  ○  ○  ○  ○  ○  Very Satisfied

Classify the sentence: "DecSpace is an innovative framework with a user-friendly interface" *

1  2  3  4  5
Strongly Disagree  ○  ○  ○  ○  ○  Strongly Agree

Classify the sentence: "DecSpace was very efficient" *

1  2  3  4  5
Strongly Disagree  ○  ○  ○  ○  ○  Strongly Agree

What did you like the most about DecSpace?
Your answer

What did you like the least about DecSpace?
Your answer
Do you have any recommendations for DecSpace?
Your answer

Do you have any other observations regarding DecSpace?
Your answer

SUBMIT

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service

Google Forms
User Evaluation Results
What is your degree of familiarity with Multi-Criteria Decision Analysis (MCDA)?

20 answers

![Figure D.1: Distribution of MCDA familiarity of the users](image)

Table D.1: Degree of difficulty and usefulness of features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Difficulty</th>
<th>Usefulness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generating new methods</td>
<td>4.4</td>
<td>4.9</td>
</tr>
<tr>
<td>Drag-and-dropping of modules</td>
<td>4.5</td>
<td>4.6</td>
</tr>
<tr>
<td>Editing module name and description</td>
<td>4.4</td>
<td>4.3</td>
</tr>
<tr>
<td>Duplicating a module</td>
<td>4.6</td>
<td>4.0</td>
</tr>
<tr>
<td>Deleting a module</td>
<td>4.7</td>
<td>4.7</td>
</tr>
<tr>
<td>Manually editing module data</td>
<td>4.3</td>
<td>4.7</td>
</tr>
<tr>
<td>Deleting a connection</td>
<td>4.6</td>
<td>4.6</td>
</tr>
<tr>
<td>Deleting workflow</td>
<td>4.7</td>
<td>4.2</td>
</tr>
<tr>
<td>Executing workflow</td>
<td>4.6</td>
<td>4.8</td>
</tr>
<tr>
<td>Uploading data files</td>
<td>4.4</td>
<td>4.9</td>
</tr>
<tr>
<td>Connecting modules</td>
<td>4.1</td>
<td>4.7</td>
</tr>
<tr>
<td>Importing and exporting workflow</td>
<td>4.4</td>
<td>4.8</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>4.5</strong></td>
<td><strong>4.6</strong></td>
</tr>
</tbody>
</table>

On a scale of 1 to 5, where 1 is "Very Difficult/Useless" and 5 is "Very Easy/Useful".

Table D.2: Usability questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approximately how much time, in minutes, did it take you to finish the test (from starting Task 1 to finishing Task 3)?</td>
<td>12.9 minutes</td>
</tr>
<tr>
<td>What is your degree of familiarity with MCDA?</td>
<td>3.1</td>
</tr>
<tr>
<td>How easy was it to learn how to use DecSpace?</td>
<td>4.5</td>
</tr>
<tr>
<td>How would you classify DecSpace in terms of usefulness?</td>
<td>4.4</td>
</tr>
<tr>
<td>How satisfied are you with DecSpace?</td>
<td>4.4</td>
</tr>
<tr>
<td>Classify the sentence: &quot;DecSpace is an innovative framework with a user-friendly interface&quot;</td>
<td>4.2</td>
</tr>
<tr>
<td>Classify the sentence: &quot;DecSpace was very efficient&quot;</td>
<td>4.6</td>
</tr>
</tbody>
</table>