**University of Groningen**
Faculty of Science and Engineering

university of groningen

faculty of science and engineering

# A Domain Specific Language for Non-programmers used in order to create questinnaires

Thesis for obtaining Master of Science title in Data Science and Distributed systems

**Author**: Marios Lykiardopoulos(S3490890)

Groningen, April 2019

The Netherlands

## Abstract

Nowadays, questionnaires are used by scientists as a way to collect data by specifically targeted social groups in order to collect and process information about specific issues that they are interested in. A way to collect these data is the use of online questionnaires, as they provide a fast and efficient way to monitor social behaviors and conditions by different social groups. For that reason, plenty of online applications exist in order to shape, render and register the answers exist.

Different web applications which are concerned with this domain are already being used. These kind of applications require a User Interface(UI) which is used by the questionnaire makers in order to define and render the questionnaires. However, these web applications are not a silver bullet as different problems appear such as vendor lock-in which may cause extra costs for the user or even difficulty from moving from one platform to another.

A different solution for questionnaire rendering is the use of a Domain Specific Language. A Domain Specific Language(DSL) is a programming language, which is designed to serve a very specific application domain. The research gap which exists has to do with examining possible differences in usage and understanding among people with different backgrounds on computer science. Furthermore, whether could it be possible to minimize possible drawbacks such as vendor lock-in by designing a platform which could be capable of compiling the questionnaire output into different application backends.

This research investigates the use of a DSL for questionnaire rendering among people with different backgrounds on computer science. Furthermore we provide a study on different questionnaire platforms in order to examine the possibility of a platform which can compile the output to other back-ends.

We present a platform which is based on a DSL and separates the application logic from the code by using a simple syntax in order to help non-programmers to get familiar with the language and the application.

Our findings show that the DSL we designed and implemented was more efficient in terms of time needed for questionnaire rendering compared to other questionnaire platforms, in our case the u-can-act.nl platform. Moreover, no significant differences were spotted among users with different scientific backgrounds.

Based on our results, we can assume that it is possible to design Domain Specific languages and people with very small experience on computer science to work effectively with them, even compared to computer science professionals.

# Contents

## List of Figures

## Listings

# 1 Introduction

Domain Specific Languages (DSLs) started to appear in their early stage around 1970. One of the very first examples of a Domain Specific Language (DSL) was the **Emacs Lisp** which appeared back in 1985 and is a dialect of Lisp programming language and was used by Emacs text editor used for editing purposes.

Over the years, domain specific languages (DSLs) evolved and became more and more popular. Some very well known examples which are widely used mainly by computer scientists are: Structure Query Language(SQL) or Hypertext Markup Language (HTML). Latest improvements allow computer scientists to quickly design and implement their domain specific language by using language workbenches such as MPS, provided by JetBrains.

Domain Specific Languages (DSLs) offer a wide variety of advantages which made them popular over the years. The most important advantages are that DSLs are less expressive compared to General Purpose Languages (GPLs) which deducts the complexity of the language. This makes easier for non-programmers to use DSls instead of GPLs. Moreover, DSLs provide a clear separation between the domain and the actual implementation(code). This helps in order to bridge the gap between the domain experts and the developers.

By taking into account the above advantages it is obvious that DSLs can minimize the gap between computer scientists and people from different application domains. Although there is still a research gap on how people with different scientific background can benefit from DSLs compared to computer scientists.

In this thesis we present a web platform for online questionnaire rendering, based on a DSL which will be used by people working on the psychology department on the University of Groningen. One of the most common ways for psychologists in order to collect data is the use of online questionnaires. This type of questionnaires provide an efficient and cheap solution for scientists in order to collect and analyze data from large social groups. Users can fill in the questionnaires at any time and in any place without the presence of the researcher.

Many applications which allow the formation of online questionnaires exist. Some examples of these applications are: Google Forms, Survey Monkey, etc. These applications provide to the user a Graphical User Interface(GUI) where through some expression builders the user can define the questionnaire. However, such platforms have significant drawbacks such as vendor lock-in or high level of complexity because of the complicated menus and dependencies which they provide.

In this document, we provide an alternative solution which gives the opportunity to the users to create questionnaires through a text based editor interface by using a Domain Specific Language(DSL). Also, we provide a study on how this platform can be extended in a way that will eliminate the vendor lock-in problem by converting the output to different backends.

## 2 Background

The main goal when analyzing data is to extract valuable knowledge in order to gain advantage or improve the existing knowledge on a particular domain. For example, psychologists use questionnaires as an efficient and fast solution in order to gain information from a large amount of people. Questionnaires can provide the means for gaining information about social life, behavior or preferences from a large amount of people.

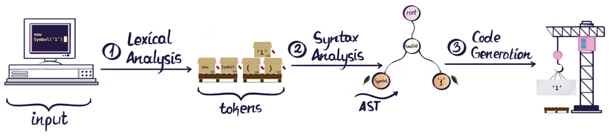
Today, many tools exist in order to format text. One of these tools is Markdown which uses a simple and extremely easy to learn syntax and it compiles it to different output formats such as Portable Document Format (PDF) or Hyper Text Markup Language(HTML). HTML is one of the most fundamental building blocks of the web as it defines the meaning and structure of the web content. One of the key advantages of Markdown is that because of its simplicity it is very easy to be used by non-programmers. Markdown offers a very simple syntax which is based on common symbols for a user. For example, in order to define a header level we use the corresponding number of hashtags which can vary from 1 to 6. Accordingly the user can form different text objects(lists, block-quotes, etc) by using the appropriate symbol.

In our study we look over different online questionnaire platforms such as Survey Monkey, Google Forms, and others in order to examine the process of building an online questionnaire. Two major drawbacks which appear by using these platforms, are: vendor lock-in and reusability. A solution to the addressed problems could be a platform based on a Domain Specific Language(DSL) which is a programming language focused in a specific application domain. In our case this domain is online questionnaire rendering and it can be compiled to different application backends. This solution could help users to minimize the impact of these drawbacks.

## 2.1 Code Generation Pipeline

Code generation is comprised by three main steps: (i)lexical analysis, (ii)syntax analysis and (iii)code generation.
Lexical analysis is performed by the lexer which accepts a stream of text and tokenizes it. By tokenizing we mean that the accepted stream of text is divided into discrete tokens, and an identification for each of the tokens is performed. The output is a new stream of the divided tokens[10].
After the lexical analysis the syntax analysis takes place. Syntax analysis is performed by the parser which identifies if the stream of text(the sequence of the divided tokens) fits the expected grammar. If it fits the grammar then the parser constructs the Abstract Syntax Tree(AST)[10].
The Abstract Syntax Tree(AST) is a tree representation of the source code which is produced by the parser. The root of the tree defines the starting rule of the grammar and each of the nodes defines a grammar rule which can be an expression, term or factor[1]. Image 1 gives a simplified view of the code generation pipeline.

Figure 1: Code Generation Pipeline



## 2.2 Domain Specific Languages

**Definition 2.1** *A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain[3].*

In contrast with the general purpose languages (GPL) such as Java, C, or Python which are designed to be used for writing software and cover a wide variety of application domains, domain specific languages are designed to be used in one very specific application domain. Some examples of domain specific languages which are widely known and used are: Structure Query Language(SQL) which is used for handling data in relational databases, Hyper Text Markup Language(HTML) for creating web pages, Extensible Markup Language for data encoding, Unified Modelling Language(UML) used for visualising the design of a system and Very High Speed Integrated Circuit Hardware Description Language(VHDL) for hardware design.

Domain specific Languages can be split in two main categories: internal and external DSLs. A DSL which is built on top of another programming language is called **internal DSL**. On the other hand a DSL which uses its own syntax and which is not built on top of another language is called **external DSL**.

### 2.2.1 Internal DSL

Internal Domain Specific Language (DSL) is a language which is usually built on top of another language. As a result, the internal DSL has to respect the grammar and the syntax of the original language. This approach comes with some serious advantages but also drawbacks. Some of the advantages are that the user can use the tools which are already available for the original language, the DSL can be implemented on top of another language and the needed time for a developer to produce the domain specific language is significantly less as he can use the already existed infrastructure(parser) of the source language[7]. On the other hand one of the drawbacks which an internal DSL has is that it comes with all the limitations of the original general purpose programming language. Furthermore the syntax of an internal DSL is limited as it should follow the legal syntax of the source language[7].

### 2.2.2 External DSL

An external DSL is making a completely new language. This results new challenges when designing and implementing an external DSL, such as compilation and symbol parsing. One of the main advantages that external DSL offer is the freedom of expressiveness which is not restricted compared to the internal DSL[7]. Furthermore the final result may be much easier to be understand from a person which is a non programmer. On the other hand external DSL require much more effort in order to be built as it requires different programming concepts to be implemented such as a completely new parser.
This analysis between external and internal DSLs will help us in the future of the project in order to examine which of these two solutions we will adopt in our case.

## 2.3 Purpose

The goal of this project is twofold. Firstly we describe the implementation of a platform for building questionnaires. The major components of this platform are a Domain Specific Language (DSL) and a User Interface. The formatting syntax of the Domain Specific Language should be constructed in a way which ensures learnability and easy understanding by the users. Furthermore, the platform should ensure that other programmers would

be capable to extend the already existing syntax or even adjust it based on their individual needs. The aforementioned platform will be implemented as a web application for inputting data and creating questionnaires by wrapping the inputted data into a syntax.
The second goal of this thesis is to provide an extensive research among the different online survey providers in order to examine the process of questionnaire rendering, the different supported question types the text formatting, etc. The architecture of our system is designed in a way which ensures that it will not perform as a single application but can interact with various platforms. This approach can significantly help users in the future to interact with different questionnaire platforms by using one application. Finally we are interested in examining how well our DSL solution performs compared to other online survey editors. For this reason we will conduct an experiment, including users with different backgrounds on computer science in order to collect data and answer our research questions.

# 3 Summary

A Domain Specific Language (DSL) is a programming language which is specialized in a very specific application domain. Instead of General purpose Programming Languages(GPLs) which are broadly applicable to different application domains. Some famous examples of DSls are Structured Query Language (SQL), Hyper Text Markup Language (HTML) and many more [3]. As Martin Fowler explains in his book 'Domain Specific Languages', they can be split into two main categories: internal and external domain specific languages. As internal domain specific language we define the language which is built on top of another language. On the other hand as external DSLs we define a language which uses its own syntax [4] for example XML. Nowadays in many different domains Domain Specific Languages are used in order to solve problems or even simplify procedures. In this thesis we will try to examine how people with different scientific backgrounds can understand and use a Domain Specific Language in the domain of questionnaire building compared to other online survey editors. The research questions which will be answered are the following ones:

- **RQ 1:** Can we reduce the time needed to create a questionnaire by using our DSL compared to other online survey editors?
- **RQ 2:** Are there significant differences in understanding and usage of the DSL between people with computer science background and people with different scientific background?

In order to answer the aforementioned research questions we developed a domain specific language which uses its own syntax, and it was inspired by Markdown.

Furthermore we conducted research between the main online survey editors platforms in order to examine the process they use in order to define questions and extend our platform in order to support the aforementioned platforms also.

# 4 Related Work

This section sums up previous work in the fields of: Involving non-programmers with Domain Specific Languages, online survey editors including text formatting and data format validation. Moreover we provide a comparison between different web based questionnaire platforms which are addressed in the following bullet list. We chose these specific platforms because they share a very big part of the market in online survey editors platforms [15]. Finally we look over u-can-act and Quby, a web based survey platform based on a DSL.

- **Survey Monkey**
- **Survey Gizmo**
- **Lime Survey**
- **Google Forms**
- **Qualtrics**

## 4.1 Involving nonprogrammers with DSLs

One of the most challenging topics nowadays is to help people who are not programmers to use and interact with a high level programming language in order to use it in their domain field. Some examples of these people are engineers, accountants, psychologists, and many more. As Martin Fowler explains the most appropriate choice for a nonprogrammers DSL is to create an external DSL because it comes with two significant advantages. The first advantage is that we can put aside all the baggage of your host language and create something which will be very clear and understandable by a user who is not a computer specialist. Secondly, compared to the internal DSLs it is very easy for the user to be confused because he can do things which make sense in the language but are completely out of the scope that the DSL comes to solve. Furthermore a non-programmers DSL which is designed as an external DSL requires a significant less amount of time for the user to be trained and learn how to use it[5]. Of course designing such a domain specific language that way is not a silver bullet. It requires more effort for designing and implementation but also increases the cost for creating the appropriate tools in order to support the language.

## 4.2 Online Questionnaire Editors

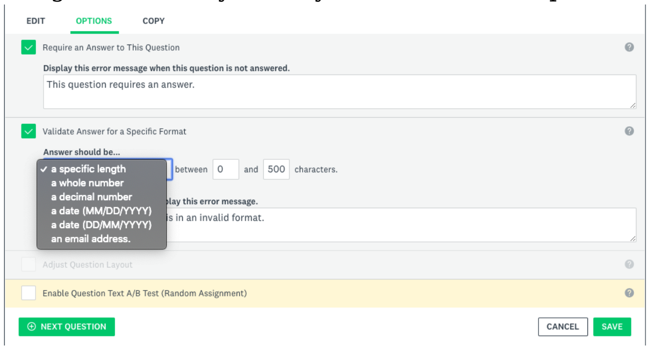
Most online questionnaire editors use a Graphical User Interface(GUI) which is composed by a set of forms which allow the user to define and edit the questionnaires. We will look into two online survey editors: SurveyMonkey and Google Forms. We choose these two platforms because they form a representative sample in the fields of text formatting and data validation which different online platforms also follow. Despite that, in the upcoming sections we will provide a more complete comparison with additional online survey platforms.

### 4.2.1 Text Formatting

When defining a question it may require to adjust the text formatting in order to serve our goal. For example when we want to emphasize we may use a **bold** format instead of a regular one. SurveyMonkey gives to the user this option with a toolbar where the user can select between different text formats. Google Forms allows the user to choose between five different text formats.

### 4.2.2 Data Format Validation

Data Validation lets you define what type of data we want to be entered in the answer field and how these data should be entered. For example when a question requires a date as an answer we must ensure that the user will enter a date information in order to be accepted by the system as a valid answer. As it can be seen from the following images SurveyMonkey lets the user to choose between different options(checkbox, dropdown menu, slider, etc) where the user can choose which is the most appropriate question type. The next step for the user is to validate the answer for the specific format that he chose. The validation options can be seen from Figure 3. On the other hand Google Forms allow the user to choose between different types of question types but the user can not set any further constraints based on the question type that he chose. For example if the question type is Date the format would be **Day, Month, Year** and this format can not be changed.



[✓] Require an Answer to This Question

Display this error message when this question is not answered.

This question requires an answer.

[✓] Validate Answer for a Specific Format

Answer should be...

✓ a specific length
a whole number
a decimal number
a date (MM/DD/YYYY)
a date (DD/MM/YYYY)
an email address.

between 0 and 500 characters.

lay this error message.

is in an invalid format.

[ ] Adjust Question Layout

[ ] Enable Question Text A/B Test (Random Assignment)

Figure 2: SurveyMonkey Data Validation Options

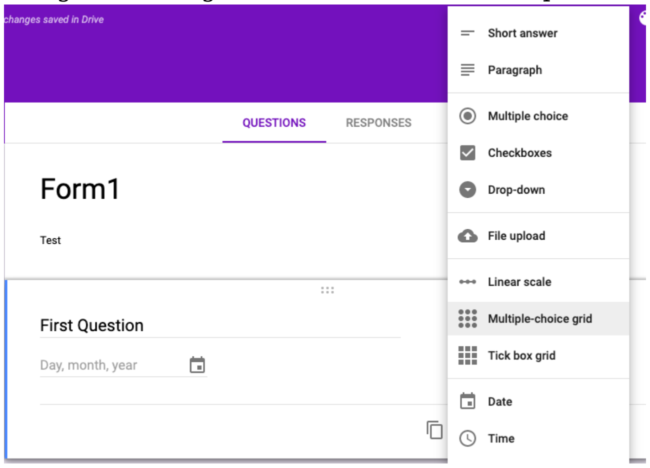


Figure 3: Google Forms Data Validation Options

### 4.2.3 Comparison between online survey editors

In this section we will provide a comparison between the five big players in the field of online survey editors. The comparison was performed among Lime Survey, Survey Monkey, Google Forms, Survey Planet and Survey Gizmo. Table 1 presents the most common question types among these platforms. This comparison can significantly help future developers for extending our platform and make it compatible with different back-ends. Developers can use table 4.2.3 in order to support the common question types for the different platforms. The main question types which are used by the aforementioned providers are described in the following list:

Listing 1: Question Types

- **Array:** An array allows for the creation of sub-questions. Each of these questions uses the same set of answers. This type of questions mainly used when we need to receive feedback for different aspects of a specific product.
- **Multiple Choice:** Multiple Choice questions are one of the most common in surveys. This type of question allows the survey maker to provide the possible answers for the participants. The survey maker can decide whether or not the user can choose one or multiple answers from the provided answers.
- **Mask Questions:** As mask questions we define a category of questions which include question types with predefined answer inputs, such as gender, date, language switch, etc.
- **Free Text:** The question type which is defined as text allows the user to answer the question by collecting the typed text from the user. In this type of question the survey maker can set characters restriction as well as can set the size of the textbox. This type of questions is mainly divided into four smaller subcategories based on the size of the text. These subcategories are the following: short free text, long free text, huge free text and multiple short text.
- **Single Choice/Radio:** The single choice question type allows the user to select only one choice among the provided choices.
- **Score/Ranking:** Scoring questions allow the participants to assign a numerical score to a series of choices. The survey maker can set the lowest and highest value for the scoring process.
  Ranking questions ask the respondents to compare items to each other by placing them in order of preference.
- **Drop-down:** Drop-down questions are close ended questions which allow the participants to select an answer by a list of choices which are presented in a drop-down menu. This type of questions is very similar to the single-choice questions and the main difference is how this question is presented to the user.
- **Image Choice:** Image choice questions are simple close ended questions where the participant is allowed to select an image from an image list which is provided.
- **Date/Time:** A date or time question type is an open text field question with validation which ensures that the correct format was chosen.

| Survey Editors / Question Types | Lime Survey | Survey Monkey | Google Forms | Survey Planet | Survey Gizmo |
|---|---|---|---|---|---|
| **Arrays** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Multiple Choice** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Mask Questions** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Text** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Single Choise** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Scale/Ranking** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Dropdown** | ✓ | ✓ | ✓ |  | ✓ |
| **Image Choise** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Date/Time** | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 1: Online Survey editors Comparison

## 4.3 Survey Platforms

We looked into different platforms in order to examine the process of questionnaire building. More specifically we focus on the use of Application Programming Interface(API) for each of the platforms in order to enter new questionnaires.

### 4.3.1 Quby

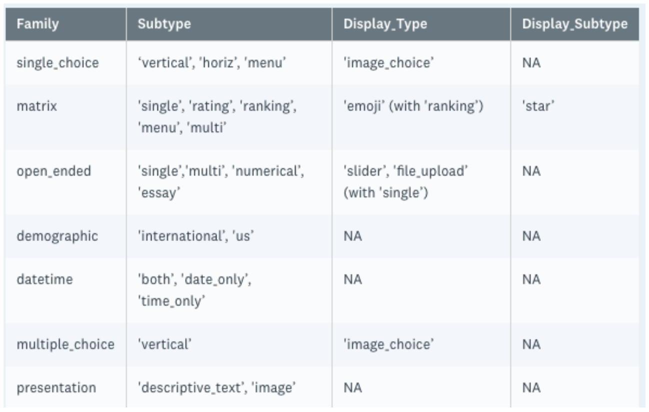
Quby is an online application designed to be used by domain experts, in this case psychiatric research staff in order to form questionnaires, which are filled by psychiatric patients. It comes up with an exposable API which allows the application to communicate with other platforms such as RoQua which provides important information about patients[18]. Users are divided into two main categories: domain experts which define the questionnaires and patients who fill in the rendered surveys. Quby provides an editing environment for users in order to write surveys in the DSL. Furthermore users can preview or test the rendered questionnaires[18]. Regarding the Quby DSL, if we examine it from a conceptual view we can divide it into four different components:Questionnaire, Panel, Question and Score. Each of these components has its own attributes which some of them are mandatory and others are not. For example each questionnaire should have its own 'title' attribute.

### 4.3.2 u-can-act.nl

U-can-act.nl is an open source web application for hosting online questionnaires. It was designed and implemented by the University of Groningen in order to examine the factors which lead to early school leaving. It provides a Graphical User Interface(GUI) where question makers can define questionnaires and participants can fill them out. This open source application is implemented in Ruby on Rails framework. The architecture of the application except from the web server which includes the questionnaire engine(VSV) contains also three different databases each of those stores different types of data. The first stores the questionnaires, the second stores users data and the third includes statistics based on user responses. This application contains also a scheduler which sends invitations to the users for filling out the questionnaires[16]. In our research we will use the u-can-act platform as a case study which will compare to our DSL solution in terms on how fast users can build questionnaires in both of the platforms. Furthermore we connected our service to the questionnaire engine provided by the u-ca-act app as a way to prove that our service does not run as a stand alone application but it can communicate with external back-ends. We choose to connect our platform with the u-can-act questionnaire engine mainly because it is an open source application and can minimize the difficulties and the drawbacks of a closed-source software.

### 4.3.3 Survey Monkey

In order to format questions in Survey Monkey we have two options. We can either use the user interface or we can define questions through API using JSON objects. In Survey Monkey all questions have a type and subtype, which define their type and some of the questions have also a display type and a display subtype which define further their type. Figure 4 shows the possible options for the 'family', 'subtype', 'Display Type' and 'Display Subtype'. Based on the different question types(single choice, multiple choice, image choice, etc) the parameters vary accordingly. In the following listings 2, 25 we present examples of different question types and how the JSON objects look like for these questions.



| Family | Subtype | Display_Type | Display_Subtype |
|---|---|---|---|
| single_choice | 'vertical', 'horiz', 'menu' | 'image_choice' | NA |
| matrix | 'single', 'rating', 'ranking', 'menu', 'multi' | 'emoji' (with 'ranking') | 'star' |
| open_ended | 'single','multi', 'numerical', 'essay' | 'slider', 'file_upload' (with 'single') | NA |
| demographic | 'international', 'us' | NA | NA |
| datetime | 'both', 'date_only', 'time_only' | NA | NA |
| multiple_choice | 'vertical' | 'image_choice' | NA |
| presentation | 'descriptive_text', 'image' | NA | NA |

Figure 4: Survey Monkey question types

```json

{
    "headings": [
        {
            "heading": "Which monkeys would you like as pets?"
        }
    ],
    "position": 1,
    "family": "multiple_choice",
    "subtype": "vertical",
    "answers": {
        "choices":[
            {
                "text": "Capuchin"
            },
            {
                "text": "Mandrill"
            }
        ]
    }
}
```

Listing 2: JSON for multiple choice questions

In the following list we will explain some of the basic JSON parameters that Survey Monkey uses in order to understand their specific role in the process of a question formatting.

- **headings:** The ’headings’ tag defines the heading of a question.
- **position:** The ’position’ tag defines the position of the question in the page and is type of "integer".
- **family:** The ’family’ tag defines the question family and the type of this parameter is a string. The possible options for this parameter can be found on figure 4.
- **visible:** The ’visible’ tag refers to whether or not the question is visible on the User Interface. The type of this parameter is Boolean.
- **required:** The ’required’ tag declares whether or not an answer is required for the question. The type of this parameter is an object.

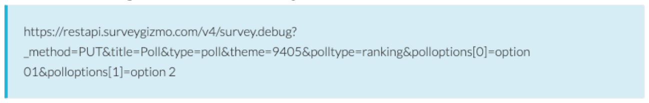
#### 4.3.4 Survey Gizmo

In this subsection we will explain through different examples the process of creating different types of questions using the Survey Gizmo REST API. In the following listing 3 we present different survey objects and the calls in order to create them through the REST API.

Listing 3: Survey Gizmo question types

- **Poll Question:** In order to create a survey object of poll type we need to declare the type of the survey which will be poll and the different poll options of the survey. A call for the aforementioned type of survey is presented in the following figure:

Figure 5: Survey Gizmo Poll question



```
https://restapi.surveygizmo.com/v4/survey.debug?
_method=PUT&title=Poll&type=poll&theme=9405&polltype=ranking&polloptions[0]=option
01&polloptions[1]=option 2
```

- **Image Heatmap Question:** In order to create an image heatmap question we need to use the survey question and survey option endpoints in order to build them. The first step in this process is to create the image heatmap question by declaring the REST method which we will use and it is of $method = PUT$ and then define the required fields which is the following: $type = heatmap$.
  The second step is to define the different options for our image Heatmap question. The REST $method = PUT$ and the required fields are: $title = OptionTitle$ which defines the title of the question and the $value = ReportingValue$.

#### 4.3.5 LIme Survey:

Lime Survey uses a JSON-RPC based web service named LimeSurvey RemoteControl 2(LSRC2) which offers different API functions. A list of the supported features are presented in the following list:

Listing 4: LSRC2 supported features

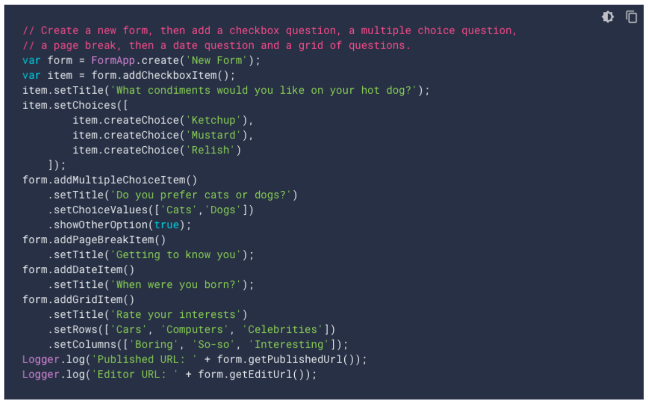
- Start a predefined survey (change titles and things)
- Add predefined groups or questions
- Activate the survey, restrict it to start and endtime
- Add participant data/tokens when you need them
- Return the unused tokens to the main application
- Get a fieldmap for a survey object.

The aforementioned actions can be implemented through different methods from the API.

### 4.3.6 Google Forms

In this section we will explain the process of creating different question types by using the Google Forms API. Google uses Google Apps Script which is a platform that allow users to interact with the different google platforms such as gmail, google sheets, google forms etc. Google App Scripts are scripts written in JavaScript and act on a Google App like the ones we reported before. In figure 6 we present an example which creates a new form and defines different types of questions along with the different options for each of these questions.

Figure 6: Google Forms create new form



```
// Create a new form, then add a checkbox question, a multiple choice question,
// a page break, then a date question and a grid of questions.
var form = FormApp.create('New Form');
var item = form.addCheckboxItem();
item.setTitle('What condiments would you like on your hot dog?');
item.setChoices([
        item.createChoice('Ketchup'),
        item.createChoice('Mustard'),
        item.createChoice('Relish')
    ]);
form.addMultipleChoiceItem()
    .setTitle('Do you prefer cats or dogs?')
    .setChoiceValues(['Cats','Dogs'])
    .showOtherOption(true);
form.addPageBreakItem()
    .setTitle('Getting to know you');
form.addDateItem()
    .setTitle('When were you born?');
form.addGridItem()
    .setTitle('Rate your interests')
    .setRows(['Cars', 'Computers', 'Celebrities'])
    .setColumns(['Boring', 'So-so', 'Interesting']);
Logger.log('Published URL: ' + form.getPublishedUrl());
Logger.log('Editor URL: ' + form.getEditUrl());
```

Summarizing, it is possible to create or modify surveys by using the specific platform that Google provides for developers. A full documentation of all the available classes and methods we can use in order to interact with Google forms can be found in google documentation [6].

### 4.3.7 Qualtrics

Qualtrics survey platform offers the possibility to define questions through an API call to the platform. The way to achieve that is by making a post request to their API and define the expected parameters. The post request includes a json request which includes the necessary data for the action the user wants to perform. The possible actions for the surveys through the API are divided in the following categories which are presented in the following listing:

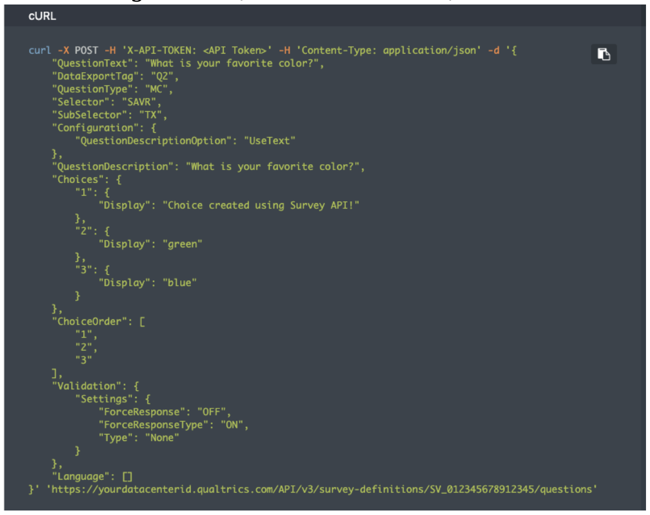
Listing 5: Qualtrics Categories

- **Questions:** The user selects which type of question wants to use.

- **Flows:** The Flow option is where the survey maker defines the correct order of the survey elements and how they are presented to the user.
- **Blocks:** A block is a group of questions which are displayed as a set in the survey. Blocks also help users and survey makers to organize better their surveys especially when these surveys are large.
- **Options:** Survey options gives the right to the user to make changes for different settings of the survey, such as the message which will be printed when the user finishes the survey, the language of the survey or the expiration day of the survey and many more.
- **Survey:** The survey option is divided to three different options. The user can get,create or delete a survey.
- **Languages:** The user defines the available languages of the survey.

In the following figure we present a POST request to the qualtrics API in order to create a new question.

Figure 7: Qualtrics Create Question


cURL

```
curl -X POST -H 'X-API-TOKEN: <API Token>' -H 'Content-Type: application/json' -d '{
    "QuestionText": "What is your favorite color?",
    "DataExportTag": "Q2",
    "QuestionType": "MC",
    "Selector": "SAVR",
    "SubSelector": "TX",
    "Configuration": {
        "QuestionDescriptionOption": "UseText"
    },
    "QuestionDescription": "What is your favorite color?",
    "Choices": {
        "1": {
            "Display": "Choice created using Survey API!"
        },
        "2": {
            "Display": "green"
        },
        "3": {
            "Display": "blue"
        }
    },
    "ChoiceOrder": [
        "1",
        "2",
        "3"
    ],
    "Validation": {
        "Settings": {
            "ForceResponse": "OFF",
            "ForceResponseType": "ON",
            "Type": "None"
        }
    },
    "Language": []
}' 'https://yourdatacenterid.qualtrics.com/API/v3/survey-definitions/SV_012345678912345/questions'
```

## 4.4 Language Workbenches for Defining DSLs

Language workbenches are developer tools for the purpose of defining and developing programming languages. Usually they provide a set of DSLs in order to define different features of languages such as syntax, structure or semantics[11]. One of these tools is **MPS** which is produced by **Jetbrains**. MPS is used for designing and developing languages(DSLs also) in order to be used in the real-world application domains. Its differentiate characteristic is that it provides an editor which supports a wide range of different composition features such as textual, mathematical and graphical notations[11]. More information about language workbenches and especially MPS can be found on [19]; the book provides a detailed explanation about language composition, modularization and extension with MPS. Other languages workbenches similar to MPS are for example: Intentional, MetaEdit which is focused on graphical modelling or Spoofax which is mainly used for textual languages.

## 4.5 Summary

We expect that new developers which will extend our work will use the comparison between survey editors in order to understand how the different APIs work and what programming concepts they should follow in order to make compatible our application with the different questionnaire platforms. Furthermore we propose some language workbenches which can be used in order to deploy domain specific languages or extend already existed ones.

# 5 System Design

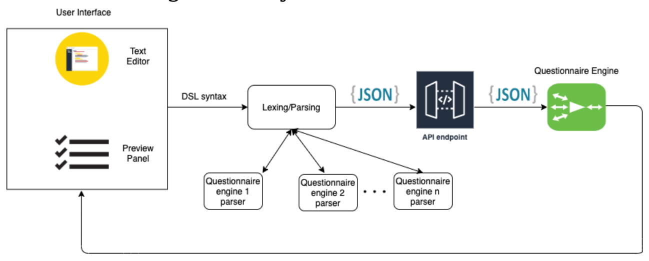
We propose a platform which is designed in order to communicate with different questionnaire platforms and interact with them. Our solution results in a great advantage that it is possible to minimize drawbacks which many users face by using other platforms sucha as vendor lock-in and reusability. We designed a platform in which domain experts can define questionnaires and users can fill them in. Our application is not designed to work standalone, instead it is served as a single point of access to interact with various questionnaire platforms. An overview of the architecture is shown in figure 8.

We provide a text editor where the domain expert can define the questionnaire by writing in the DSL. Our application exposes an API which can communicate with different questionnaire engines. In our case it communicates with the VSV questionnaire engine where we send the questionnaire parsed into JSON and receive the questionnaire in form of HTML.

Our platform works as a micro-service, as a part of a system which is already implemented and used. This is achieved by a web exposed API which can interact with various platform according to the user needs. Microservices are defined as a specific architectural style that structures an application as a collection of services which should follow the basic characteristics as they described below: highly maintainable and testable, loosely coupled, independently deployable, organised based on business capabilities and owned by a small team[12].

The platform is consists of three components: (i) a UI component, (ii) a parser, and (iii) the questionnaire engine. User Interface (UI) allows the user to interact with the system. Parser is responsible for taking the stream of tokens and converts it into an Abstract Syntax Tree (AST), which represents the structure of our DSL. The third component of our platform is the questionnaire engine which is responsible for the questionnaire rendering. Figure 8 depicts the architecture of the system.

Figure 8: System Architecture



To allow domain experts produce questionnaires in our application we provide a web editor. Inside that editor the user can write questionnaires in the DSL. Then the questionnaire which is written in the DSl is parsed into a JSON object which acts as an intermediate object and through the API it is sent to the questionnaire engine which is selected by the user. The questionnaire engine renders the questionnaire and send it back to the application in order to be displayed for the user.

The Domain Specific Language (DSL) we created is designed in a way which ensures extensibility and flexibility to third party developers in order to extend or modify the existing language. Extensibility is achieved by the fact that a third-party developer can easily add new keywords, concepts and syntax rules to our source language.

The syntax was designed in a way which ensures that is easily extensible, flexible and usable for people who are not professional programmers.

## 5.1 Syntax Design

The top priority when designing the syntax was to be easily understandable by new users. In order to achieve that we need to minimize the syntax amount in our DSL. The decision we took was to use different symbols which a non-programmer uses in his/her everyday life. For example many users use every day the hashtag symbol in their social network activity. A tool which played a significant role in the design of our syntax is the Markdown language. Markdown is a web tool for converting text to HTML. Its syntax is based on the philosophy that is easy for the users to read, write or edit. Taking this idea a step further we decided to design our syntax based on this pattern. We used plenty of symbols which are used also by Markdown language. This decision ensures that our syntax will be easily understandable and usable by new users especially for those who already have an experience with Markdown language.

# 6 Analysis

In this section we will explain and analyze the whole process of implementing the Domain Specific Language. We will also refer all the technical decisions we take in order to deploy the system. Furthermore we will provide some useful examples that we think are necessary for the reader in order to understand the whole implementation process.

## 6.1 JSON object

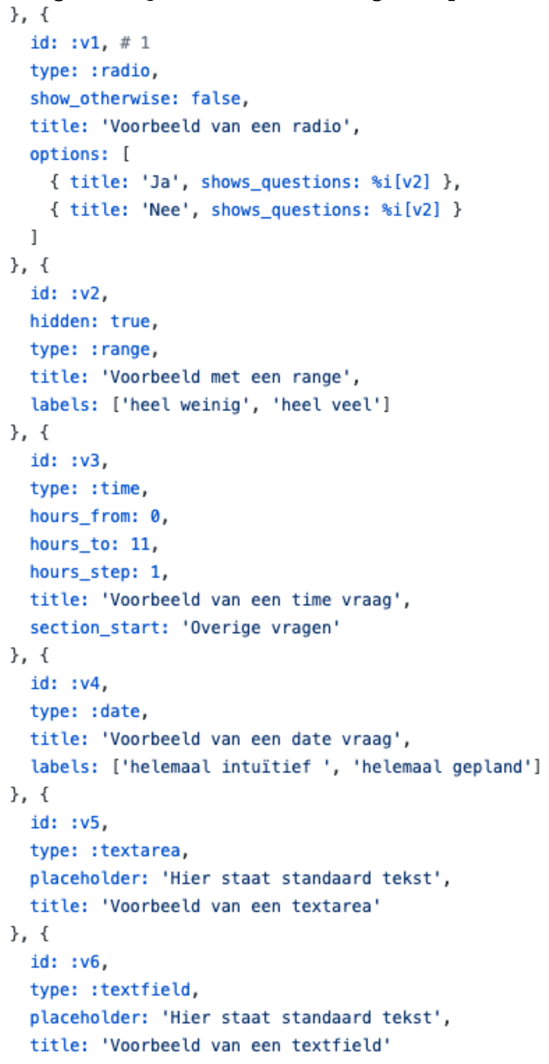
As we described in the previous chapter the output of the parser will be a JSON object which will act as an intermediate that links the syntax with the questionnaire engines.
To provide a structure to our intermediate object we used the JavaScript Object Notation(JSON). JSON is a data format which is widely used for data exchange[8].

## 6.2 Questionnaire Design

Each questionnaire is comprised by different questions. Depending on the question type each question has its attributes which may differ, depending on the question type and the dependencies between questions. The content of the questionnaire is an array which stores the questionnaire definitions.

Figure 9 presents the specific format of the JSON object which is the questionnaire syntax that the VSV engine accepts in order to render the questionnaires.

Figure 9: JSON for rendering the questionnaires



```
}, {
  id: :v1, # 1
  type: :radio,
  show_otherwise: false,
  title: 'Voorbeeld van een radio',
  options: [
    { title: 'Ja', shows_questions: %i[v2] },
    { title: 'Nee', shows_questions: %i[v2] }
  ]
}, {
  id: :v2,
  hidden: true,
  type: :range,
  title: 'Voorbeeld met een range',
  labels: ['heel weinig', 'heel veel']
}, {
  id: :v3,
  type: :time,
  hours_from: 0,
  hours_to: 11,
  hours_step: 1,
  title: 'Voorbeeld van een time vraag',
  section_start: 'Overige vragen'
}, {
  id: :v4,
  type: :date,
  title: 'Voorbeeld van een date vraag',
  labels: ['helemaal intuïtief ', 'helemaal gepland']
}, {
  id: :v5,
  type: :textarea,
  placeholder: 'Hier staat standaard tekst',
  title: 'Voorbeeld van een textarea'
}, {
  id: :v6,
  type: :textfield,
  placeholder: 'Hier staat standaard tekst',
  title: 'Voorbeeld van een textfield'
```

As it can be easily examined from the above figure 9 each question is determined by a set of elements such as the id, type, title, placeholder, options and many more. It is very important to investigate the format of each question type because based on this format we will design and implement the parser which takes the stream of text and produces the output.

In the following listing we will explain the role of each of these basic elements and in the following image we will present the results of the questionnaire engine when it accepts this JSON file.

- **id:** Each question should have its own id which is unique for every question.

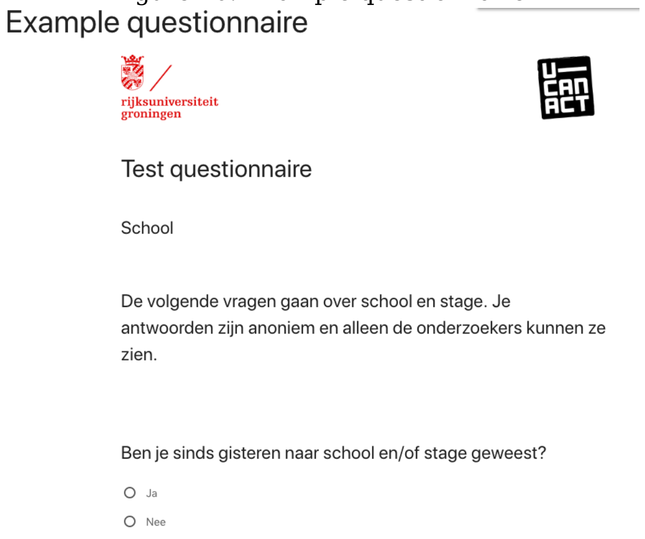
- **type:** The type element determines the type of the question which will be rendered. The question types which are supported are the following ones: checkbox, radio, range, raw, textarea, textfield, expandable, time, date and dropdown.

- **title:** The title in every question determines the main body of each question.

- **options:** Options element determines an array which can contain either hashes or strings.

In the following image we can see how the rendered questionnaire looks like when it is created by the VSV engine.

Figure 10: Example questionnaire



Example questionnaire

Test questionnaire

School

De volgende vragen gaan over school en stage. Je antwoorden zijn anoniem en alleen de onderzoekers kunnen ze zien.

Ben je sinds gisteren naar school en/of stage geweest?

○ Ja

○ Nee

The question types which are supported by the u-can-act platform are the following ones, presented in the following listing 6:

Listing 6: u-can-act supported types

- **Checkbox**

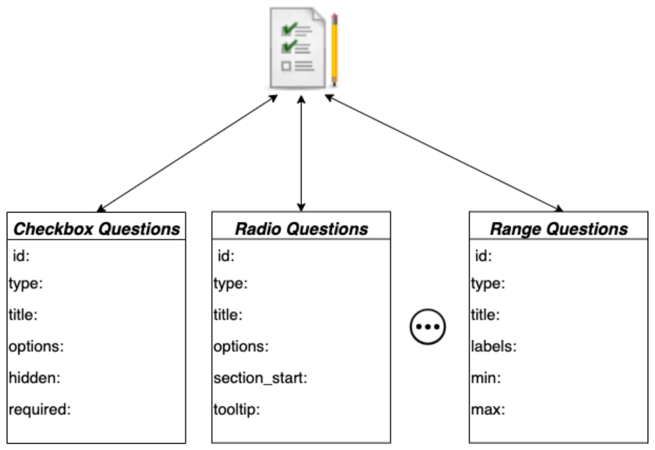
- **Radio**

- **Range**

- **Raw**
- **Textarea**
- **Textfield**
- **Expandable**
- **Time**
- **Date**
- **Dropdown**

Each of the above question types have their specific parameters which they differ from question type to question type.
Figure 11 presents a diagram of the questionnaire design along with the different types of questions and their individual parameters.

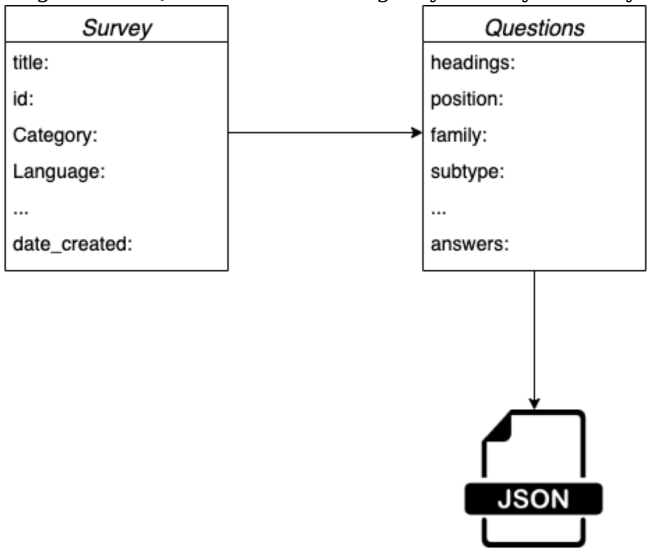
Figure 11: Questionnaire design



### 6.2.1 Survey Monkey Design

Except from the VSV engine we looked into the Survey Monkey questionnaire platform in order to examine the questionnaire structure which they support. We will go deeper in the syntax design of the Survey Monkey platform. We will present through the upcoming figure 12 the syntax design of the Survey Monkey surveys which was necessary to study in order to proceed for the next steps of the project which is the design of the Domain Specific language.

Figure 12: Questionnaire design by Survey Monkey



Survey Monkey questionnaires are divided into two main components: (i)survey, (ii)questions, each of these components has its own attributes. The first component is the survey and is characterised by the **title**, **id**, **category**, **language**, etc. When the aforementioned parameters are defined the next step is to define the questions of the survey. This is the second component which includes parameters such as the **headings**, **position**, **family** and others. The role for each these parameters is explained analytically in section 4.

# 7 Implementation

Our implementation focuses on developing a generic solution that could interact with multiple questionnaire engines from different platforms. In the following sections we will explain the tools we used in order to implement our platform as long as with some important code snippets.

## 7.1 PEG.JS Parser Generator

Peg.js is a parser generator which uses specific grammar rules that are very close to the JavaScript Programming language. Given a set of rules which consist the grammar of the language, the parser accepts a stream of text and if this stream fits to the defined grammar the Abstract Syntax Tree is constructed and this leads to the code generation. Peg offers a set of significant advantages with the most important to be: (i) performs both lexical analysis and parsing, (ii) provides a simple and extensive grammar, and because it is based on parsing expression grammar formalism it is much more powerful compared to traditional parsers which are based context free grammars. Furthermore peg.js is usable from the browser or through a JavaScript API[13].

The grammar is consisted by different rules. Each of the rules has a unique name which identifies the rule and a parsing expression which defines a pattern to match against the input text. It may also contains some Javascript code that determines what happens if the pattern matches successfully [13]. The parsing start from the first rule which is also called 'start rule'. An example of a simple rule can be found in the following listing.

```
start
  = additive

additive
  = left:multiplicative "+" right:additive { return left + right; }
```

Listing 7: Example of a simple start rule

As it can be seen from the above listing a start rule named additive is defined. This rule takes as inputs two numbers and when the "+" sign is recognised it computes and returns the sum of these numbers. Except of the rules there are also some others essential components when we define a grammar with peg.js. These components will be explained in the following list.

- **Rules:** As we described, a rule matches a parsing expression of a rule recursively and returns its match result. Each rule should be unique and followed by an equality sign("=") and a parsing expression which defines a pattern to match.
- **Start Rule:** A start rule is the first rule which is being processed. This block of code is executed before the generated parser starts parsing.
- **Expressions:** Many expressions can be used in order to determine a sequence of occurrences in a matched syntax. For example if we have an expression of type [3,4] and we want to match this expression as a point the code block will look like as in the following list example.

```
point
= "[" left:value ws right:value "]" {
    return {
        type: "point",
        x:left,
        y:right
      }
    }
```

Listing 8: Example of a simple expression

- **Character classes:** Character classes are used in order to define a set of characters and match any of these characters . An example of a character class can be found in the following listing.

```
alphanumerical:[A-Za-z0-9\]+
```

Listing 9: Example of a simple Character class

  The aforementioned character class example tries to match an alphanumerical sequence of characters and name it as "alphanumerical."

These examples help us understand on how to define the rules which form the grammar for a language.

## 7.2 User Interface

We want users to be able to easily input new questionnaires in a textual format. In order to achieve this goal we should design and develop a User Interface which should have a high level of usability. By usability we mean how easy is for the users to reach their objectives by using the web interface. Of course the term usability is composed by others sub-factors such as **efficiency**, **learnability**, **effectiveness** and **colour using**. These factors should be taken seriously into account when designing a user interface. In the following listing we will explain each of these factors we mentioned above:

Listing 10: User Interface key points

- **Efficiency:** By the term ’efficiency’ we mean that the user is able to complete a task with the minimal effort. In our case the goal for the user is to produce the correct json output depending on the platform which they want to use, either the u-can-act or the Survey Monkey platform.
- **Learnability:** By the term ’learnability’ we define how easy is for the user to get used to the GUI. This key point is very crucial for the users because it can reduce significantly the time needed for the users in order to reach their objectives.
- **Effectiveness:** One of the most important key drivers for a well designed GUI is how effective this GUI is. By effectiveness we mean how well it performs the tasks which is designed to perform.
- **Colour using:** Another significant factor is the use of colours which the designer of the GUI decide to use. For example the use of bright

colours helps to give emphasis on specific points in the User Interface but simultaneously may be frustrating for the users especially for those who use the User Interface for a long period of time.

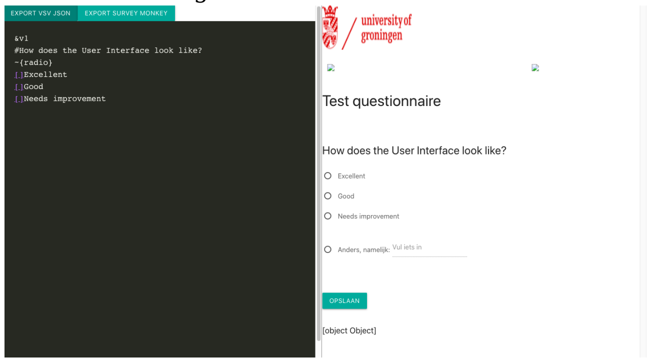
### 7.2.1 React JS

Our GUI was designed and implemented with ReactJS. which is a JavaScript library which gives the developers the opportunity to create fast and highly dynamic web-pages. Our decision to use React is based on the advantages which offers and more specifically to the reactive way on displaying data.

### 7.2.2 UI Components

Our GUI can be divided into two main components. The first component is the ’editor’ panel which is used by the user in order to define the questions by writing them in our DSL. The second component of the UI is the ’view’ panel where the user can see the rendered questionnaire.
In the following image we present an overview of how the User Interface looks like. On the left side the ’editor’ panel exists while in the right side there is the ’view’ panel.

Figure 13: User Interface



```
&v1
#How does the User Interface look like?
~{radio}
[ ]Excellent
[ ]Good
[ ]Needs improvement
```

Test questionnaire

How does the User Interface look like?

- Excellent
- Good
- Needs improvement
- Anders, namelijk: Vul iets in

[object Object]

## 7.3 DSL Syntax

Our DSL is designed to be used mainly by people who are not developers or computer scientists. Our philosophy is that we should design a language which will be easy to read and easy to write for the users. Based on these remarks we set that the top priority when designing the syntax is to be understandable and intuitive by a non programmer but also to ensure extensibility for users with programming experience who want to modify it or extend it.

We tried to design a syntax which follow similar syntax as the Markdown language, explained in section 3. Its primary operators are symbols such as the number sign, brackets, curly brackets etc. Each of these symbols has a special usage and specifies a unique element.

For example when the user writes &v1 this is recognized by the parser as id:v1. The syntax is interpreted sequentially and designed to be user friendly for non-programmers. Some of the attributes are not compulsory for the user to be determined and in this case the value which is assigned to these elements is null. In the following listing we explain the syntax of the DSL and how it is interpreted by the parser. We present the syntax we defined and the type of data which should follow our syntax.

- **&alphanumerical**: The value which follows the ampersand symbol is parsed as the "id" of the question. For example when the user types the following syntax &v1 the parser renders the following output **id:v1** in terms of JSON syntax. The id of the question is compulsory to be declared by the user.

- **#Text**: After the hashtag symbol, the user can define the title of the question.

- $\sim \{Text\}$: Inside the curly brackets the user can define which question type to use. The supported question types are the following: **checkbox**, **radio**, **range**, **raw**, **textarea**, **textfield**, **expandable**, **time**, **date**, **dropdown**.

- **[]Text**: Next to the brackets the question maker defines the available options for the user in order to choose one or multiple of them, depending on the question type. The question maker can use the options notation as many times as the available answers are.

- **$Text**: The dollar sign matches the placeholder of the question.

- $*$**Text**: The asterisk notation denotes the possible labels on a question. We can have multiple labels which are separated with comma.

- **%true/false**: The text that follows the title of the question is recognised by the parser as the **required** tag. The required tag can take only two possible values. It can be either 'true' or 'false'.

- **@min:Number:** Defines the minimum value in a range of numbers. It is mainly used in number question types.

- **@max:Number:** Defines the maximum value in a range of numbers. It is mainly used in number question types.

- **@ml:Number:** Defines the maximum length value that the user can enter as a response to the question. It is mainly used in number question types.
- **+hf:Number:** Hours from(hf) is used in time question types and specifies the value where the drop-down will start.
- **+ht:Number:** Hours from(hf) is used in time question types and specifies the value where the drop-down will start.
- **+hs:Number:** Hours to(ht) is used in time question types and specifies the step size between hours from and hours to.
- **Raw Text:** The user can add also raw text or HTML code in the questionnaire if needed in the questionnaire.

## 7.4 Survey Monkey DSL syntax

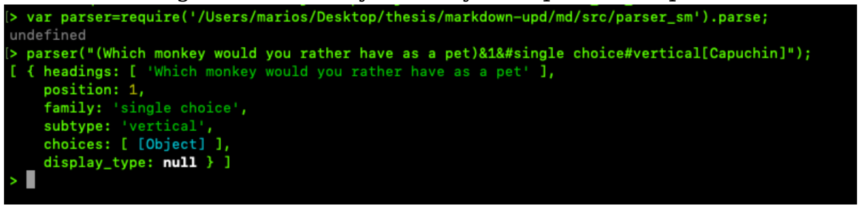
In the previous section we presented the syntax design and the output based on the VSV questionnaire engine which is used by the u-can-act platform. In this section we will present the DSL syntax we designed and implemented in order to adjust it to the Survey Monkey platform. With this implementation we give the option to the user to use our implementation in order to define questionnaires through the Survey Monkey API.
In order to serve usability we used a very similar DSL syntax as the one we used before but the output is different based on the needed format for the Survey Monkey platform as it is described in section 3.4.1 .
In the following listing we describe the syntax along with the rendered output.

- $(\boldsymbol{Text})$: The text which is inside the parenthesis denotes the **heading** the question.
- **&Number&**: The number between the 'and' symbols is parsed as the "position" and denotes the position of each question in the questionnaire.
- **#Text#**: The text between the 'hash' signs denotes the 'family' of the question which determines the type of the question which will be rendered.
- **Subtype**: The text which follows the 'family' question type is recognised by the parser as the 'subtype' of the question which specifies further the type of the question.
- $[\boldsymbol{Text}]$: The string inside the brackets is parsed as the choices for each question which is the list of the available choices for the user.

- **{*Text*}**: The string inside the curly brackets is parsed as the 'display type' of the question. The 'display type' is an optional field which further define the question type.

Image 14 shows how the grammar is parsed based on the Survey Monkey questionnaire platform. The grammar is not complete as it can be extended based on the Survey Monkey API but it is a good base for developers to extend it in the future and make it compatible with Survey Monkey.

Figure 14: Survey Monkey DSL parser output



```
> var parser=require('/Users/marios/Desktop/thesis/markdown-upd/md/src/parser_sm').parse;
undefined
> parser("(Which monkey would you rather have as a pet)&1&#single choice#vertical[Capuchin]");
[ { headings: [ 'Which monkey would you rather have as a pet' ],
    position: 1,
    family: 'single choice',
    subtype: 'vertical',
    choices: [ [Object] ],
    display_type: null } ]
>
```

Through our Survey Monkey implementation we proved that our platform can be extended to work also with different back-ends. Actually no current connection exists between Survey Monkey and the DSL but future developers can use our work as a basis in order to extend and connect it to different platforms.

## 7.5 Defining the Syntax Rules

The new language is consisted by rules. Each of the rules matches the parsing expression and returns the matched result.
In the following listing we can see how we defined the basic rules for the Domain Specific Language.

Listing 11 presents three functions of the parser. The first function converts its first argument into a string, parses the rendered string and finally returns an Integer or a null value. Some of the question attributes such as the id and the title are compulsory for each type of questions except the raw type. Other attributes such as the labels or the placeholder are not compulsory and used depending on the type and the complexity of the question. If these attributes are not needed in a question we assign them the null value. The second function is one of the most important functions as it is the function that we specify all the possible attributes that consist a question. These attributes are for example the id, title, type etc. The third function is called 'removeDuplicateSpaces' and takes as input the typed text by the user and is responsible to remove double spaces which may be typed by the user.

```
{
  function makeInteger(o) {
    return parseInt(o.join(""),10);
}
function createQuestion(id, title, type, options, placeholder, labels, required
     , min, max, ml, hf, ht, hs) {
  var question = {
    id: id,
    title: title,
    type: type,
    options: options,
    placeholder: placeholder,
    labels: labels,
    required: required,
    min: min,
    max: max,
    maxlength: ml,
    hours_from: hf,
    hours_to: ht,
    hours_step: hs
  }
  return question;
 }

 function removeDuplicateSpaces(text) {
    return text.replace(/\s\s+/g,'');
  }
```

Listing 11: Create question Example

Code listing 12 shows the separation between two categories of questions. The first category includes the raw type questions where the user can directly give as input raw text or HTML code and the parser directly recognise the question type and matches the input to the grammar.
The second category consists of questions where the user should define the id of the question, the title and the type by using the DSL syntax.
Furthermore through the code snippets we can understand how the rules are defined in order to match the input stream to the defined grammar.

```
QuestionsType
  = questions:(Question/QuestionCont)*  { return questions; }

QuestionCont "(question with content)"
  = content:Raw


Raw 'rawtype'
   =  content:RichText QuestionSeparator  {return{type:"raw",content}}


Questions
   = _ questions:(Question)* _ { return questions }

Question '(question)'
  = QuestionAnswersAtEnd
```

```

 QuestionAnswersAtEnd "(question with answers at end)"
  = id:QuestionId _ title:QuestionTitle _ type:QuestionType? _ options:
      QuestionOptions? _ placeholder:QuestionPh? _ labels:Questionlab? _
      required:Questionreq? _ min:Questionmin? _ max:Questionmax? _ ml:
      Questionmaxlength? _ hf:QuestionHoursFrom? _ ht:QuestionHoursTo? _ hs:
      QuestionHoursStep? QuestionSeparator
  {
    var question = createQuestion(id, title, type, options, placeholder, labels
        , required, min, max, ml, hf, ht, hs);
    return question;
  }

QuestionId
  = '&' id:RichText  {return id}

QuestionTitle
  = _ '#' title:RichText  { return title }


QuestionType
  = _ '~{' type: RichText '}' {return type}

QuestionOptions "Options"
  = options:(QuestionOption) + { return options; }

QuestionOption "Option"
  = '[]' option:(RichText)
    { return option; }
```

Listing 12: Defining Question Elements

These rules were defined in a way which ensures usability for question makers as they can use simple every day symbols in order to build questionnaires, but also extensibility for future developers to extend or modify the already existed rules.

Code listing 13 defines the different character classes we used in our grammar. A character class defines a group of characters and matches any single character of the set.

```
RichText
  = Text  { return text().trim('') }

Number
    = chars:[0-9]+ frac:NumberFraction?
        { return parseFloat(chars.join('') + frac); }

NumberFraction
    = "." chars:[0-9]*
        { return "." + chars.join(''); }

_ "(whitespace)"
  = (EndOfLine !EndOfLine / Space / Comment )*
```

```


Comment "(comment)"
  = '//' (!EndOfLine .)* ( EndOfLine / EndOfFile)

Space "(space)"
  = ' ' / '\t'
EndOfLine "(end of line)"
  = '\r\n' / '\n' / '\r'
EndOfFile
  = !. { return "EOF"; }
```
Listing 13: Defining the character classes

From the above listing 13 we can observe the process of defining the different character classes such as the **Text**, **Number** or **Number Fraction** but also other classes such s the **end of line** or the **end of file**.
The process of defining the character classes for the Survey Monkey platform is very similar to the process we described above.

## 8 DSL vs u-can-act platform

A comparison between the DSL platform and the u-can-act platform was performed in order to collect and analyze data to answer our research questions. We asked from users with different experience levels on computer science to build a questionnaire by using our platform and the u-can-act. We keep track of the time needed to build the same questionnaire in both of the platforms and we analyzed the data we collected. Through this analysis we will be in position to justify whether or not a DSL can reduce the needed time for questionnaire building compared to other platforms but also how easy is for non-programmers to understand and use DSLs.

### 8.1 Hypothesis

The aim of this study is to explore whether a DSL related platform for building questionnaires constitutes a development over a JSON related platform. We hypothesised that the DSL solution can reduce significantly the necessary time in order to build questionnaires. The above assumption comes from the fact that most of the users are not familiarized with JSON syntax. On the other hand a Domain Specific Language which is based on a user friendly syntax can provide us better results in terms of time. We believe that the results of our experiment can also show that DSLs can also bridge the gap between developers and domain experts. The acceptance of this assumption can significantly boost the usage of DSLs in the future as both developers and domain experts can both benefit from DSLs.

## 8.2 Experiment Procedure

We performed an experiment in order to measure how fast our DSL performs compared to other online survey editors(u-can-act). Intended participants are users with different experience in computer science. The participants are divided into three main categories based on their computer science background. The first category includes participants with a high level of computer science, the second group includes participants with a low level background and the third group includes users which already use the `app.u-can-act.nl` platform. All the participants were asked to build the same questionnaire 16 which includes seven different questions, by using both of the platforms.
Before the experiment starts, a manual was provided to the users which describes how both of the platforms work, along with examples and a brief description of the goal of the research. After participants read the manual they could ask questions in case that something was unclear or they need further clarifications about the platforms.
The next step was to ask the participants to start the experiment by starting building the questionnaire. During the experiment we keep track of the time needed for the participants to build the questionnaire by using both of the platforms in order to examine if our DSL solution constitutes a development, compared to the application which is already being used. When the aforementioned procedure is completed the times are recorded and will compared in order to examine the differences between the platforms and the users.
Our experiment acts as a case study in order to collect data and justify our hypothesis. In the future it can be extended by including more questionnaire platforms which can give us a more complete picture on how our DSL acts compared to other platforms.

# 9 Analysis Methods

## 9.1 ANOVA

Analysis Of Variance (ANOVA) is a set of statistical tests, used in order to examine whether or not there are significant differences between the means in a sample[2]. This method is mainly used in the analysis of experimental data. Anova is based on accepting or rejecting the null hypothesis. The null hypothesis is that between group means there are no statistical significant differences. A result is considered as statistically significant when a probability $p - value$ is less than a predefined threshold, which justifies the rejection of the null hypothesis[2].
In statistics, the null hypothesis is a general statement that there is no relationship between two measured, or no association between groups. For ex-

ample, in a typical ANOVA scenario where we compare two different treatments the null hypothesis would be that both of the treatments have the same results. By rejecting the null hypothesis we accept that the observed differences between the groups are unlikely to be due to random chance. In our case the null hypothesis is stated as following: There are no significant differences in the time means for the DSL platform and the u-can-act platform.

### 9.1.1 Assumptions

In order to apply the ANOVA test we that data should follow three main assumptions as they described in the following listing.

- **Normality:** Each of the groups which are compared should follow the normal distribution.
- **Independence:** The dependent variables should not be affected by the conditions in other samples.
- **Variance:** Samples should come from populations with the same variance.

### 9.1.2 Checking Assumptions

As we described in the above section in order to apply the ANOVA test the above assumptions should be fulfilled. In order to check for these assumptions we will apply two tests, to check for the assumptions. The first test is the Shapiro-Wilk test which checks for the normality and the second test is the Levene test which checks for the variance assumption. In the following listing we give an overview for both of the tests.

- **Shapiro-Wilk:** The Shapiro-Wilk test tests the null hypothesis that a sample $x_1...x_n$ came from a normally distributed population[14]. The null hypothesis means that the population is normally distributed. If the $p - value$ is less than a specific $\alpha$ value, the null hypothesis is rejected, meaning that the data come from a non-normally distributed population. On the other hand if the $p - value$ value is greater than the $\alpha$ value then the data come from a normally-distributed population[14].
- **Levene's test:** Levene's test the null hypothesis that the population variances are equal. If the $p-value$ is less than a specific value(typically 0.05) then the null hypothesis is rejected, meaning that the the population variances are not equal. On the other hand if the $p - value$ is greater than the significance level(0.05) the null hypothesis is valid[9].

### 9.2 Turkey's Honestly Significant Difference

If the ANOVA results show that there is evidence that the means of the population differ significantly, we need to examine which of the means are different. This is where the Turkey's test is used. This test compares the difference between each pair of means with the necessary adjustment in order to perform multiple testing[17].

# 10 Results

In this section we will present the results from the analysis we performed after the data collection through the experimental procedure.

## 10.1 Datasets

### 10.1.1 ANOVA Dataset

The ANOVA dataset is organised in a way in order to be able to perform the ANOVA test. The dataset is organised in 4 different columns based on the user's computer science background and the application they are tested. The four columns are explained in the following listing:

- **CSExpAPP:** This column includes the time records from the users with high computer background which they tested the `http://staging-app.u-can-act.nl/questionnaire/interactive` platform.
- **CSExpDSL:** This column includes the time records from the users with high computer science background, testing the DSL platform for the questionnaire rendering.
- **NoCSExpApp:** In this column the included time records are from users with low level of computer science background, testing the `http://staging-app.u-can-act.nl/questionnaire/interactive` platform.
- **NoCSExpDSL:** This column includes the time records from users with low computer science background, testing the DSL platform for questionnaire rendering.

### 10.1.2 Turkey's HSD Dataset

In order to apply the Turkey's test we need to modify the way which the dataset is organised. In the following listing we present the columns that the dataset is organised.

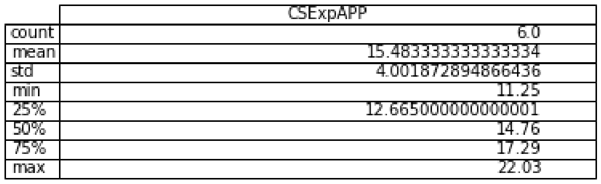
- **App:** This column includes the time records for the users that tested the original app.u-can-act platform.

- **DSL:** This column includes the time records for the users that tested the DSL platform.
- **CSBG:** This column includes the computer science background for each user. The different values in this column, depending on the users are: "high" , "low".

## 10.2 Statistics

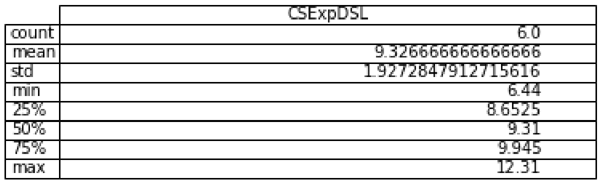
In the following figures 15, 16 present some basic statistics regarding the time records for the users based on their computer science experience level.

Figure 15: High CS Background-Original platform



| | CSExpAPP |
|---|---|
| count | 6.0 |
| mean | 15.483333333333334 |
| std | 4.001872894866436 |
| min | 11.25 |
| 25% | 12.665000000000001 |
| 50% | 14.76 |
| 75% | 17.29 |
| max | 22.03 |

Figure 16: High CS Background-DSL platform

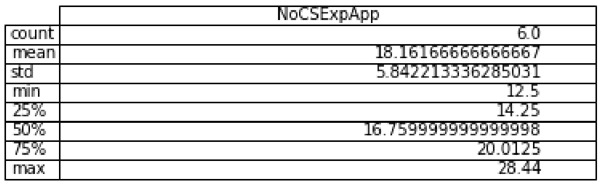


| | CSExpDSL |
|---|---|
| count | 6.0 |
| mean | 9.326666666666666 |
| std | 1.9272847912715616 |
| min | 6.44 |
| 25% | 8.6525 |
| 50% | 9.31 |
| 75% | 9.945 |
| max | 12.31 |

The above figures are pertained to the testers with a high computer science background, comparing the two different platforms. As we can observe from the results there is a significant difference of 39.79% between the means for both of the platforms. Furthermore the lowest value, using the original platform is 11 minutes and 25 seconds and the highest is 22 minutes and 3 seconds. On the other hand the counterpart statistics for the DSL platform are: 6 minutes and 44 seconds, 12minutes and 31 seconds. In terms of the minimum values we observe a decrease of 42.76% and 44.12% among the highest ones.

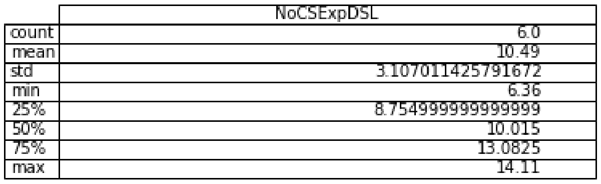
So far, we examined the differences between the two platforms regarding people with high computer science background. Now, we will present the differences among the platforms between users with low computer science background. In the following figures 17, 18 we present these differences:

Figure 17: Low CS Background-Original platform



| | NoCSExpApp |
|---|---|
| count | 6.0 |
| mean | 18.16166666666667 |
| std | 5.842213336285031 |
| min | 12.5 |
| 25% | 14.25 |
| 50% | 16.759999999999998 |
| 75% | 20.0125 |
| max | 28.44 |

Figure 18: Low CS Background-DSL platform



| | NoCSExpDSL |
|---|---|
| count | 6.0 |
| mean | 10.49 |
| std | 3.107011425791672 |
| min | 6.36 |
| 25% | 8.754999999999999 |
| 50% | 10.015 |
| 75% | 13.0825 |
| max | 14.11 |

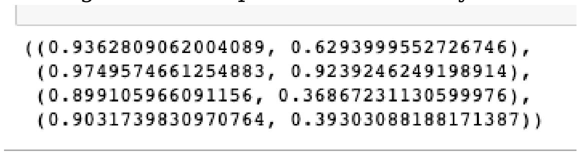
As we observe from the above figures there is a significant difference between the means for both of the platforms, regarding users with the same level in computer science(low). The mean value in the original platform is 18min and 16secs, while the mean for the same users, tested the DSL platform is 10min and 49secs. This results a decrease of 42.24%. Big differences between the platforms are also observed by the min and the max values. More specifically the difference between the min values is 47.22%, while the difference between the max values is 50.39%.

By comparing the differences between the users with "high" and "low" levels of computer science backgrounds we can assume that people with a higher level in computer science manage to achieve better scores by testing both of the platforms. Although it is also worth mentioning that for both of the users the difference between the two platforms is very big, meaning that the DSL platform managed to simplify the procedure of questionnaire rendering compared to the original platform which uses the JSON format.

## 10.3 ANOVA results

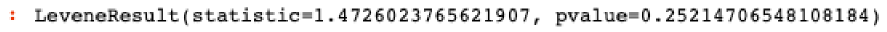
In this section we will present the results from the ANOVA test. As we described in the previous chapters in order to proceed with ANOVA test first we have to check for the assumptions regarding the normality and the variance. In the following figures 19, 20 we can see that all $pvalues$ are greater than the threshold $a = 0.05$, therefore we fail to reject the null hypothesis. That means that the samples come from the population follow the normal distribution have the same variance.

Figure 19: Shapiro-Wilk normality test



```
((0.9362809062004089, 0.6293999552726746),
 (0.9749574661254883, 0.9239246249198914),
 (0.899105966091156, 0.36867231130599976),
 (0.9031739830970764, 0.39303088188171387))
```

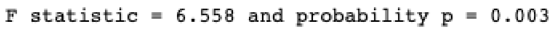
Figure 20: Levene variance test



```
: LeveneResult(statistic=1.4726023765621907, pvalue=0.25214706548108184)
```

The ANOVA results are presenting in the following figure. We performed an ANOVA test among the four different categories in our data which we collect through our experiment with users among the two different survey platforms.

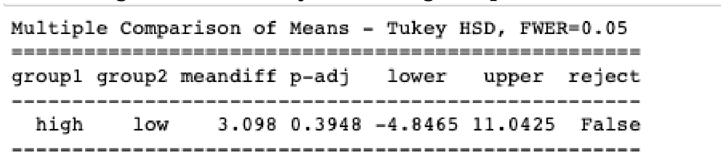
Figure 21: ANOVA results



```
F statistic = 6.558 and probability p = 0.003
```

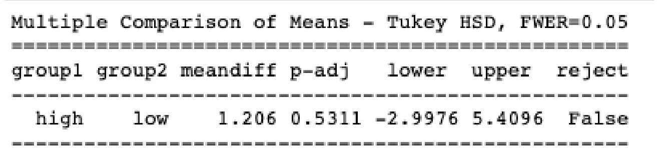
As $p < a(0.05)$ we state that we have a main interaction effect. As the probability is lower than 0.05 we reject the null hypothesis. This means that statistically significant differences among groups have identified. However these results does not identify the pairs which cause these significant differences. For our further examination we will use the Turkey HSD (honestly significant difference) test which is a single multi-comparison test. For this purpose we categorized our user based on their computer science level("high", "low") and we will compare their times for both of the platforms. The results can be found on the following images 22, 23.

Figure 22: Turkey HSD Original platform



```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower   upper  reject
----------------------------------------------------
  high    low    3.098 0.3948 -4.8465 11.0425  False
----------------------------------------------------
```

Figure 23: Turkey HSD DSL platform



```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
  high    low    1.206 0.5311 -2.9976 5.4096  False
---------------------------------------------------
```

Based on the above results we can assume that the computer science background of the users does not have a significant impact on the times that the users did using the platforms. This is the reason why we do not "reject" the null hypothesis that there are no significant differences based on the computer science background for each user. Although we observe that the "meandiff" value between the users with different computer science background is much lower on the DSL platform. This shows that it is easier for non-experienced users to cover the gap from users which are much more experienced.

### 10.4 Results from RUG psychology department users

In order to examine the behavior of users experienced in the app.u-can-act.nl platform we asked them to take part in the experiment. As we can see from table 10.4, despite the fact that the users were coming in touch with the DSL platform for the first time, the difference in terms of time is very significant as the DSL was 57% faster . This difference can be explained by focusing in three main factors: (i) users can faster trace and correct their syntax errors and (ii) it is very easy easy for the users to understand and reproduce the syntax and (iii) they could copy/paste questions of the same type and directly modify them.

| DSL | u-can-act |
|---|---|
| 11min 10sec | 23min 56sec |
| 4min 35sec | 11min 19sec |

## 11 Discussion

Our main goal was to develop a platform which will be based on a DSL in order to provide non-programmers an easy and efficient way for survey building. The results show that our approach is much faster not only for non-programmers but also for people with experience in computer science. Apart from that, we found that it is easier for the users to work with a DSL, as they can easily copy/paste questions which are of the same question type or modify them. Furthermore it is much easier for users to detect possible

errors which may have been made regarding the syntax.
The biggest problem we detected was that the users were not familiar with the parser precision as the syntax has a clear and strict order which can not be violated. This can lead to small errors mainly regarding the order of the syntax rules which are defined. There was a significant learning effect regarding the DSL platform as users managed to understand and use it much faster that the other platform. Also they liked the fact that through simple notations you can directly have a complete questionnaire. Although what would be also interesting is to have a comparison between the DSL and other GUI platforms. The results of this comparison would help us to have the complete picture of our platform compared to other platforms.

## 12 Conclusion

In this thesis, implementation of an application for building questionnaires by writing them in a DSL has been discussed. An easy text syntax was developed in order to make our platform accessible and usable by people with different experience on computer science. The architecture supports that our platform can interact with various questionnaire platforms and extended in the future.

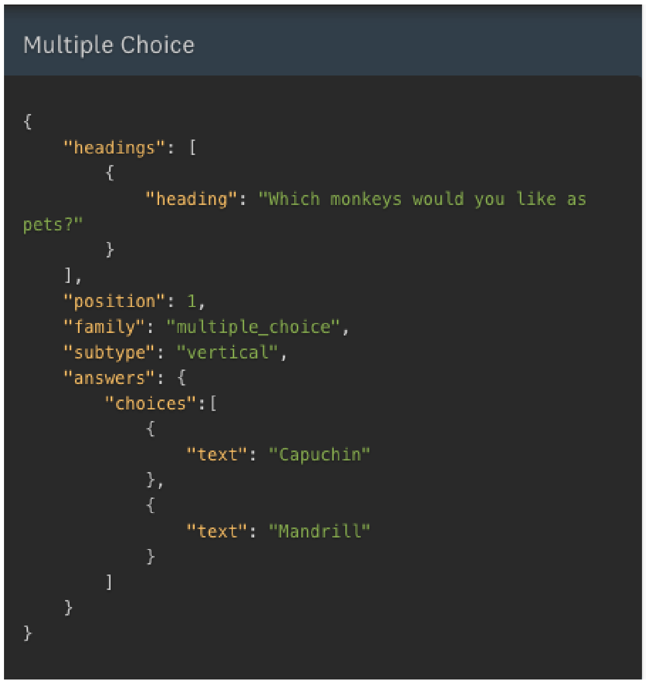
The goals of the project were met as we proved that a DSL can significantly reduced the needed time for building a questionnaire compared to other online survey editors and secondly that there are no significant differences in understanding and usage between people with different backgrounds on computer science. This approach can have a significant impact on computer science domain as it can extend the field of Domain Specific Languages even more in domains which are still unexplored.
Of course, further development that adds more features will be welcome and will further improve the final result.

## 13 Future Work

The architecture of our platform is designed to fit to different back-ends. Currently no actual implementation of that exists but it would be a nice feature for future work to be implemented.
One important extension would be to make our platform compatible with other online survey editors. For example a user using our DSL would be capable of rendering questionnaires in other online survey platforms such as Survey Monkey, Google Forms, etc. Of course this depends on other platforms too, and it has to do with how much access they give you to their API.

Another possible feature which could be added would be to inform users about possible syntax errors. For example we could highlight the line where the errors has been made in order to be easier for the user to correct it. Finally, we could support a feature which could read a questionnaire from a word file and convert it to our DSL format. This could be very useful for the users in order to directly render already predefined questionnaires without the need of converting it manually to our DSL syntax format.

## A SurveyMonkey JSON examples



```
{
    "headings": [
        {
            "heading": "Which monkeys would you like as
pets?"
        }
    ],
    "position": 1,
    "family": "multiple_choice",
    "subtype": "vertical",
    "answers": {
        "choices":[
            {
                "text": "Capuchin"
            },
            {
                "text": "Mandrill"
            }
        ]
    }
}
```

Figure 24: Survey Monkey Multiple Choice question

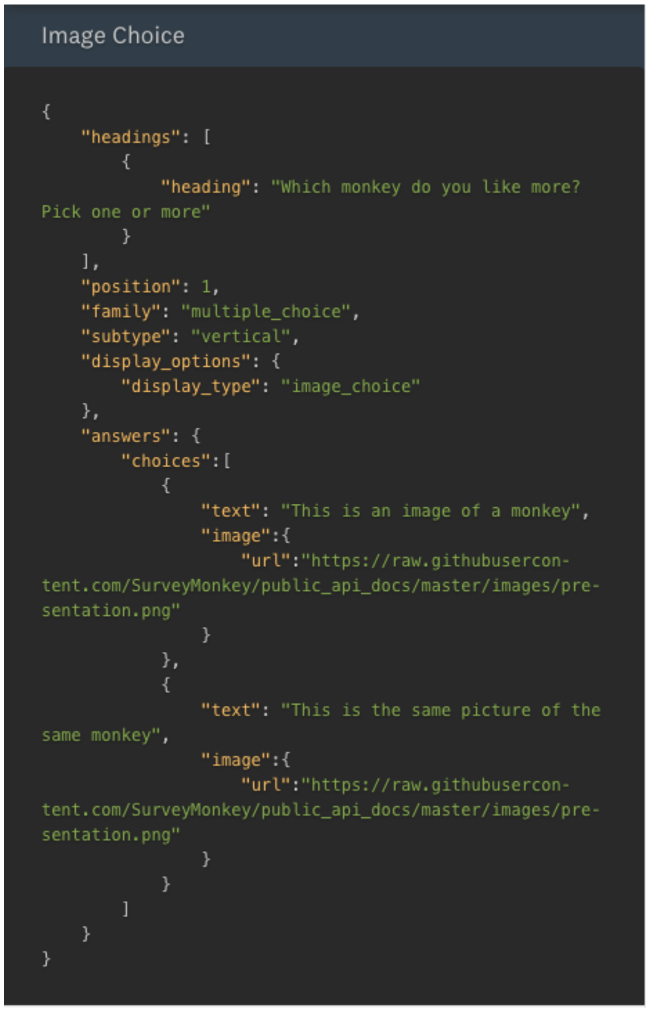


Image Choice

```
{
    "headings": [
        {
            "heading": "Which monkey do you like more?
Pick one or more"
        }
    ],
    "position": 1,
    "family": "multiple_choice",
    "subtype": "vertical",
    "display_options": {
        "display_type": "image_choice"
    },
    "answers": {
        "choices":[
            {
                "text": "This is an image of a monkey",
                "image":{
                    "url":"https://raw.githubusercon-
tent.com/SurveyMonkey/public_api_docs/master/images/pre-
sentation.png"
                }
            },
            {
                "text": "This is the same picture of the
same monkey",
                "image":{
                    "url":"https://raw.githubusercon-
tent.com/SurveyMonkey/public_api_docs/master/images/pre-
sentation.png"
                }
            }
        ]
    }
}
```

Figure 25: Survey Monkey Image Choice question

## B Peg.Js Installation and Compilation commands

```
$ npm install -g pegjs
```

Listing 14: Installation of peg.js with node package manager

```
$ pegjs parser.pegjs
```

Listing 15: Compilation of pegjs grammar into javascript code

## C Questionnaire to render

```
Id :v1
title: 1. How much do you suffer from psychological complaints?
type : radio
option: Not or hardly
option: Quite
option: Very much

Id :v2
title: 2. How much do you suffer from physical complaints
?
type : radio
option: Not or hardly
option: Quite
option: Very much

Id: v3
title: 3. How much do psychological complaints interfere
with your (looking for) work?
type: radio
option: Not or hardly
option: Quite
option: Very much

Id: v4
title: Year of first psychotic episode:
type: number
min:1900
max:3000
maxlength 4

Id: v5
title: Year of first mental health care - contact
type: number
min:1900
max:3000
maxlength:4
```

Listing 16: Python example

## D Questionnaire answers-Original Platform

```
[{"type":"raw","content":"Very briefly! Help us understand how you feel"},{"
    type":"raw","content":"Can you tell us how you felt about the past week,
    through today, have felt? We ask you to give report marks in a number of
areas."},{"type":"raw","content":"Read each question carefully and circle the
    number that best describes your current situation."},{"id":"v1","type":"
    radio","title":"1. How much do you suffer from psychological complaints?","
    options":[{"title":"Not or hardly"},{"title":"Quite"},{"title":"Very much"
    }]},{"id":"v2","type":"radio","title":"2. How much do you suffer from
    physical complaints?","options":[{"title":"Not or hardly"},{"title":"Quite"
    },{"title":"Very  m u c h }]},{"id":"v3","type":"radio","title":"3. How much
    do
psychological complaints interfere with your (looking for) work? ","options":[{
    "title":"Not or hardly"},{"title":"Quite"},{"title":"Very  m u c h }]},
{"id":"v4","type":"number","title":"Year of first psychotic
episode?   , maxlength  ":4,"placeholder":"1234","min":1900,"max":3000,"
    required":true}]]
```

Listing 17: u can act platform questionnaire

## E Questionnaire answers-DSL Platform

```
Very briefly. Help us understand how you feel

Can you tell us how you felt about the past week,
through today, have felt? We ask you a
to give report marks in a number of areas.

Read each question carefully and circle the number that
best describes your current situation.

&v1
#1. How much do you suffer from psychological complaints?
~{radio}
[]Not or hardly
[]Quite
[]Very much

&v2
#2. How much do you suffer from physical complaints?
~{radio}
[]Not or hardly
[]Quite
[]Very much

&v3
#3. How much do psychological complaints interfere
with your (looking for) work?
~{radio}
[]Not or hardly
[]Quite
[]Very much

&v4
#Year of first psychotic episode:
~{number}
@min:1900
@max:3000
ml:4

&v5
#Year of first mental health care - contact
~{number}
@min:1900
@max:3000
@ml:4
```

Listing 18: DSL questionnaire

## References

[1] *Abstract Syntax Trees*. URL: https://ruslanspivak.com/lsbasi-part7/.

[2] *ANOVA*. URL: https://en.wikipedia.org/wiki/Analysis_of_variance#:~:text=Analysis.

[3] Paul Klint Arie van Deursen and Joost Visser. "Domain-Specific Languages: An Annotated Bibliography". In: ().

[4] Martin Fowler. *Domain Specific Languages*.

[5] Martin Fowler. "Language Workbenches: The Killer-App for Domain Specific Languages". In: ().

[6] *Google Forms*. URL: https://developers.google.com/apps-script/reference/forms.

[7] *Internal vs External DSLs*. URL: https://subscription.packtpub.com/book/application_development/9781782166504/1/ch01lvl1sec09/internal-versus-external-dsls#:~:text=The%20use%20of%20a%20DSL,be%20called%20an%20External%20DSL..

[8] *Json Documentation*. URL: json.org.

[9] *Levene test*. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35a.htm.

[10] *Lexing and parsing overview*. URL: http://savage.net.au/Ron/html/graphviz2.marpa/Lexing.and.Parsing.Overview.html.

[11] Klaus Birken Markus Voelter Bernd Kolb and Federico Tomassetti. "Using Language Workbenches and Domain-Specific Languages for Safety-Critical Software Development". In: ().

[12] *Microservices*. URL: https://microservices.io/.

[13] *Peg.js Documentation*. URL: https://pegjs.org/documentation.

[14] *Shapiro-Wilk test*. URL: https://www.itl.nist.gov/div898/handbook/prc/section2/prc213.htm.

[15] *Survey platforms market share*. URL: https://www.pcmag.com/picks/the-best-online-survey-tools.

[16] *The u-can-act Platform: A Tool to Study Intra-individual Processes of Early School Leaving and Its Prevention Using Multiple Informants*. URL: https://www.frontiersin.org/articles/10.3389/fpsyg.2019.01808/full.

[17] *Turkey's HSD*. URL: http://www.blackwellpublishing.com/specialarticles/jcn_8_304.pdf.

[18] M. Veldthuis. "Quby, a domain-specific language for non-programmers". In: ().

[19] Markus Voelter. “Language and IDE Modularization, Extension and Composition with MPS”. In: pp. 383–430.