FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Conversational Interface for Webpage Code Generation

Luís Miguel Santos Monteiro Saraiva

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: André Restivo

Co-Supervisor: Hugo Sereno Ferreira

July 30, 2020

A Conversational Interface for Webpage Code Generation

Luís Miguel Santos Monteiro Saraiva

Mestrado Integrado em Engenharia Informática e Computação

July 30, 2020

Abstract

Since the creation of the World Wide Web in 1993, several languages were developed that permit to specify the elements in the webpages and change their appearance. Furthermore, tech companies and developers felt that it was required to create prototypes of the webpages before implementing them. Before the creation of these prototypes, hand-drawn mockups are made to outline the elements that will appear on the prototype as well as their position. After creating the prototype, the designers and developers present the result to the client. In this step, the client can verify if the prototype is according to his specification. In case that something is missing on the prototype, the mockups and the prototype must be created again. The repetition of these steps can be an arduous, time-consuming and a skill-intensive process. Easing this process would permit developers to create more accurate prototypes.

Natural Language Programming allows developers to program using natural language. These systems usually use a Speech Recognition (SR) and a Natural Language Processing (NLP) module. The SR module transforms the user's voice into text. The NLP component tries to understand what was the user's intention using the text that was obtained in the SR module. Due to the current development of Artificial Intelligence, both modules were largely improved. However, until now, these tools do not permit the generation of webpages. This thesis aimed at creating a tool to generate webpages using natural language without designing its sketch.

Given the problems detailed above, we developed a conversational interface as an alternative way to create web page prototypes, without needing HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) knowledge, and without creating a visual sketch of the website beforehand. First, a domain-specific language was specified. This language allows creating elements on a webpage, changing their appearance and it contains other functionalities that developers are familiarised when creating software. Afterwards, the conversational interface was developed using this domain-specific language. The conversational interface has a NLP module that permits understanding the action that was requested to be executed by the user. After assessing what the user's intention was, the conversational interface executes the functionality that was requested.

Afterwards, the domain-specific language was validated by creating a questionnaire where asking the participants to write commands that could replicate a webpage. The most used elements in the answers were present in the domain-specific language as well as the most used properties to change the appearance of the elements. Then, an experiment was conducted to evaluate the developed conversational interface. The participants of this experiment felt that the domain-specific language was not hard to understand and the ones that usually do not develop front end answered that they would use this tool to develop prototypes.

We believe this work can influence how a prototype is developed by easing the arduous and continuous task of creating mockups and transforming them into prototypes.

Keywords: Computer-aided Design, Conversational Interface, CSS, HTML, Natural Language Processing

Resumo

Desde a criação da World Wide Web em 1993, foram desenvolvidas várias linguagens para especificar elementos nas páginas web e mudar a sua aparência. Para além disso, empresas tecnológicas e programadores sentiram que era necessário criar protótipos das páginas web antes de as implementar. Antes da criação do protótipo, são feitos desenhos à mão para definir os elementos que irão aparecer na página web e a sua posição. Após a criação do protótipo, os designers e developers apresentam o resultado ao cliente para verificar se o protótipo está de acordo com a especificação. Se o protótipo não estiver de acordo com a especificação, os mockups e o protótipo têm de ser refeitos. A repetição destes passos pode ser um processo árduo, demorado e exigente pelo que é importante tornar este processo mais eficiente.

A Programação por Linguagem Natural incluem um módulo de *Speech Recognition* (SR) e um módulo de *Natural Language Processing* (NLP). O módulo de SR transforma a voz do utilizador em texto. O módulo de NLP pretende compreender qual foi a intenção do utilizador usando o texto que foi obtido no módulo de SR. Devido ao desenvolvimento da Inteligência Artificial, foram realizados progressos em ambos os módulos. Apesar disso, atualmente, estas ferramentas não permitem a geração de páginas web. Neste trabalho foi criada uma ferramenta para gerar páginas web usando linguagem natural sem ser necessário fazer esboços dessa página.

Dados os problemas previamente detalhados, foi desenvolvida uma interface conversacional como uma forma alternativa para desenvolver páginas web sem ser necessário ter conhecimento de HTML e CSS e sem ser preciso fazer um esboço da página web. Em primeiro lugar, foi detalhada uma linguagem específica de domínio. Esta linguagem permite a criação de elementos numa página web, mudar a aparência dos elementos e contêm outras funcionalidades com que os developers estão familiarizados. Depois, foi desenvolvida a interface conversacional utilizando esta linguagem específica de domínio. A interface conversacional tem um módulo de NLP que permite perceber qual foi a ação cuja execução foi solicitada pelo utilizador. Depois de compreender essa intenção, a interface conversacional executa a funcionalidade solicitada.

De seguida, a linguagem específica de domínio foi avaliada através de um questionário em que foi perguntado aos participantes para escreverem comandos que poderiam replicar uma página web. Os elementos mais utilizados nas respostas estavam presentes na linguagem específica de domínio assim como as propriedades que permitem mudar a aparência dos elementos. Depois, foi realizada uma experiência para avaliar a interface conversacional desenvolvida. Os participantes desta experiência sentiram que a linguagem específica de domínio não era difícil de perceber e os que normalmente não desenvolvem *front end* responderam que utilizariam esta ferramenta para desenvolver um protótipo.

Ao finalizar este trabalho pensamos que ele pode influenciar a forma como um protótipo é desenvolvido facilitando a árdua e contínua tarefa de criar esboços e transformá-los em protótipos.

Keywords: CSS, Desenho assistido por computador, HTML, Interface Conversacional, Processamento de linguagem natural

Acknowledgements

Firstly, I want to express my gratitude to my Supervisor, André Restivo, and Co-Supervisor, Hugo Sereno Ferreira, for their constructive feedback and time which permitted to improve the tool that was developed and is explained in this document.

Secondly, I want to thank my friends, especially to Francisca and Mariana, for always motivating me.

Finally, I wish to thank my family for always being in my side and caring for me. A special thanks to my father for the revision of the thesis.

Luís Miguel Santos Monteiro Saraiva

"Give me six hours to chop down a tree and
I will spend the first four sharpening the axe."

Abraham Lincoln

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
CSS	Cascading Style Sheets
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
LDB	Local Discriminant Bases
LPC	Linear Predictive Coding
MFCC	Mel-Frequency Cepstral Coefficients
NLP	Natural Language Processing
RNN	Recursive Neural Network
SQL	Structured Query Language
SR	Speech Recognition
TS	Text Synthesis
UI	User Interface
URL	Uniform Resource Locator
WWW	World Wide Web

Chapter 1

Introduction

This chapter provides an introduction to this MSc Thesis by addressing its scope, motivations, and goals. Section 1.1 outlines the problem associated with the creation of user interface prototypes. Section 1.2 delineates the main drivers of this work. Section 1.3 introduces the issue tackled by this thesis. Section 1.4 describes what this project aimed at achieving. Section 1.5 details the hypothesis of this work. Lastly, Section 1.6 explains the organisation of the rest of the document.

1.1 Context

At the end of the 90 century, Tim Berners-Lee started working on the World Wide Web (WWW), on the HyperText Transfer Protocol (HTTP), on the HyperText Markup Language (HTML), on the first web browser and on the first web server. Nowadays, the WWW is used by the majority of humans and can be accessed using a computer, laptop or smartphone.

Several languages that allow developing webpages were created since the creation of the WWW. Some of these languages are HTML, Cascading Style Sheets (CSS), JavaScript, PHP and Structured Query Language. These languages can be divided into two groups. The first group is composed by the languages that permit the creation of webpages (HTML) and change how the webpage looks (CSS) and how it can be interacted (JavaScript). These languages are normally used by front end developers. The second group contains the languages that receive and send requests from the webserver (PHP and SQL) which are used by back end developers.

Normally, designers draw sketches of the webpages and then transform it into a intermediate representation. Afterwards, front end developers create prototypes from these intermediate representations. These sketches permit the designers to express their views in an easy way to understand. The transformation of the intermediate representation to a web prototype gives the client a better picture of how the website he/she asked will function. With this better understanding, the client can verify if all the requirements he/she specified are considered and included in the design. This four-phase process can be very time-consuming because it only ends when the client agrees with the design that is presented to him.

After this procedure, the developers need to transform the mockups into a functional prototype. During this transformation, some design flaws can be detected by the developers, which means that the process mentioned previously must be iterated. This repetition indicates that even more time is taken to complete the whole development process.

1.2 Motivation

As mentioned previously, designers utilise hand-drawn sketches to express their ideas more conveniently. This procedure permits them to express their proposals in a quick way. Afterwards, these sketches are transformed in a intermediate representation that is sent to the developers. However, wrong assumptions can be made while transforming the sketches into the intermediate representation and in the conversion of the intermediate representation to the prototype. So, the designers must keep clarifying the mockups they draw which means they have a continuous communication with the developers. If the developers make incorrect assumptions, the design that the client agreed upon will probably not match the developed prototype. Therefore, the three-phase process explained before must be repeated to correct the issues that were detected meanwhile. This repetition signifies that it is taken more time and resources during the conception of a prototype. This ultimately means that along this process, there can be fidelity problems in reflecting the client specifications in the final webpages.

This continuous clarification of the mockups can be a tedious process. It also means that designers have to continue working on the project until the creation of the prototype is completed. Therefore, an alternative way to create web page prototypes, without needing HTML and CSS knowledge, and without creating a visual sketch of the website beforehand (either digital or analogue) was the primary motivation for this work. Another motivation was to create a tool that permits persons with accessibility constraints to generate prototypes.

1.3 Problem

As mentioned previously, the current web page development process is time and resource consuming. One of the most important tasks is the drawing of the sketches of the webpage. This phase can be a tedious process because it will eventually have to be repeated numerous times.

A possible way to create web prototypes without previously requiring the creation of a hand-drawn mockup is to use a conversational interface that enables the creation of web prototypes.

However, conversational interfaces face numerous challenges because machines do not efficiently perform tasks such as interpreting natural language into logical representation intuitively as humans can do. Another difficulty arises when the user does not provide all the required information to perform a certain action. Furthermore, the conversation between the chatbot and the user should not appear to be repetitive.

A conversational interface also requires the specification of a domain-specific language. This language permits the chatbot to determine what was the intent of the user and to give an adequate response. Some challenges arise when specifying a domain-specific language such as (a) the language should not contain ambiguities, and (b) the users should easily comprehend the language.

1.4 Goals

As described above, the objectives of this work were to develop a tool that enables the creation of a webpage without requiring the use of sketches. These objectives can be achieved by contributing to the research field of prototype generation using natural language.

To achieve this goal, it was necessary to develop a software that has the ability of (a) obtaining information from natural language, (b) developing all the required logic to execute the actions that were interpreted in the previous step, (c) generate HTML and CSS code and (d) provide continuous feedback of the actions that the user demanded to be executed. The program should be able to translate a sequence of instructions into the final code by employing the necessary steps.

1.5 Hypothesis

In this work, we wanted to investigate if a conversational interface developed with the purpose of generating webpages using a simple to understand domain-specific language is more pleasant to utilise than the current forms to develop a prototype. We also wanted to analyse if front end and back end developers would use such a tool.

1.6 Thesis Structure

Chapter 2 (p. 5) provides contact with the environment that supports this work. Chapter 3 (p. 13) describes the current state of the art associated with the development of user interfaces through natural language processing. Chapter 4 (p. 27) describes the issues and limitations of the existing implementations and introduces the proposed solution and validation. Chapter 5 (p. 33) details the tool that was developed. Chapter 6 (p. 49) describes the strategy that was used to validate the work developed and the results obtained. At last, Chapter 7 (p. 65) addresses the current state of the project, its goals and future work.

Chapter 2

Background

This chapter introduces concepts that are fundamental to a good understanding of this thesis. Section 2.1 explains the main techniques to experience a conversation with virtual software. Section 2.2 addresses some methodologies whose objective is to give the users continuous feedback. Section 2.3 presents the most relevant languages that developers use when developing a prototype. Finally, Section 2.4 summarizes this chapter.

2.1 Architecture of a Conversational User Interface

The typical conversational user interface requires three components. The first is a Speech Recognition (SR) component that allows the software to convert the audio of the user into a text string. The second is a Natural Language Processing (NLP) module that extracts the meaning of a text string, and it is also able to generate a text string containing a response for the audio of the user. The last piece is a Text Synthesis (TS) that transforms a text string into an artificial voice, doing precisely the opposite of the SR component.

2.1.1 Speech Recognition

According to Yu *et al.* [YD16], Speech Recognition can be defined as the ability of a computer to transform the audio of the user's voice into a text string.

As shown in Figure 2.1 (p. 6), the underlying architecture of an Automatic Speech Recognition (ASR) system is composed of four modules: Signal Processing and Feature Extraction, Acoustic Model, Language Model and Hypothesis Search. These four items will be briefly addressed in the next paragraphs.

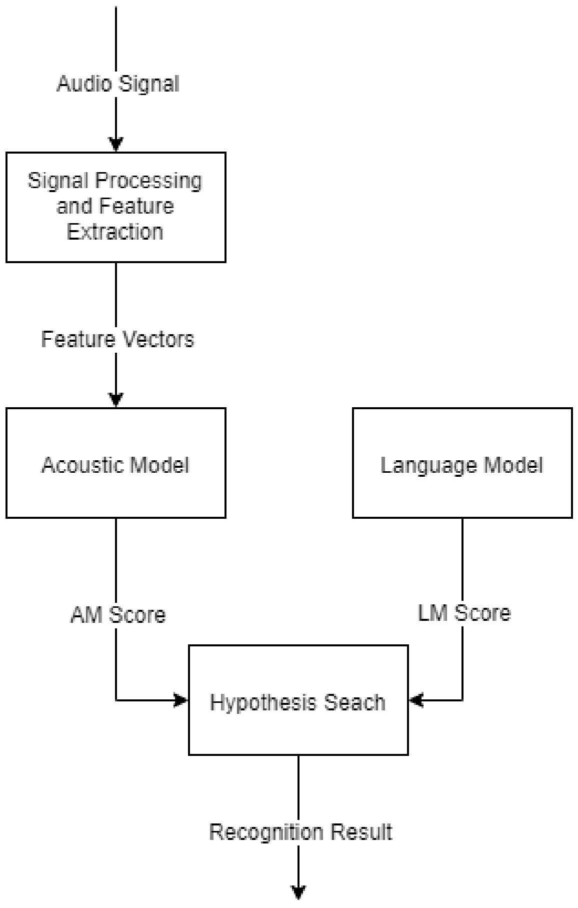


Figure 2.1: Architecture of an Automatic Speech Recognition System

The **Signal Processing and Feature Extraction** receives the audio signal as input and removes the noises and channel distortions, enhancing the quality of the audio. Afterwards, it converts the sound from the time-domain to the frequency-domain and obtains feature vectors that are appropriate for the acoustic model. This component is a critical module in SR and has a notable impact on the recognition performance.

The **Acoustic Model** incorporates information about acoustics and phonetics to create a score for the variable-length feature sequence from the feature vectors that were generated by the previous model.

The **Language Model** learns how words associate each other from a dataset and produces an estimate of the probability of a hypothesised word sequence. This estimation can be more accurate if the programmer knows the domain or task previously. Therefore, it is a probabilistic model that is capable of predicting the next word on the sequence given the previous terms.

One way to improve the performance of ASR is to develop better language models. One of the current methods to develop these models is to use Deep Neural Networks (DNNs), and more specifically, Recurrent Neural Networks (RNNs). RNNs perform the same function for all inputs, and the output of the current input will influence the next computation. So, to get a decision, the network takes into consideration the current input and the output from the previous input. This property gives the RNN Language Models the potential to model long span dependencies. Therefore, RNNs are becoming increasingly popular for language modelling purposes. However, Sundermeyer *et al.* indicates that *"recurrent networks are challenging to train and therefore are unlikely to show the full potential of recurrent models."*

Finally, the **Hypothesis Search** merges the scores of the Acoustic Model and the Language Model for the specified feature vector sequence and the hypothesised word sequence. The word sequence that has the highest score is the output of this system.

2.1.2 Natural Language Processing

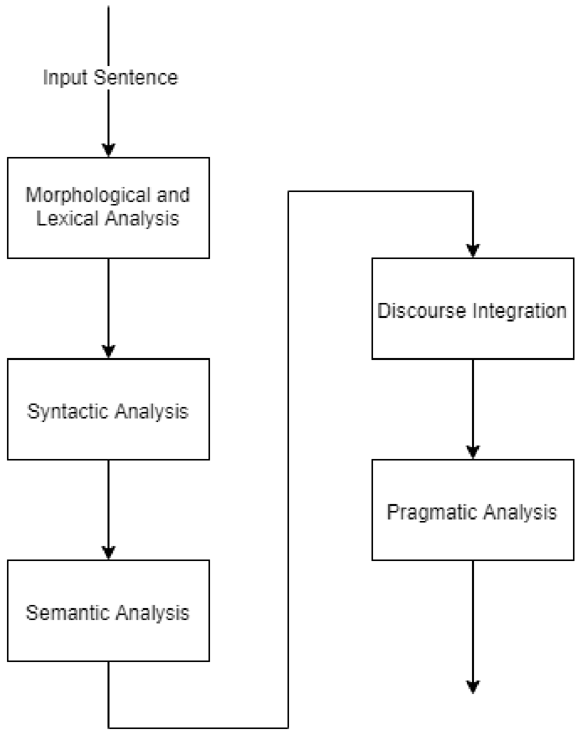


Figure 2.2: Natural Language Processing Components

According to Chowdhury [Cho03], developers face three crucial problems while creating programs that understand natural language: the first one is associated to thought processes, the second to the representation and meaning of the linguistic input, and the third to world knowledge. Therefore, a pipeline for this type of programs can be defined. The pipeline starts by understanding the morphological structure and nature of all the words. Afterwards, it can find the word order, grammar and meaning of the entire sentence and then it will understand the overall environment of the sentence. Certain words or sentences may have different meanings depending on the context and can be related to other words or sentences in the given context.

Liddy [Lid98] and Feldman [Fel99] state that it is relevant to discern seven levels that people utilise to understand natural languages. These levels are Phonetic Analysis, Morphological Analysis, Lexical Analysis, Syntactic Analysis, Semantic Analysis, Discourse Integration and Pragmatic Analysis. A program that is capable of understanding natural language may use all or some of these levels. The next paragraphs provide some details on these seven levels.

Phonetic Analysis handles pronunciation. It is utilised in speech recognition systems that accept spoken queries or provide spoken documents. This process is crucial in voice recognition systems. However, it is not essential for written text in information retrieval systems.

Morphological Analysis deals with the smallest parts of words that carry meaning, suffixes and prefixes. Some examples are child (the stem for childhood, children and childish), the prefix *un-* and the suffix *-ation*.

Lexical Analysis is used for part-of-speech tagging or the utilisation of lexicons. It is responsible for the lexical meaning of words and parts of speech analyses. Therefore, it divides a text into paragraphs, sentences and words. It also separates non word tokens such as punctuation from words.

Lexical Analysis reads character streams from the source code, checks for legal tokens and passes the data to the Syntax Analyser. This analysis is an essential part of NLP because it is in this phase in which natural language is segmented into words. Therefore, the scanning and identification of valid strings should be made by considering the language that is being used.

Usually, regular expressions are used in this step because they can express finite languages by defining a pattern for finite strings of symbols. This method is viable because each pattern corresponds to a set of strings.

Syntactic Analysis handles the grammar and structure of sentences. Thus, the focus of this level is to find the correct order of words. The syntactic analysis of a text requires checking whether the words in the text conform to the grammatical structure of the sentence.

A Syntax Analyser receives the part-of-speech tagging of the Lexical Analyser as input. It analyses this input against the rules that integrate the grammar of the language to assign phrase and clause brackets. This phase utilises Context-Free Grammars and outputs a parse tree.

Semantic Analysis deals with the meaning of words and sentences. Semantics understands that several words can have different meanings that depend on the given context. Therefore, the Semantic Analysis focuses on the literal meaning of words and sentences by abstracting the real definition from the given context. Thus, a Semantic Analyser task aims at ensuring that the purpose of the sentences is clear and consistent with the way they are supposed to be used. It can expand the query by adding all synonymous equivalents of the query terms.

Discourse Integration handles with the structure of different kinds of text using document structures. All different types of text have a defined structure. A newspaper article describes the facts at the beginning and makes predictions about the effect of the events in the end. A technical document starts with an abstract, detailing the contents of the paper. Discourse Integration utilises this expected structure to recognise the specific role of a segment of information in a document.

Pragmatic Analysis addresses the knowledge that comes from the outside of the content of the document. It tries to understand the user and his needs in the context and their goal. This can be achieved by gathering all the knowledge we know about the world. However, this takes too long, and it is not easy to add new information quickly.

All these levels are interconnected. This interconnection means that, for example, the meaning of a word in a sentence narrows the options of reasonable roles of the word within the sentence. Context and syntax should cooperate for a better understanding of the meaning of words. For instance, in the phrase "I watered the plant as soon as I got back to the house.", the word plant can be either a noun or a verb. With Syntax Analysis, we understand that it is the object of the verb "to water". However, we can narrow the meaning because it is unlikely that someone would water an industrial plant.

2.1.3 Text Synthesis

Text Synthesis (TS) aims at generating an artificial speech from a given text. Rashad *et al.* [REBIM10] explains that TS can produce this result in two steps. The first step is Text Analysis, where words are transformed into a phonetic representation. The second step is the generation of speech waveforms, where the phonetic representation from the previous step generates an acoustic output.

Text Analysis is typically composed of three components. The first module, Text Normalisation, is responsible for pre-processing the input text, breaking it into sentences and dividing these sentences into a sequence of tokens. The second component is Pronunciation which finds the correct pronunciation for all the words. The majority of TS systems utilise a first pronunciation lexicon and a name pronunciation lexicon. The combination of both lexicons enables the system to find the correct pronunciations for all the words. The third component is a Prosodic Analysis that makes the flow a sentence feel natural. Without this component, a speech would sound like reading a list of words.

2.2 Live Software Development

Aguiar *et al.* [ARC$^+$19] state that all development activities would benefit from increasing liveness of the entire software. They continue saying that if there were smaller feedback loops between the changes and the results, developers could accomplish better results in a quicker way.

Software development is known as the creation of software by the combination of several tasks. This process is acknowledged as the software development lifecycle, and it includes several functions as gathering and analysis of requirements, the design, implementation, verification and maintenance of software.

Typically, the performers of these tasks are humans, which are likely to make mistakes. Therefore, some methodologies propose quicker feedback to narrow the feedback loop between the human and the software. Usually, these methodologies focus on programming and are in a category known as Live Programming.

According to Burckhardt *et al.* [BFdH$^+$13], Live Programming requires an environment where the programmer can edit the code continuously with constant feedback. This methodology allows developers to detect and solve problems in a quicker way compared with the common “edit-compile-debug” style of programming. As claimed by Tanimoto [Tan13], in live programming, the program is running continuously, despite the occurrence of editing events. Tanimoto continues saying that it is not necessary to execute everything in live programming because running the program can distract the developer and it is not required to show intermediate states.

Spreadsheet software and visual languages usually provide live programming experience. In spreadsheet software, data and formulas can be edit, and the consequence of these changes is visualised immediately. Visual languages are programming languages that allow users to create programs by adding, removing or editing program elements graphically. Several visual languages are based on treating entities as boxes that can be related between themselves using arrows. However, these languages are not expressive enough for more complex general-purpose programs.

Software that enables a developer to program with live programming requires at least a state snapshot, a re-execution of the code that was changed and a re-displaying of the results. As Burckhardt *et al.* says, it is difficult to determine what information to save, what code to re-execute and which parts of the User Interface to maintain.

2.3 Web Development

In 1989, Tim Berners-Lee proposed to the European Organization for Nuclear Research an idea [BLC90] that later would be known as the World Wide Web (WWW). During the following years, Berners-Lee and his associates started working on the HyperText Transfer Protocol (HTTP), on the HyperText Markup Language (HTML), on the first web browser and on the first web server. Meanwhile, the first websites were also developed.

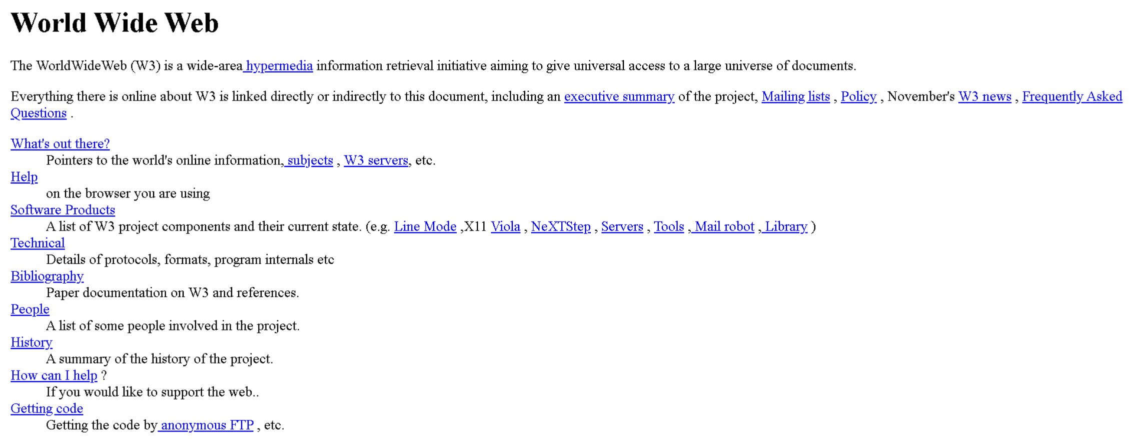
The first adopters of the WWW were mostly scientists, and at the beginning of 1993 there were just fifty web servers in the world. During that year, the WWW was made available without any royalties, which increased vastly the number of users. By the end of 1993, the number of web servers increased to five hundred. Nowadays, the majority of humans can access the WWW with their computers, laptops or smartphones.

Since the creation of the WWW, several programming languages were created aiming at developing websites. Some of these languages are HTML, Cascading Style Sheets (CSS), JavaScript, PHP and Structured Query Language (SQL). These languages are divided into two groups: the languages that create (HTML) and change how the user sees and interacts with a website (CSS and JavaScript) also known as front-end languages and the back-end languages that handle requests to and from the webserver (PHP and SQL). To develop a prototype, developers normally just utilise front-end languages.

HTML was the first language to be created to develop websites. The initial version, HTML 1.0, contained 18 elements[1] which allowed developers to program the first websites. However, these websites were not aesthetically pleasing, as it can be seen in Figure 2.3 (p. 11). Around 1994 several style sheet languages for website development were proposed. These languages enabled the developer to change the aspect of the website. Due to the increase of style sheet languages, in 1996 the World Wide Web Consortium (W3C) released the first W3C CSS Recommendation. Therefore,

[1] http://info.cern.ch/hypertext/WWW/MarkUp/Tags.html

the group of persons that were responsible for developing the standards of the WWW suggested that developers should use CSS to alter the way that the websites look like.



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
    Pointers to the world's online information, subjects , W3 servers, etc.
Help
    on the browser you are using
Software Products
    A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
    Details of protocols, formats, program internals etc
Bibliography
    Paper documentation on W3 and references.
People
    A list of some people involved in the project.
History
    A summary of the history of the project.
How can I help ?
    If you would like to support the web..
Getting code
    Getting the code by anonymous FTP , etc.

Figure 2.3: First Website

CSS was proposed in 1994 by Håkon Wium Lie, and it differentiates from the other style sheet languages because the HMTL elements are modified based on who specified the change, on the specification of the component and its position. In 1996, it was released CSS 1, but it had several limitations, and it was not supported by the majority of the browsers. Thus, the developers of CSS started developing a new version, CSS 2, that was released in 1998. This new version solved many of the problems of CSS 1 and added new functionalities that allowed changing the aspect of the website. By 1999, almost every web browsers achieved near-full implementation of CSS 1, and the same happened in 2003 for CSS 2. In 2011, it was published a revision of CSS 2 termed as CSS 2.1. This revision fixed some errors of CSS 2 and added some extensions that were already implemented in web browsers. Immediately after the publication of this revision, work on CSS 3 started. CSS 3 is divided into multiple documents called modules. Each one of these modules adds new features or improves some functionalities of CSS 2.

A website is a collection of webpages, and the user can navigate from one webpage to another using links. Websites are published on a server, and a user can access them using a web browser, also called a client. Although the essential function of a server is to serve contents, they are also capable of receiving contents from the clients. This functionality is used for submitting web forms, including uploading of files.

2.4 Summary

This chapter described significant concepts that support the understanding of this thesis. The knowledge of the components that are required for developing a Conversational User Interface was introduced. It was also presented the concept of Live Programming and described the main languages to build a prototype.

Chapter 3

State of the Art

This chapter describes the state of the art for the creation of a conversational interface that can reduce the prototype development cycle. Section 3.1 introduces some methodologies to help humans to design objects. Section 3.2 explains the leading implementations of conversational user interfaces. Section 3.3 describes methodologies that transform hand-drawn mockups into prototypes Section 3.4 details methodologies that enable developers to program using natural language. Finally, Section 3.5 summarises the main conclusions from this analysis.

3.1 Interaction in Computer-aided Design Softwares

Computer-aided design (CAD) software helps designers to create, modify, analyse and optimise a design. CAD software is utilised in several areas of engineering, such as electrical and mechanical. AutoCAD [1], SolidWorks [2] and Blender [3] are some of the most well known CAD software tools of this type.

This type of software typically allows the designer to add new components to the design and change them in an iterative way, and the results of each action are shown almost immediately. These functionalities enable the designer to verify if the design is according to the specifications. If it is not, the designer can easily change the drawing. These features are the main reasons for the popularity of this type of software because their use increases productivity, and it improves the quality of the design.

[1] autodesk.com/autocad
[2] www.solidworks.com
[3] https://www.blender.org

The users can easily create prisms, cylinders and spheres, for example, and add or remove forms continuously. This process is the equivalent of fitting together several parts of one object or cutting the object. An example of this process is shown in Figure 3.1.

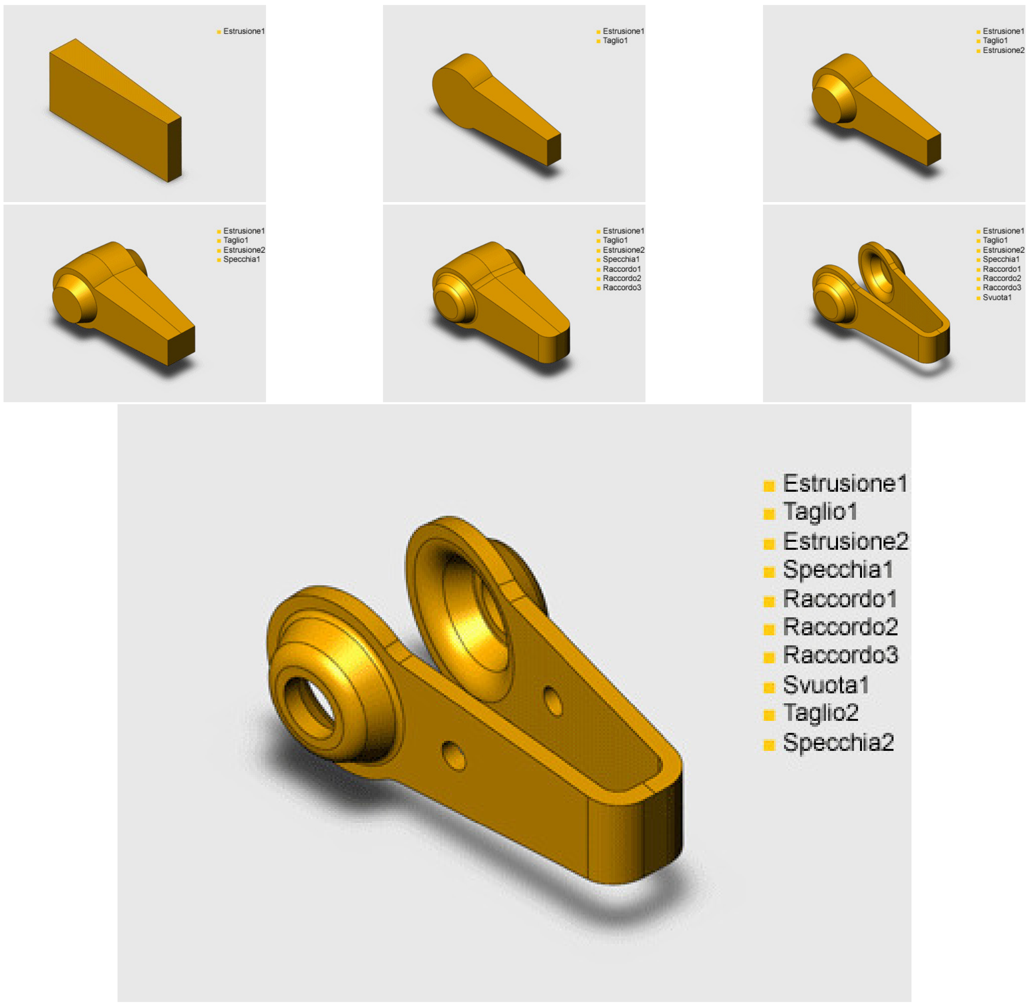


Figure 3.1: Process of creating a complex object in a CAD software

Some CAD software allows the designer to create objects using a command line. This command-line has a strict set of commands. This command-line shows possible instructions and options to the user. However, this feature can not be classified as a conversational interface because it does not provide any kind of feedback to the user. Furthermore, it is also not possible to write sentences. The users are only able to write the commands that were specified. Figure 3.2 (p. 15) shows the command line feature in AutoCAD.

3.2 Conversational User Interfaces

A conversational interface is a system that allows a more natural interaction than a graphical interface. Bieliauskas *et al.* [BS17] explains that this is due to the fact that conversational interfaces enable the same interactions that humans have between each other. These software tools attempt to

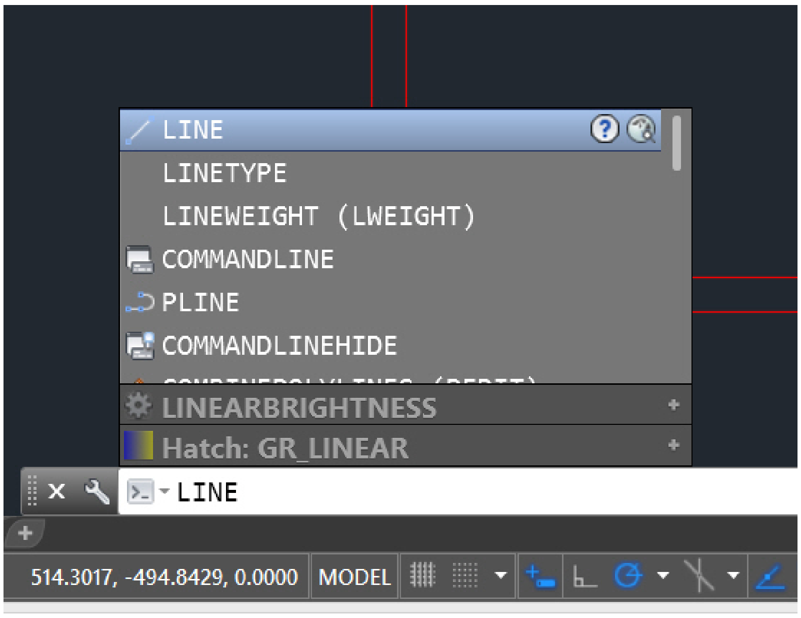


Figure 3.2: Command line feature in AutoCAD

understand natural language (speech or text) and to perform actions based on the input of the user. There are two kinds of conversational interfaces: chatbots and assistant systems. Sections 3.2.1 and 3.2.2 describe the current state of chatbots and assistant systems, respectively. Section 3.2.3 details some platforms where programmers can develop their own conversational interface.

3.2.1 Chatbots

A chatbot is a specification of an assistant system, in other words, a chatbot is designed for more specific tasks instead of general ones like an assistant system. Typically, a chatbot can perform more complex actions because it retains information about the interactions executed previously. This system is generally integrated inside other platforms. Some of the most acknowledged platforms that contain a chatbot are Slack and Skype.

Usually, chatbots accept text as input but, in some cases, voice can also be used. Furthermore, chatbots are able to ask questions to the users and answer questions made by the user. According to Abdul-Kader *et al.* [AKW15], a chatbot tries to reply with the smartest response to the input the user entered, and this process continues until the conversation ends. This answer can be either text or speech.

Abdul-Kader also indicates that the development of a chatbot can be divided into three parts: the recognition and conversion from speech to text, the processing of the text and the response and the execution of the corresponding action. The first component receives the digital signal captured from the microphone. Then it converts this signal into text. The text translated in this step is used in the second phase, where it is split into words. Afterwards, keywords can be extracted by removing unwanted words. At last, a chatbot can give an answer to the input that was entered by the user. The chatbot can also execute an action according to the input.

According to Klopfenstein *et al.* [KDMB17], some of the advantages of chatbots are the instant availability because they run immediately as a conversation starts. Chatbots also have a gentle learning curve. Typically, users write text which means that learning how to interact with the chatbot is not hard if the chatbot provides sufficient advice of its features.

Rahman [RAMI17] divides the chatbot platforms into three categories:

Nonprogramming chatbots: This type of chatbots do not require advanced programming skills. Their objective is to be not concerned about technical details. Chatfuel, ManyChat and Motion.ai are some chatbots of this category;

Conversational-Oriented Chatbots: This type of chatbot is able to establish a more dynamic conversation with the users. This kind of chatbots use a specification language to model their interactions;

Chatbots by tech giants: Google, Facebook, Microsoft, Amazon and IBM are developing their own chatbots. These companies also provide platforms that enable programmers to develop chatbots. These platforms are described in Section 3.2.3.

Rahman continues saying that there are two main challenges when developing a chatbot. One of these challenges is Natural Language Processing. The chatbot must realise that "What is the weather?" and "Could you check the weather?" are similar questions. The other challenge is that the chatbot should learn the correct response for each of the possible questions. This learning can be achieved by using artificial intelligence concepts. Rahman also concludes that developers should consider the stability, scalability and flexibility of the chatbot system while developing it.

Shawar *et al.* [SA07] concluded that there should not be a standard evaluation methodology to validate a chatbot system. Instead, the evaluation should check if the chatbot accomplishes the task it is meant to achieve.

3.2.2 Assistant Systems

Assistant systems are software solutions that allow users to speak or write and can comprehend the intentions of the user. Therefore, these systems must combine Speech Recognition (SR), Natural Language Processing (NLP) and Text Synthesis (TS) to be able to do such actions. These systems are designed to handle all types of questions. Consequently, they often integrate third-party functions. Some of the world's largest companies have developed their smart assistants such as Google's Assistante[4], Apple's Siri[5], Amazon's Alexa[6] and Microsoft's Cortana[7] [LDF20].

As mentioned previously, these software tools enable the user to enter text as input and, at the same time, they also allow voice as input. Although the user does not need to use any physical

[4]https://assistant.google.com/
[5]https://www.apple.com/ios/siri/
[6]https://developer.amazon.com/alexa
[7]https://www.microsoft.com/en-us/cortana

device, it is required a microphone to enable the interaction between the user and the system. This feature is one of the main characteristics of the assistant systems because it gives an extra flexibility to the users, and it provides a natural means of communication.

Each of the smart assistants developed by the world's largest tech companies mentioned previously has its specific features. Despite this, all of them are very similar: they respond to questions based on information that can be found on the internet, they also execute simple tasks and can utilise third-party services. Developers were able to implement in these bots the ability to correlate individual phrases to extract a standard line of thinking or, in other words, assistant systems have conversational awareness. Thus, developers were able to incorporate an essential component of human communication to an assistant system.

With the advancement of conversational technologies, having a conversation with a smart assistant is becoming very similar to having a conversation between two humans.

Currently, every mobile or computer user can have access to one good assistant system. The most popular smart assistants are:

Google's Assistant: The main focus of the Assistant is to give information to the user without asking for it. It also aims at providing personalised recommendations with the data gathered from the calendar events, the traffic to the user destination and other information. It also enables users to make hands-free cellphone calls by integrating third-party services. The Assistant is integrated into products that are related to Google such as Android smartphones, Pixel laptops and Google Home hardware.

Apple's Siri: Siri provides several hands-free features to its users. Apple integrates Siri with their smart home products, which gives its users the ability to interact with their home systems without touching it. Siri is embedded in all Apple mobile devices.

Amazon's Alexa: Alexa enables users to search the web or the systems that are connected with it. Alexa also connects with the Amazon marketplace. Therefore, users can purchase items from other Amazon services. It is possible to communicate with Alexa through Amazon's hardware or mobile apps.

Microsoft's Cortana: Cortana grants their users the ability to perform web searches, set reminders, recognise music, and it can make predictions. Cortana was created for Windows 10, Windows 10 mobile, Windows Phone 8.1, Invoke smart speaker, Microsoft Band and Xbox.

Smart assistants can be beneficial and easy to use. They also provide an alternative to standard input devices due to their ability to utilise voice as an input method. Therefore, individuals with accessibility constraints can utilise them. Thus, they can be handy when the objective is to develop software that is planned to be used by persons with accessibility constraints.

3.2.3 Conversational Interface Development Platforms

The development of conversational interfaces can be executed by everyone that has the required skills to do so, like all software applications. This development can be simplified if the developers include open-source Application Programming Interfaces (APIs) and libraries that handle the main difficulties of these systems, which are speech recognition and natural language processing.

Nowadays, several platforms help developers to create their chatbots. The most well known conversational interface development platforms are Dialogflow[8], Amazon Lex[9], IBM Watson Assistant[10], Wit.ai[11], Azure Bot Service[12]. All the bots that can be generated through these platforms allow the user to interact through voice and text-based conversational interface. Except for Wit.ai, all the development platforms mentioned before provide an easy way to use web interfaces that allow the user to efficiently create and deploy bots. They are also able to understand the intent, entities and meaning from the user's input. The bot's responses to the user input can change based on the examples given by the developer.

The bots that are created in the platforms mentioned previously can be easily integrated into applications such as Slack and Facebook Messenger. Therefore, developers can set aside the development of an interface between the chatbot and the users and focus entirely on the conversational component of the bot.

Focusing on Dialogflow, it is a product owned by Google that provides developers with a platform to create conversational interfaces.

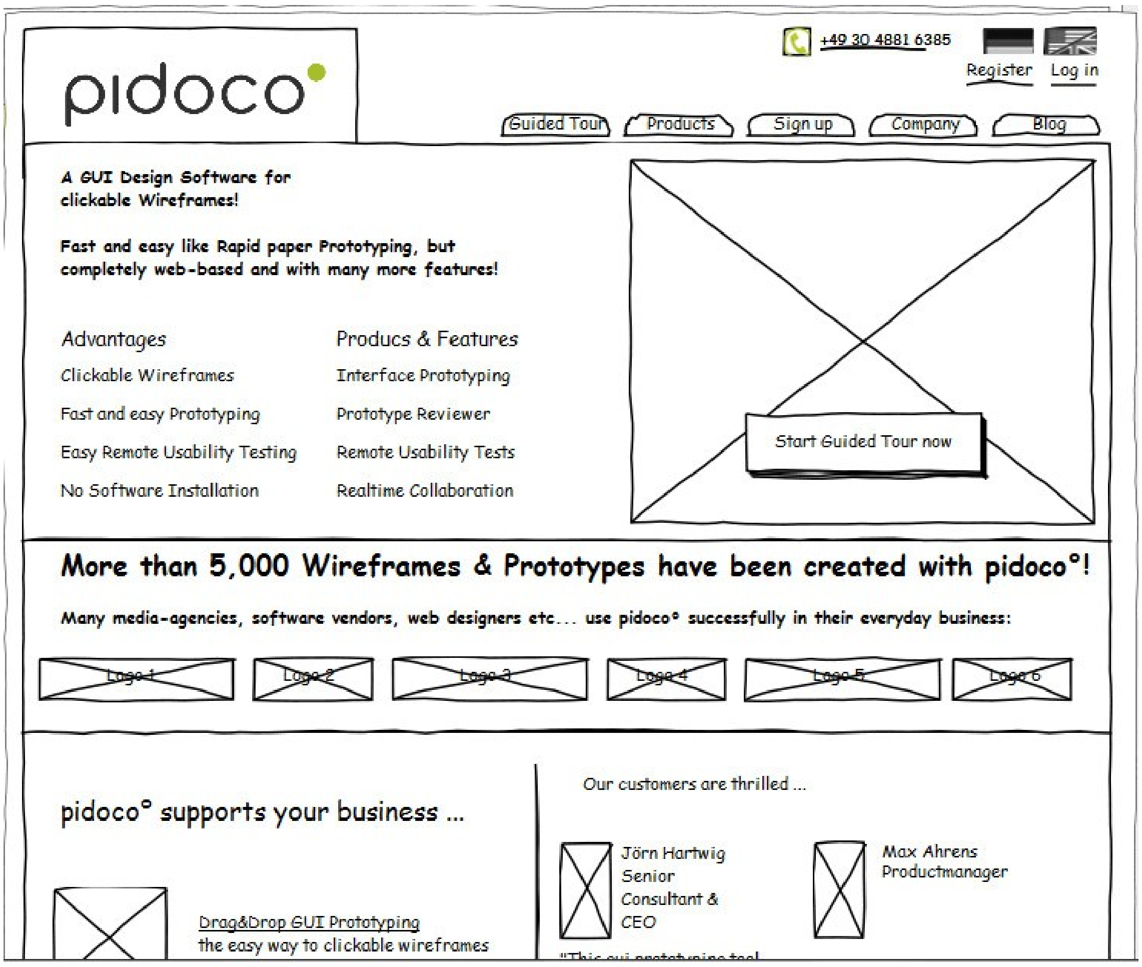
To develop a chatbot in this platform, the programmer must define the intents that stand for a user query. As an example, to specify the intent of a user saying "hi", the developer needs to create an intent that allows "hi" as user input. Afterwards, the developer must specify the output response for the intent that was created. Then, through machine learning, Dialogflow generates phrases that are equivalent to the intents that were created. Thus, the bot can answer to a broader range of user inputs.

The developer can also specify entities, which are structures that correspond to changeable segments of user sentences that can have distinct values. The value of the entity can be useful if the response of the bot depends on it, which means that the bot can accept even more user queries.

The web user interface of Dialogflow allows the developers to specify the intents and entities. The processing of the user queries occurs in Dialogflow's backend. Nevertheless, developers can send queries to the bots through external APIs and manage them differently.

The sophisticated natural language features that are implemented in Dialogflow and the different applications where the bot can be integrated, explain why it is one of the most used platforms for chatbot development.

[8] https://dialogflow.com/
[9] https://aws.amazon.com/lex/
[10] https://www.ibm.com/cloud/watson-assistant/
[11] https://wit.ai/
[12] https://azure.microsoft.com/en-us/services/bot-service/



A GUI Design Software for clickable Wireframes!

Fast and easy like Rapid paper Prototyping, but completely web-based and with many more features!

More than 5,000 Wireframes & Prototypes have been created with pidoco°!

Many media-agencies, software vendors, web designers etc... use pidoco° successfully in their everyday business:

pidoco° supports your business ...

the easy way to clickable wireframes

Figure 3.3: Hand-drawn mockup of a webpage

3.3 Hand-drawn mockup to Prototype

One of the objectives of this work was to create a tool that permits designers to create prototypes in a quicker way. This objective was addressed in several works such as Jain *et al.* [JAB$^+$19], Robinson [Rob19], Beltramelli [Bel18] and Ferreira *et al.* [dSF19]. These four works aim at reducing the duration of the development of a prototype by transforming a hand-drawn mockup into a prototype.

A hand-drawn mockup illustrates the content, features, and links that will be displayed on the page. It also aids designers to generate a prototype of the web page and helps programmers to understand the features and their working on the web page. Figure 3.3 shows a hand-drawn mockup of a webpage.

After creating the hand-drawn mockup, designers generate a prototype of the webpage. A prototype is a product that is created to test the different functions of the website before continuing developing the website. It also permits to identify mistakes in an easier way. Clients can visualise the website that is being created in a design more user-friendly. Figure 3.4 (p. 20) shows a prototype of a web application.

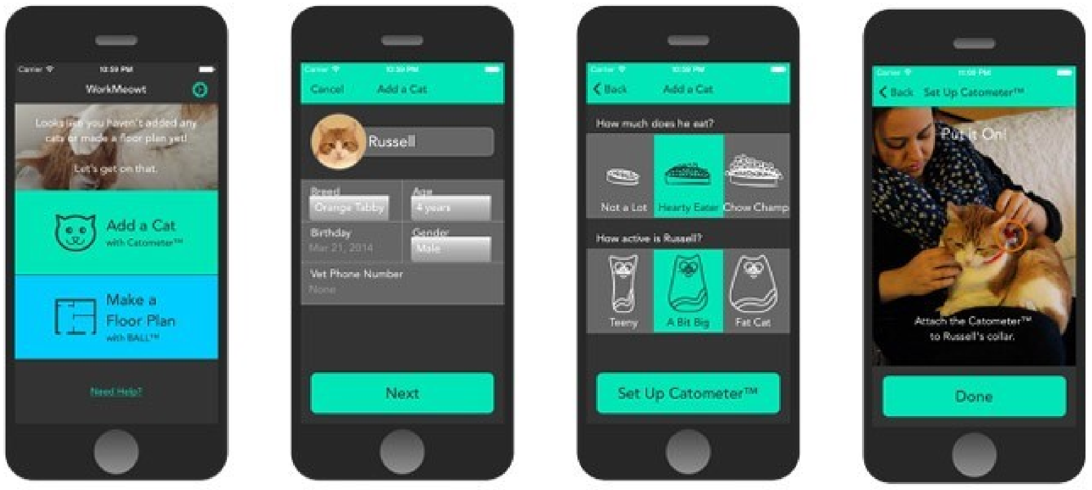


Figure 3.4: Prototype of a web application

The works mentioned previously, transform a hand-drawn mockup into a prototype by utilising Deep Neural Networks (DNNs) to detect the objects that were drawn in the mockup. Then, they identify the corresponding User Interface (UI) element for each object. Finally, they generate a prototype using the result obtained previously.

3.3.1 Sketch2Code: Transformation of sketches to ui in real-time using deep neural network

Jain *et al.* [JAB+19] mentioned the main problem of generating a prototype from a sketch and divided it into three sub-problems. The first one is to recognise the objects in the hand-drawn mockup and to define bounding boxes for each detected object. The objects inside these bounding boxes are associated to a corresponding User Interface (UI) component. The bounding boxes can overlap some times, and the second sub-problem solves this issue. The last sub-problem is to generate the prototype with the elements previously identified.

These authors utilise a DNN based object classifier to tackle the first sub-problem where they map the input image with a set of elements. The training of the DNN can be done independently of the language or platform because of this mapping. It is also in this phase in which the bounding boxes are displayed in the image.

These authors utilised hand-drawn mockups from several persons as the dataset to train the DNN. In these sketches, ten UI elements were used. According to Jain *et al.*, these were identified as the most common elements on several websites and UI applications.

Jain *et al.* continues saying that the built DNN was based on the RetinaNet object detection architecture because of the better accuracy regarding other detection models as a result of the novel loss function. This loss function, which is called focal loss, outperforms other loss functions when training difficult to classify examples. Additionally, using Feature Pyramid Networks [LDG+17], the DNN can classify the elements and define the bounding boxes in one stage.

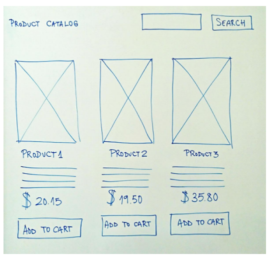
The result of the first sub-problem is a list of coordinates for the bounding boxes that delimit the UI elements on the mockup and a confidence score for each box. It is also added a prediction for the component inside each bounding box.

The overlapping of bounding boxes can occur if two elements are near each other or if one component is recognised as two different elements. The first case can typically happen if the sketch contains a form because it is usual to have buttons near text. They also identified that specific elements usually appear together. The DNN can misinterpret the image which originates the second problem. One example of this case is a check-box with a label that can also be interpreted as a check-box with a title.

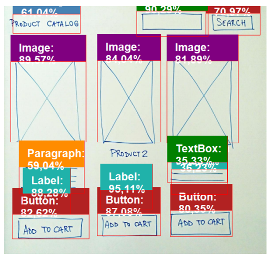
The solution to this problem is to modify the output of the first sub-problem checking if the overlap is lower than 50% or not. If it is, then it is verified if the two elements detected are present in a dictionary containing pairs of common elements that overlap and if they usually are represented through properties or parent components. In the case that a couple of elements is present in the dictionary, the elements are represented as a component that contains both elements. In the case that the pair of elements is not in the dictionary, the elements are expressed according to the result of the dictionary. If the overlap is higher than 50%, it is just kept one of the components according to a priority list.

The last sub-problem uses the list without conflicting bounding boxes to generate the prototype. The notation utilised in the UI representation does not vary from the one utilised to develop HyperText Markup Language (HTML) files. These authors deployed a UI parser using React[13] in a NodeJS server. The file containing the prototype is an HTML document.

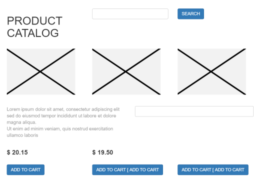
Figures 3.5a, 3.5b, 3.5c show a hand-drawn mockup, the prediction of the DNN and the result obtained. These figures were obtained from the website develop by Jain *et al.*[14].



(a) Hand-drawn mockup



(b) Prediction



(c) Result

Figure 3.5: From a hand-drawn mockup to prototype

3.3.2 pix2code: Generating Code from a Graphical User Interface Screenshot

Belmatrelli [Bel18] also divides the problem into three sub-problems. The first sub-problem is the same as the first sub-problem of Jain *et al.*, that is, identifying the objects present in the image and their corresponding elements and positions. The second sub-problem is to define a domain-specific

[13]https://www.npmjs.com/package/react

[14]https://sketch2code.azurewebsites.net/

language that easily describes the results obtained in the first sub-problem. The final sub-problem is to combine the other sub-problems and generate a textual description of the objects detected in the first sub-problem using the language of the second sub-problem.

Belmatrelli utilised a Convolutional Neural Network to solve the first sub-problem because this Neural Network is capable of performing unsupervised feature learning by mapping an input image to a learned fixed-length vector. First, they resize the input images to 256 x 256 pixels, and the values of the pixels are normalised. Then, the images are fed in the Convolutional Neural Network. They encode each image to a fixed-size output vector by using small receptive fields. Belmatrelli continues saying that the width of the first convolutional layer is 32, followed by a layer having a width of 64, and the last layer has a width of 128. The vision model is complete when two layers of size 1024 are fully connected.

According to Belmatrelli, they utilised a domain-specific language to describe the UI. It was selected a part of all elements that can be created to have a simple domain-specific language. The defined language utilises brackets to establish a hierarchy. The children elements of one parent element are inside a block. They used a Long Short-Term Memory neural architecture because traditional Recurrent Neural Networks are not able of modelling relations as the one described previously.

The model was trained by feeding an image and a contextual sequence of tokens as inputs. The Convolutional Neural Network encodes the image into a vector. The series of inputs is encoded by the Long Short-Term Memory language model into an intermediary representation. The vision-encoded vector and the language-encoded vector are combined into a single vector which is provided to a second Long Short-Term Memory model that decodes both representations. Therefore, this decoder gains knowledge on the relationship between the objects present in the image and the corresponding tokens present in the domain-specific language code. The output layer of the second Long Short-Term Memory model contains the same number of cells as the size of the vocabulary.

The get the domain-specific language code of an image, it is provided to the model the image and a sequence containing 48 tokens which are empty. The last symbol of this sequence is a unique token, $< \text{START} >$. The model utilises this token to start the prediction and uses the predicted token to update the next series of symbols. This process is repeated until another unique token, $< \text{END} >$, is generated by the model. The generated code can be compiled to the desired target language.

3.3.3 Generating a website from a paper mockup

The main objectives of Robinson's work [Rob19] was to create an application which translates a hand-drawn mockup into code and to compare classical computer vision techniques with deep learning methods. The tool developed by Robinson required a dataset containing sketches of websites, but such dataset was not available. Therefore, Robinson decided to find sites and automatically sketch them.

Robinson focused on finding pairs of a sketch version of the website as well as the corresponding code of the website. During this process, he identified six problems:

1. Hand-drawn mockups have a smaller element set than HTML;
2. In wireframes, the style does not vary while in a webpage there can be several styles for elements;
3. Sketches are static, and the structure of a website can be modified using JavaScript;
4. Wireframes and webpages contain a structure. However, webpages also include content which can modify the structure;
5. HTML sometimes is invalid or poorly formatted which reduces the quality of the dataset;
6. There can be several HTML representations for the same structure.

In order to address these problems, he developed a process to normalise the website where each one of these problems was solved. The first problem was solved by grouping elements of the same type. The second one was solved by replacing all styles from the elements with consistent values. The third problem was tackled by merely disabling the JavaScript. He changed the width of the elements to a fixed amount in order to solve the fourth problem. The extraction of the websites consisted of normalising a webpage, screenshotting the normalised website, extracting the elements, and deducing the structure of the webpage.

Afterwards, he created 18 sketches for each of the five classes of elements that were selected, and for each website, he replaced all the elements by a corresponding mockup that was chosen randomly. He also transformed the sketches by making a translation, rotation and scaling of the element but not so large as to change the structure of the website. The dataset consists of 1750 sites which were normalised and screenshotted. Afterwards, the structure was obtained, and it was generated the sketch for each website.

Then, he developed a framework that receives an image and removes the background from the sketch. It also eliminates noise from the picture due to lighting and rotates the image to be ready to feed the part of the pipeline where the objects of the image are identified and recognised. The result of this step is a code in a domain-specific language that represents the wireframe, and it is translated to HTML code. Afterwards, the website is updated.

He implemented two approaches to identify the objects and compared the results obtained by both. The first method uses computer vision to detect the elements. This method is composed of four phases. First, the elements are recognised and then their position, size and type are classified. Secondly, it is generated a hierarchy tree from all elements. Thirdly, containers are identified to create a correct structure. The last step is to correct errors that were made in the sketch. The second approach utilises deep learning to translate a wireframe into a domain-specific language code. An Artificial Neural Network was used to learn how an image of a wireframe relates to a picture of the result. This Neural Network can be used to translate the image of a wireframe to a structured image which is similar to a normalised website. This structured image is used to classify the elements and containers.

3.3.4 Live web prototypes from hand-drawn mockups

As describe previously, DNNs need a large amount of data, and there are not available datasets containing sketches. So, Ferreira [dSF19] implemented a synthetic mockup generator to fix this problem. This generator can create images of hand-drawn mockups that simulate the drawing of a human.

This simulation starts by creating a canvas that has a fixed width and height. Then, it is calculated the width and height of cells by multiplying a fixed value by a random amount which means that each sketch differs from the others. Afterwards, the canvas is divided into cells, and since some space can be unused, this space is used to calculate an offset for the starting point of the working area. The working area is composed by the space that all cells occupy in the canvas. The next step is to assign containers to cells. These containers must contain at least two cells. Then, it is defined the area for each element, and it is placed a random element. The last step is to draw the corresponding elements in the assigned place. To make the mockup look like it was designed by a human, the points that define the shape of the element can suffer a small translation. Additionally, instead of having straight lines between the points, it is applied a distortion on some points of the line.

A pipeline that acquires the image and then detects the elements presented in the picture was created to transform the sketch into a prototype. After this step, the program makes a hierarchical reconstruction and, finally, generates the HTML and CSS code. During the image acquisition phase, the image obtained is transformed into a high contrast version due to the use of high contrast image in the training of the DNNs. It was utilised You Only Look Once (YOLO)[15] to detect the elements of the mockup and a combination of Pix2Pix[16] and YOLO to identify the containers.

The hierarchical reconstruction phase receives a list of objects detected as input without any order. Firstly, the objects that are in the same container are aggregated and are set as children of the container. Afterwards, the elements that the work classifies as annotations are joined together with the associated container. Then, using overlapping techniques, check-boxes and radio buttons are joined with the corresponding texts. Next, the pipeline identifies elements that are near each other and that are of the same kind and groups them. Finally, the elements are aggregated into lists "to generate a more concise and aligned hierarchy."

The last step of the generation of the prototype is to create the HTML and CSS code using the hierarchy defined in the previous step. The creation of the code starts in the root of the hierarchy and continues using a deep-first algorithm. If a container has annotations, CSS tags are generated according to these annotations.

3.4 Programming using Natural Language

According to Rosenblatt *et al.* [RCHB18] and Arnold *et al.* [AMG00], it is necessary to develop tools that enable users with upper-body restrictions to program. The solution to this problem is to

[15] https://pjreddie.com/darknet/yolo/
[16] https://phillipi.github.io/pix2pix/

use speech recognition.

Rosenblatt *et al.* [RCHB18] developed an editor that allowed to program using code. According to them, this editor allows users with upper-body restrictions to program quickly and precisely. This editor is a web application that contains two components: an automatic speech recognition and a syntax parser. The user speech is recorded and recognised by the editor. Afterwards, the recognised speech is then passed to a syntax parser.

Arnold *et al.* [AMG00] developed a generator for voice recognition programming environments. The input of this generator is the context-free grammar of a programming language. The generator outputs a programming environment where the developer can use his voice to write programs. This programming environment contains a voice recognition system that aids the programmer by giving automatic completion of the program text.

Cozzie *et al.* [CK12] developed a non-deterministic version of a traditional natural language programming system. This program generates all equivalents for the natural language input and verifies if one of them passes in a unit test created by the developer. It also has a probabilistic model that permits handling large problems with several rules. The system generated correctly 55 of the 69 test problems.

Price *et al.* [PRZH00] developed a program to understand the instructions that were given by the user without having to know the exact syntax of the programming language. However, the program only supports a subset of Java and only handles written natural language.

According to Begel [Beg04], most of the existing software that allows programming using natural language require too much input to search and navigate in the program. He also says that searching and navigating in this software is too slow and is more complicated than using a mouse and a keyboard. Therefore, he developed a system that allows writing Java programs using a more natural language. This language is semantically identical to Java.

3.5 Conclusion

Since the creation of the first CAD system, it dramatically changed how engineers and designers work. These tools contributed to shorten the duration of the development of a product and also lowered the product development costs. Some of these software permits the creation of objects using a command-line such as AutoCAD.

Rahman [RAMI17] explains that there are three types of chatbots. The Nonprogramming Chatbots that do not require advanced programming skills such as Chatfuel and ManyChat. The Conversational-Oriented Chatbots have more dynamic conversations with their users because they utilise a domain-specific language. Finally, there are the Chatbots developed by tech giants.

Nowadays, some of the largest technology companies in the world are developing their own smart assistants, for instance, Google's Assistant, Amazon's Alexa and Apple's Siri. Additionally, some of these companies allow developers to create their bots by using the same system that the companies use in their intelligent assistants. Some of these systems are Dialogflow, Amazon Lex and IBM Watson Assistant. This strategy enables companies to gather more data to validate their systems.

There are some works that aim at transforming sketches of webpages to prototypes. These works aim at reducing the duration of the development cycle of a prototype. These works utilise DNN's to detect the objects that are present in the sketch.

Finally, speech recognition is being used to permit persons with upper-body restrictions to the program. These works recognise the voice of the user and translate it into text. Afterwards, the text goes through a parser that is able to determine the command that the user intended to execute.

Chapter 4

Problem Statement

This chapter describes the limitations of the current solutions to improve the efficiency of the developing cycle of a web page prototype and presents a possible approach to tackle these problems. Section 4.1 details the problem definition. Section 4.2 presents the proposed solution. Section 4.3 explains the main challenges that are associated with the problem. Section 4.4 presents the thesis statement. Section 4.5 details the research questions that were addressed in this work. Section 4.6 explains the validation methodology used to measure the impact of this work. Finally, Section 4.7 summarizes the problem statement and the solution that was implemented.

4.1 Problem Summary

The development of a prototype of a website is a time-consuming and challenging process. Designers represent their ideas in hand-drawn mockups while communicating with the client. Then, the designers transform these sketches into an intermediate representation that is used by programmers to create the prototype of the webpage. Afterwards, the client evaluates if the prototype has the functionalities he/she asked for and some adjustments may be included. The repetition of this cycle means that resources and time are wasted. Therefore, reducing the duration of the development of a prototype is essential both for the company and the client.

Despite this problem being addressed by transforming images of mockups into prototypes [JAB+19, Bel18, Rob19, dSF19], this type of software uses a small portion of the elements that can be created in a webpage. Furthermore, the designer still has to do a sketch of the website, and he/she can only change the appearance of the elements after the prototype being generated. Therefore, designers should keep clarifying the mockups, which can be a tedious process.

4.2 Proposal

This project aimed at developing a conversational interface that allows the programmer to create a prototype without requiring a mockup. This work also involved the creation of a program to enable users to visualise the effect of their actions immediately.

The program should allow the user to add and eliminate HyperText Markup Language (HTML) elements to the prototype that is being developed. Additionally, users should be allowed to change the visual aspect of the elements. Therefore, the program should be able to change the position of the elements easily. Another requirement is to permit the user to retrieve the final HTML and Cascading Style Sheets (CSS) code. One possible sequence of interactions between the user and the program is detailed below:

User: "Add a container"

Program: "Adding a container"

User: "Add a p"

Program: "Adding a p"

User: "Change text"

Program: "What is the text?"

User: "Lorem Ipsum is simply dummy text..."

Program: "Changing text to Lorem Ipsum is simply dummy text..."

User: "Go back to the container"

User: "Add an image"

Program: "Image added."

User: "Change source to img.jpg"

Program: "Changing source to img.jpg"

User: "Go back to the container"

User: "Same row"

Program: "Changing same row"

User: "Download files as example"

Program: "Downloading files as example"

In this example, the program adds a container in the first place. Inside the container, there must be a paragraph and an image side by side. There should be a text inside the paragraph, and the source of the image is img.jpg. In the end, the program should download the HTML and CSS files that contain the result of these interactions. The visual result of these operations is shown in Figure 4.1.

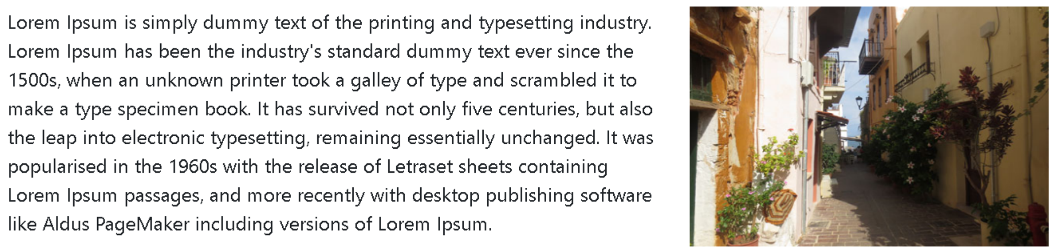

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Figure 4.1: Expected result of the interaction between the user and the system

Additionally, the result of these operations must be shown to the user almost immediately. Otherwise, the user would not be able to evaluate and analyse the result of his actions quickly.

4.3 Fundamental Challenges

The ability that humans have to understand the meaning of a book, a paragraph, a sentence or even a word is, nowadays, taken for granted. To utilise machines to do this intrinsic ability that humans have means that some difficulties and challenges must be addressed and solved. The problem becomes even more challenging because code must be generated from the user input in a way that it gives the user immediate feedback. Therefore, this work required an understanding of various areas and many decisions were assumed. The following paragraphs detail some of the challenges that were addressed in this work:

1. **The definition of a domain-specific language.** A domain-specific language was defined before implementing the software. This language should be easy to comprehend, and it should not be ambiguous. It should also allow users to do the same operations that they can do with a keyboard and a mouse.

2. **Understanding the context of the user input.** During a conversation between humans, it is possible to comprehend sentences that require an understanding of the previous phrases. However, machines do not easily perform this task.

3. **The acceptance of operations that modify the aspect of elements.** Users can easily add and remove CSS code with conventional input devices. So, the software should allow operations that result in changing the appearance of the elements.

4. **Show almost immediately the result of the operations.** Live programming is difficult to implement. However, this project should implement it to fulfil its objectives.
5. **Non repetitive interaction.** One of the most difficult aspects of creating a conversational interface is to make it non-repetitive.

4.4 Thesis Statement

In this work, it was admitted that a conversational interface developed with the purpose of generating webpages allows the creation of complex webpages in a short amount of time. Therefore, the following hypothesis was proposed for this work:

"A conversational interface that allows the generation of complex webpages using an easy to understand domain-specific language is more comfortable to utilise than the current forms of developing a prototype."

4.5 Research Questions

From the thesis statement specified above, there are some research questions that emerge. The following list contains the research questions:

RQ1: *Are users able to create a complex webpage using a conversational interface?*

It is not clear that a conversational interface is adequate for the creation of complex web prototypes. We addressed this question by developing a chatbot that permits the generation of complex webpages and by evaluating this tool with developers.

RQ2: *Is it possible to specify an easy to understand domain-specific language aiming at generating webpages?*

A domain-specific language is required to develop a conversational interface. This domain-specific language should replicate the actions that developers can execute while writing HTML and CSS code. We answered this question by creating a domain-specific language and by executing an experience that permitted the participants to interact with the conversational interface.

RQ3: *Would front end developers use a conversational interface to develop webpages?*

Front end developers can develop web prototypes by transforming manually or automatically hand-drawn mockups. We assessed if front end developers would consider a conversational interface to develop web prototypes instead of the forms described previously.

RQ4: *Would back end developers use a conversational interface to develop webpages?*

Back end developers usually do not develop web prototypes. One reason is that creating prototypes is a time and resource-consuming task. We assessed if back end developers would start developing prototypes if they used a conversational interface.

4.6 Validation Methodology

As previously described, the main goal of this project was to develop a conversational interface that is capable of generating a prototype of a webpage.

Therefore, a domain-specific language that contains the possible actions that the user can do in the conversational interface was defined. This language was evaluated in two phases, (1) get possible descriptions of websites and compare them with the defined language and (2) determine if the defined language is easy to comprehend.

The conversational interface was also evaluated to determine what is the opinion of the users and if they would use it. This evaluation was done by testing the tool with users and ask them to create some webpages in order to follow their actions and assess the obtained results.

4.7 Conclusions

The goal of this project was to propose a domain-specific language to be used to perform operations on the layout of websites and a conversational interface where these operations can be executed. The user executes these operations using a conversational interface that was developed. The result of these operations must be shown almost immediately to the user.

As a result of this work, we developed a domain-specific language and a conversational interface that when paired enables the creation of webpages.

Chapter 5

Implementation

This chapter describes how the problem explained previously was addressed and details how the fundamental challenges were overcome.

Section 5.1 outlines the main aspects of the implementation by introducing technologies and strategies used to handle the various problems faced during the development of this project. Section 5.2 explains the possible interactions between the user and the chatbot. Section 5.3 describes how the chatbot was implemented and its details. Section 5.4 includes a number of comments on the implementation.

5.1 Overview

The first step in developing a chatbot is to specify the domain-specific language. Afterwards, the program that generates webpages using a chatbot can be developed. During the development of this program, it is required to receive the commands written by the user and interpret and understand the associated meaning. If the user does not provide all the necessary information for the chatbot to perform a specific action, the chatbot must be able to ask for this information. When all the required data is gathered, the chatbot should then execute the intended action.

5.2 Domain-Specific Language

Sections 5.2.1, 5.2.2 and 5.2.3 explain how the domain-specific language was defined. Section 5.2.1 details the operations that the user can perform. Section 5.2.2 explains the commands entered by the user for each operation as well as the questions and answers given by the chatbot. Section 5.2.3 describes some entities that were defined to aid defining the domain-specific language.

5.2.1 Defining the operations of the chatbot

There should be a corresponding command to the actions that programmers can perform while writing a program. Thus, it can be assumed that the chatbot will be able to **Add** HTML elements to the webpage. Since mistakes can occur when creating a webpage, the chatbot needs to have intentions that allow us to fix these errors. So, a **Remove** intention has to be included to enable the user to delete the element which is being edited, and a **Undo** intention can also be added to allow the user to reverse the effects of the last command. In a computer program where a undo intention is available, a **Redo** intention also exists.

At this point, the user can create an element in a webpage, but he/she can not change how it looks like or add a text to it. Thus, the chatbot must be able to **Change** the appearance of the elements or modify the text content of the element or if, it is an image, the source of this image. Consequently, several intentions can be added to allow the user to change the margin or to change the text content of the current element or to change the source of the image.

Now, the user can create a website, change the appearance of the elements and delete the current elements. However, the user does not have a way to fix errors on an element after passing to a new one. Therefore, the user should be able to change the element which is being edited to fix this problem. An intention that can be defined as **Select** can be added. Programmers can also create elements inside of other elements, generating a hierarchy. So, the chatbot must allow the user to move from an element to its parent and vice-versa. This means that two more intentions called **Enter** and **Exit** are included. One of the requirements is to get the final HTML and CSS files which means that one more intention must be defined that can be termed as **Download**.

As of this moment, some commands that will aid the user during the creation of the website can be added. Thus, we can have an intention that allows the user to **Save** the current webpage as a project and one to **Open** a previously saved project. The chatbot can also have an **Upload** intention that enables the user to change the webpage according to a provided HTML and CSS file and an **Upload Image** intention that permits the user to upload an image to the server. It can also be added an intention that allows the user to **Make** a certain number of copies of the current element as well as an intention to show the possible commands that the user can write depending on the current state termed as **Help**. It was also implemented a functionality that allows the user to distinguish the created elements. This functionality can be enabled with the **Show Borders** intention and disabled with the **Hide Borders** intention. Finally, it is necessary to include an intention that allows the user to invalidate the current action in the case that the chatbot asks him for an entity that he/she did not give in the initial command and he/she no longer wants to perform this action. This intention is termed as **Cancel**.

5.2.2 Definition of the commands, answers and questions

After defining the intentions that determine what operations the chatbot can execute, it is necessary to identify the commands entered by the user and the answers given by the chatbot for each intention.

5.2.2.1 Add

So, for the **Add** intention, there are several possibilities for the instruction such as "Add h1" or "Add navbar" or "Add image". These examples demonstrate that this command requires an entity that determines the elements that the user wants to create. After deciding on the commands for the Add intention, it must be chosen an answer for this command which can be "Adding ELEMENT", where ELEMENT is the element specified by the user. This means that the answer changes according to the element specified in the command. If the user does not provide the element in the instruction, the chatbot must ask for it, and a possible question is "What element do you want to add?".

5.2.2.2 Select

The command for the **Select** intention can be "Select h1" or "Select navbar" or "Select image". These alternatives depend on what the user wants to select. Since the elements that can be created must be the same as the elements that can be selected, the Select intention utilises the same entity as the Add intention.

The commands described previously only work to create one element of each type. So, the user should provide an ordinal to pinpoint the element and the command changes to "Select ORDINAL ELEMENT". The answer given by the chatbot will depend on the element and the ordinal provided in the command and can be, for example, "Selecting ELEMENT ORDINAL". In the case the user does not write an element, the chatbot should ask "What element do you want to select?". If the user does not provide an ordinal, but there is only one element of the type he/she wants to select, the chatbot should not request anything. However, if there are more than one, the chatbot should make the question: "Which one?".

5.2.2.3 Change the Appearance

The commands that allow **Changing** the appearance of the webpage should be divided according to the type of value they require. For example, to change the colour of the text in an element, it is necessary to provide a hexadecimal code and to change the margin of an element. Several values can be accepted to change the margin, such as 1em, 0.5px, 2rem and 1.5%. Therefore, one intention can handle the attributes that require hexadecimal such as colour, background colour and border colour. Another intention can take care of the attributes that need an URL, for example, the source of an image and the logo of the navbar and another one can be used to manage the attributes that require values such as margin, padding, width and height.

Three more intentions were created, one to handle the attributes that require text like text-align, border, the name for the navbar and the text for the elements. Another one allows the user to change the links of the navbar. This intention must be different from the one associated with the text attributes because the number of links can vary. And, finally, one intention for the attributes that do not require any value such as a command that allows the user to have multiple elements on the same row and another command that does the same but on the same column. For these intentions, the command was the attribute that the user wants to modify, followed by its value. The answer given by the chatbot can be "Changing ATTRIBUTE to VALUE", where the user provided the ATTRIBUTE and the VALUE. If the user does not provide the value or the value provided does not match the type of the attribute, the chatbot should ask for it, and the answer depends on the attribute given by the user. If the type of value is colour, the chatbot asks "What is the hexadecimal of the colour?". The chatbot asks "What is the URL of source?" if the type is an URL. If the user does not provide value to the margin and padding attributes, a default value was adopted. The question associated with the width attribute is "What is the value of the width?", and a similar one was used for the height attribute. "What is the text of ATTRIBUTE?" is the question asked when the attribute is of the text type. If the user wants to change the links of the navbar, it is first asked how many links the user wants to specify and, afterwards, it is asked the text for the link as many times as the number previously given.

5.2.2.4 Make

If the user wants to **Make** copies of the element he/she is currently editing, he/she can use the command Make followed by the number of copies he/she wants to obtain. The chatbot answers saying "Making NUMBER copies", where NUMBER is the number of copies. If the user does not provide a number, the chatbot asks for it with the question "How many copies?".

5.2.2.5 Download, Save and Open

The **Download**, **Save** and **Open** intentions are almost the same because the user can give the name of the intention followed by any word. If the user just gives the name of the intention, the chatbot asks for the word with the question "What name to give to the files?", "What name to give to the project?" and "What project do you want to open?" for the Download, Save and Open intentions respectively. The answer given by the chatbot is "Downloading NAME", "Saving NAME" and "Opening NAME" for the Download, Save and Open intentions accordingly, where NAME is the word provided by the user.

5.2.2.6 Remaining intentions

The **Enter**, **Exit**, **Undo**, **Redo**, **Clear**, **Upload**, **Upload image**, **Remove**, **Cancel**, **Help**, **Show Borders** and **Hide Borders** intentions are almost equivalent because the user does not need to give any more details apart of the name of the intention. Therefore, there are not questions associated with these intentions. The answer given by the chatbot is the name of the intention followed by the suffix -ing with the exception of the Undo and Redo intentions where the answer equals the name of the intention.

5.2.2.7 Alternative definition of the commands

The commands entered by the user can be the ones explained previously or phrases containing these commands. For example, the user can write "change margin to 1em" instead of "margin 1em" or "edit margin". The commands "download files as example" and "download example" are equivalent. This functionality approximates the conversation that a user can have with the chatbot to the communication between two humans.

5.2.3 Entity Definition

From all elements that can be created in a webpage, the following ones were selected to be included in the domain-specific language: `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `navbar`, `p`, `button`, `div`, `container`, `form`, `input`, `unordered list`, `ordered list`, `list item`, `image`, `label`, `span` and `footer`.

There are several attributes that enable changing the appearance of the elements, and just a part of these attributes were selected which are: `border`, `border style`, `type`, `text` (lets the user change the text of the element), `text align`, `name` (allows the user to change the name of the navbar), `logo` (permits the user to change the logo of the navbar), `source`, `margin`, `padding`, `height`, `width`, `border width`, `colour`, `background colour`, `border colour`, `same row`, `same column` and `links` (allows the user to change the links of the navbar).

Finally, it is necessary to have an entity to handle all the values accepted by margin, padding, width and height. This entity is termed value.

5.2.4 Domain-specific language Configuration file

Appendix A (p. 73) shows the file that contains the specification for the domain-specific language defining the entities, specifying the intentions as well as the commands entered by the users, detailing the questions asked by the chatbot and the given answers. This file is used to configure the chatbot when the server is started.

5.3 Chatbot Implementation

The chatbot was divided into two parts, one that handles the recognition of the commands entered by the user and returns the appropriate answer or question. This part was developed in a server using NodeJS, and it was utilised a Node package called Node-Nlp[1] to recognise the intention and other vital information from the command entered by the user. This package enables the programmer to classify the intent of an utterance and to get an answer from this intention. It can also return a

[1] `https://www.npmjs.com/package/node-nlp`

question if a piece of information is not in the utterance. The second part of the chatbot is an HTML page that contains several Javascript files where there are functions that send the commands written by the user to the server and receives the question or answer that was obtained by the Node-Nlp package. In these files, it is also executed the intended action of the command written by the user. The user can write the intended actions on a chat, where he/she can also see the questions of the chatbot. In this conversation panel, it is also shown information that helps the user to understand how the chatbot works.

Section 5.3.1 describes the implementation of the Server and the respective files. Section 5.3.2 explains how the JavaScript part of the chatbot was developed.

5.3.1 Server

As previously indicated, it was utilised a package called Node-Nlp to allow the developer to define the intentions of the commands entered by the user, the phrases that determine a specific intention and the answer for a particular intention. This package also permits to define entities which allow the developer to establish a correlation between several words that have the same meaning for the chatbot. The commands entered by the user and the answers given by the chatbot can contain entities. The developer also can write questions in case the user enters a command, but the command does not have an entity that is required.

After defining the domain-specific language, several functions were developed to read the file with the domain-specific language and to add all the information to the Node-Nlp framework. Then, it was done a validation in order to check if all the commands work as expected. When this validation was finished, it was noticed that the Text attribute, the Link, the Download, the Save, the Open, the Select and the Cancel intentions did not work as planned. In the next paragraphs, we will detail the reasons for the detected problems and how they were fixed:

1. the Select was not working properly because this intention needs to access the webpage in order to check how many elements of the type the user wants to select;

2. the Cancel intention was being recognised, but it did not behave as expected, so it was necessary to alter the results obtained from the Node-Nlp in order to fix this problem;

3. the remaining intentions did not work properly due to the fact that the Node-Nlp was not capable of recognising entities that can be any word. For example, when a user wants to modify the text of an element, any word after the text command should be accepted. In order to correct this problem, a number of checks were done after getting the result from the Node-Nlp framework to assess if the intention obtained is equal to one of the described previously. If so, the result obtained is modified according to the intention acquired.

Then, it was created a server whose primary function is to receive the commands written by the user in the conversational panel and return the appropriated answer or question. When a command is received, it is checked in the first place if the user wants to create an element or not. If he/she wants to do so, then it is verified if the command contains the word "with". If so, the command is split into two parts with the centre in the word "with". The first part of this split contains the information for the Add intention, and the second part contains additional commands specified by the user that are executed after the creation of the element. All the commands specified by the user are pushed to a stack that contains the commands to be interpreted. If the user does not want to Create an element or the Add command does not include the word "with", the command is simply pushed to the stack. Subsequently, all the commands are interpreted, consecutively. If the result of this interpretation is a question, the server returns the question to the client. Otherwise, the result is pushed to a stack that contains all the results that were obtained. When this procedure finishes, the stack with the results is sent to the client.

The server also needs to include functions to save projects and open them. When the server receives an order to save a project, it creates a folder with the name of the project, and it also creates two files with the HTML and CSS code that were sent by the chatbot. The uploaded images are also copied to a folder inside the project. When the user requested to open a project, the server starts by reading the HTML and CSS files inside the project with the name that was requested. Afterwards, the server copies the images to the appropriated location and sends to the client the HTML and CSS code.

When the server receives the order to upload the HTML and the CSS files, it uses a package called htmlparser2[2] to interpret the HTML code and sends the result to the chatbot. When it is requested to upload an image, the server saves the file in the proper folder.

5.3.2 Client

To allow a faithful representation of the actions that the user is able to perform in writing a webpage in HTML and CSS, it was decided to create a webpage that would have an input where the user could write the commands and a place where he/she can see all the messages of the conversation. This conversational panel should not occupy any space of the webpage to develop, in other words, it should allow the creation of elements that can fill the whole width of the page and should be at the front regarding the elements of the webpage to develop. Otherwise, this conversational panel would have an impact on the webpage to create. The user should be able to change this panel from one side to another and to minimise it, to permit the user to have a better understanding of his actions. Figure 5.1 (p. 40) shows the webpage created with the conversational panel on the right side of the image.

As described previously, several Javascript functions were developed to send to the server the commands entered by the user, to receive the responses and to execute the intended actions. When the received response is a question, the chatbot writes it in the conversational panel immediately. However, if the user wants to change the appearance of the element but he/she did not provide the required value, the question is slightly modified to incorporate the name of the element that

[2] `https://www.npmjs.com/package/htmlparser2`

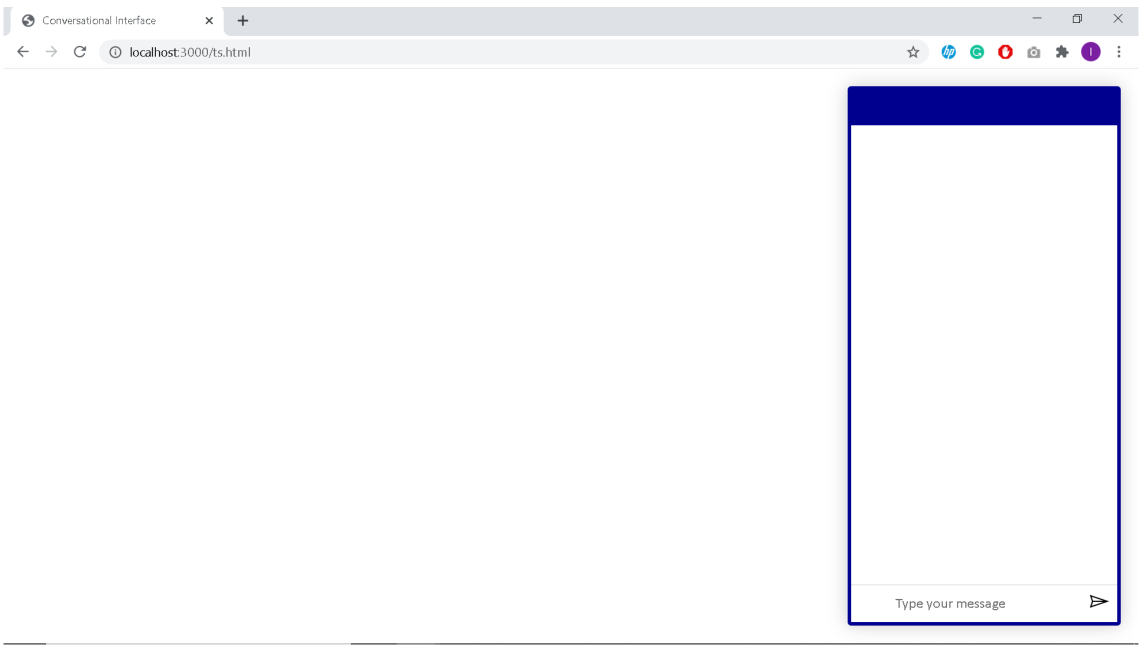


Figure 5.1: Webpage with conversational panel

the user wants to change and then it is added to the conversational panel. As an example, if the question is "What is the hexadecimal of colour?" and the current element is an h1, the question will be changed to "What is the hexadecimal of colour for h1?". This slight difference gives a more human aspect to the question.

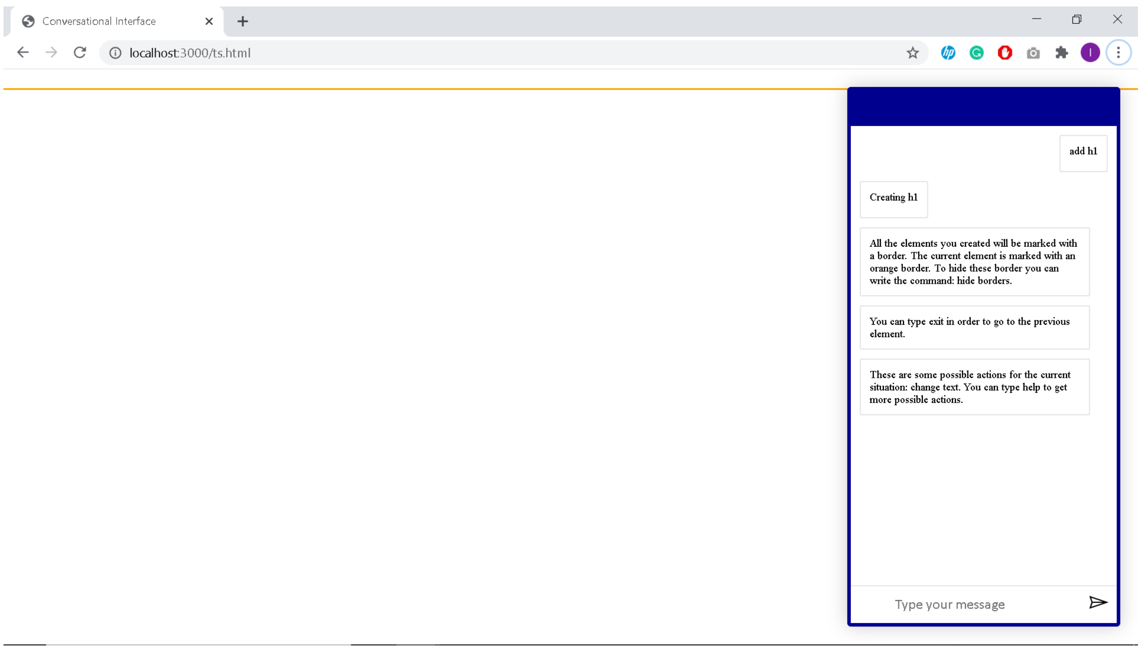
Later, a series of checks were created to determine the correct action to be executed by the chatbot from the result obtained from the server. This can be verified with the first word of the result because it is different for all the intentions. Then, it was implemented a class using the command pattern to allow the user to perform the **Undo** easily and **Redo** intentions. This pattern is only implemented for actions that will change the webpage, which are the **Add**, **Select**, **Remove**, **Change**, **Make**, **Enter** and **Exit** intentions. Then it was implemented another class that handles all the functionalities of the chatbot. These functionalities include adding one element to the webpage, changing the appearance of one element, removing one element, selecting one element and making copies of one element.

The following sections describe how the operations that the user can execute were implemented.

5.3.2.1 Add and Remove

When adding an element to the webpage, the chatbot creates the element request using the Javascript functionality to create elements and then appends this element to the element which the user is currently editing. Afterwards, the current element is added to an array containing the path from the central element of the webpage to the current element. The user is only permitted to interact with the webpage inside of this primary element. While creating the new element, it is also added an orange border to it, to identify better the element the user is working at the moment. It is also created an object of a class responsible for storing all the CSS code of the elements, and this object is added to an array. The process of removing an element from the webpage consists of removing the child elements in the first place. The corresponding objects that store their CSS are also deleted. Afterwards, the element the user requested to remove and the associated object with its CSS is removed. The parent of the element which was removed becomes the element which the user is editing.

Figure 5.2 shows the result of adding an element to the webpage and Figure 5.3 (p. 42) illustrates the result of deleting this element.



add h1

Creating h1

All the elements you created will be marked with a border. The current element is marked with an orange border. To hide these border you can write the command: hide borders.

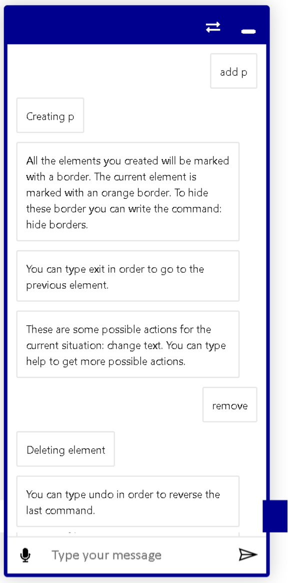
You can type exit in order to go to the previous element.

These are some possible actions for the current situation: change text. You can type help to get more possible actions.

Figure 5.2: Adding an Element

5.3.2.2 Change the Appearance

When changing the appearance of the elements, in the first place, it is checked if the action can be performed by all elements or it is specific for a type of element. For example, it is only allowed to change the *type* property if the user is editing an *input* element and it is only permitted to change the *source* of an *image* if the user is currently working in an *image* element. If it is a property that can be modified by all the elements, it is changed the style property of the current element and added the change to the object that stores the CSS. However, if the user wants to change the name, logo or links of the *navbar*, it is necessary to change the HTML code of the *navbar* by adding or changing a part of the HTML code of the navbar. If the user requested to change the type of input or the source of an image, this property is modified. If the user wants to change the text of the element, first the element must be one of the following: *h1*, *h2*, *h3*, *h4*, *h5*, *h6*, *p*, *button*, *li*, *span* and *label*. Secondly, it is changed the property *textContent* of the element to the value written by the user.



add p

Creating p

All the elements you created will be marked with a border. The current element is marked with an orange border. To hide these border you can write the command: hide borders.

You can type exit in order to go to the previous element.

These are some possible actions for the current situation: change text. You can type help to get more possible actions.

remove

Deleting element

You can type undo in order to reverse the last command.

Figure 5.3: Removing an Element

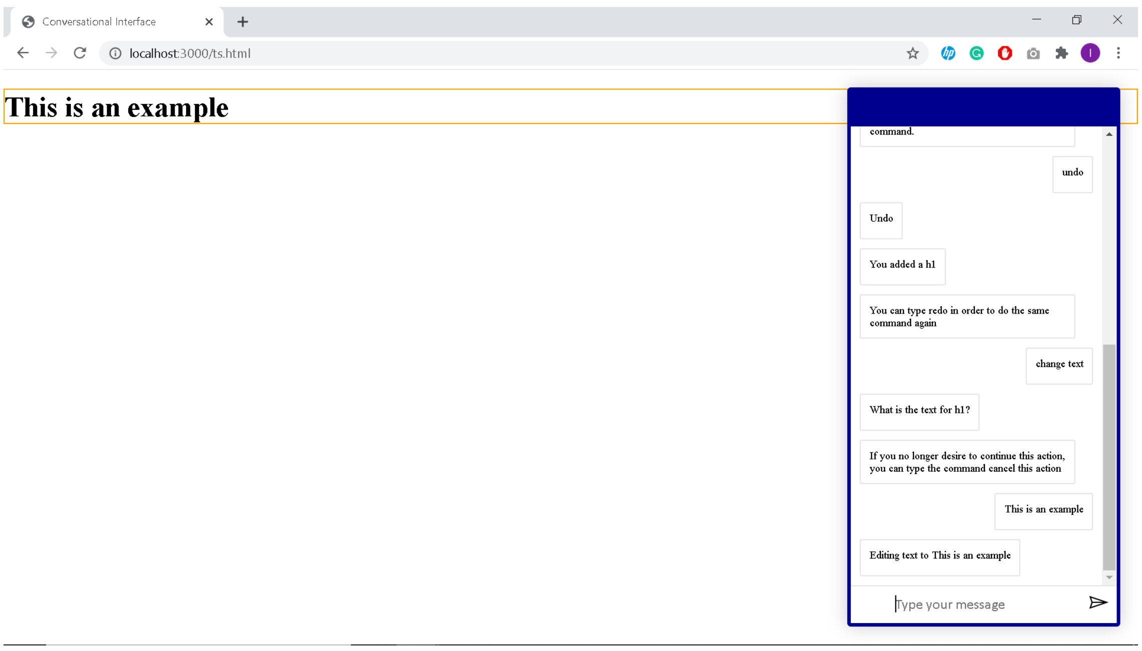
Figure 5.4 (p. 43), Figure 5.5 (p. 44) and Figure 5.6 (p. 45) illustrate the result of changing the text, the margin and the colour of an element, respectively.

5.3.2.3 Select

To select an element of the webpage, in the first place all the elements of the type that the user wants to select are gathered. Then, if there is only one element or if there is more than one element and the user specified the element he/she wants to select with an ordinal, the chatbot starts the process of changing the current element. However, if there are several elements and the user did not specify the element he/she wants, all the elements of the type the user requested are coloured and then the user is asked which element he/she wants to select. As illustrated in Figure 5.7 (p. 46), elements that have the same parent are coloured with the same colour but with a different colour of its predecessors or successors, if they exist. After the user specifies the element he/she wants, the chatbot starts the process of changing the current element. This procedure consists of changing the orange border from the element that the user was editing previously to the selected element. It is also modified the path from the central element of the webpage to the current element by iterating the selected element until it is reached the main element.

5.3.2.4 Make

To make copies of the same element, in the first place it is created a copy of the element that the user wants to copy, and then the id of the element and the child elements, if they exist, are changed. Afterwards, the copy is added to the webpage. This process is repeated until the number of created copies equals the number specified by the user. Then, the current element is changed to the last created copy as well as the orange border.



This is an example

command.

undo

Undo

You added a h1

You can type redo in order to do the same command again

change text

What is the text for h1?

If you no longer desire to continue this action, you can type the command cancel this action

This is an example

Editing text to This is an example

Figure 5.4: Changing the text property of an Element

Figure 5.8 (p. 47) shows the result of making two copies of one element.

5.3.2.5 Exit and Enter

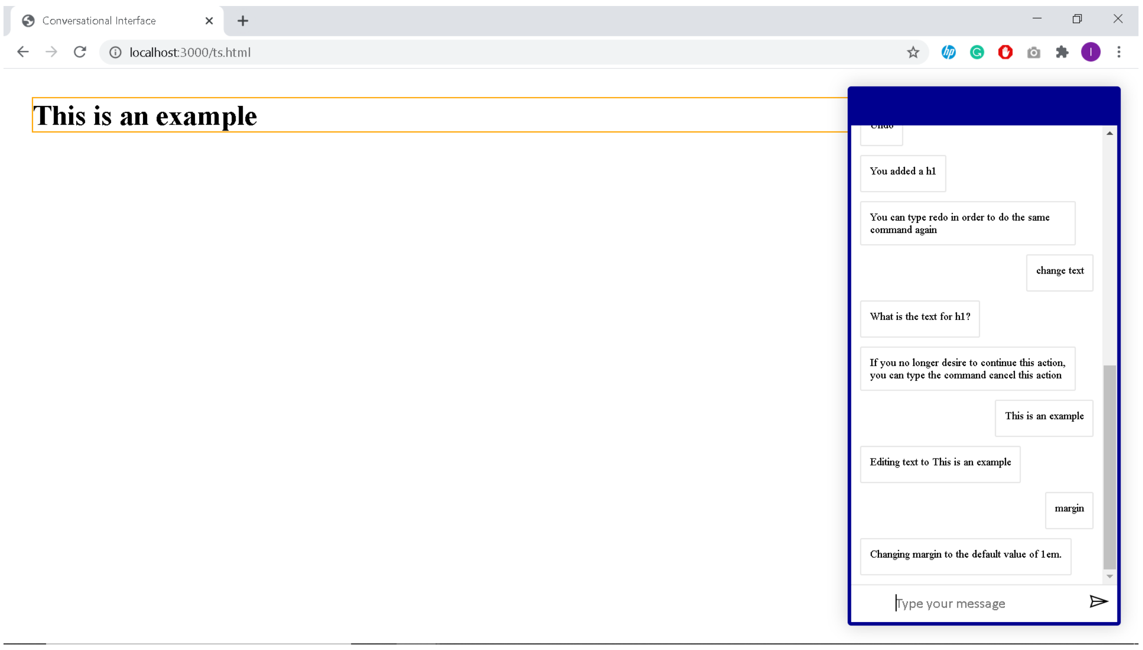
The Exit command allows the user to change the current editing element to the parent of the element the user was working. In the first place, it is removed the orange border of the previous element, and then it is updated the path that keeps track of the elements that exist from the current element to the main element of the webpage. Then, it is added an orange border to the parent element. It is also added a yellow background colour to the element that the user was previously working if the user wants to enter inside this element again. Thus, the Enter action adds the element with a yellow background colour to the path and then it modifies the colour of the borders of the element that the user was working and of the element that the user will work onwards.

5.3.2.6 Cancel, Clear and Download

The Cancel action does not have any effect on the client-side. The Clear intention removes all the elements created and resets the chatbot as if anything had happened. The Download action gets the elements inside the main element and then returns an HTML file with a webpage containing these elements inside of the body tag. It also retrieves all the CSS code stored in the objects detailed previously and returns a CSS file containing this code.

5.3.2.7 Undo and Redo

As described previously, all the actions that change the webpage or the chatbot variables are executed using the corresponding command. Using this approach simplifies the implementation of



This is an example

Undo

You added a h1

You can type redo in order to do the same command again

change text

What is the text for h1?

If you no longer desire to continue this action, you can type the command cancel this action

This is an example

Editing text to This is an example

margin

Changing margin to the default value of 1em.

Figure 5.5: Changing the margin property of an Element

Undo and Redo actions. The Redo executes the same action with the same values, and the Undo action does the opposite action of the previously performed action. So, for example, the Undo action for the Add intention is to remove the element that was created, and the Undo action for changing the appearance of the element is to change the same attribute to the previous value.

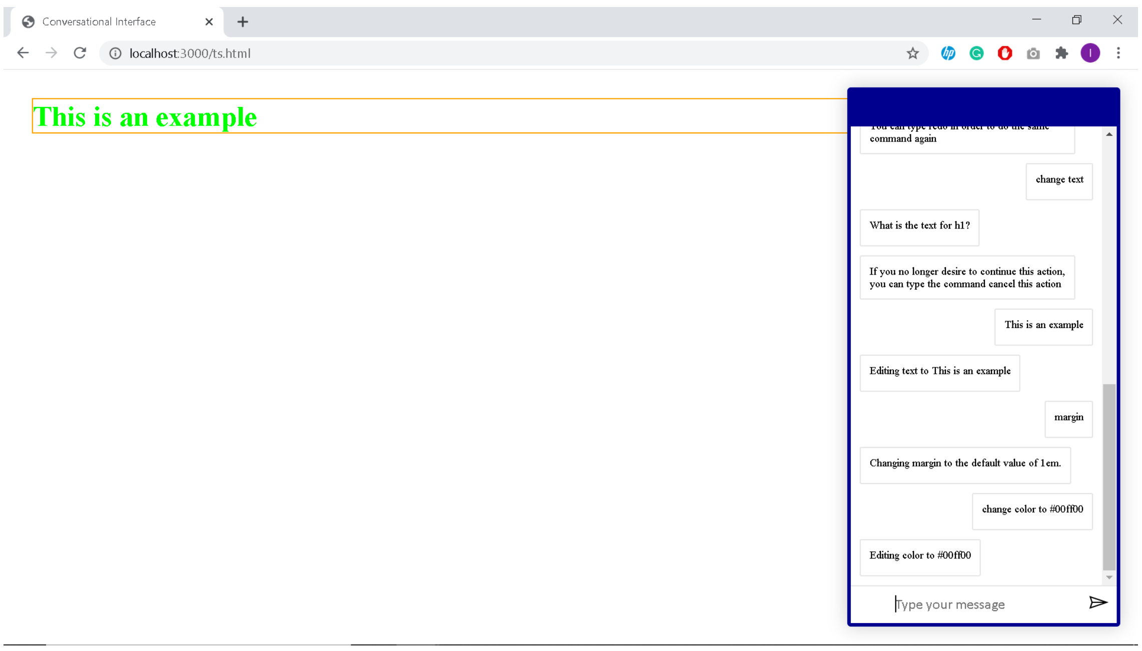
5.3.2.8 Show Borders, Hide Borders, Upload and Upload Image

A class from the main element is removed when the user requests to hide the borders of the elements. The appearance of these borders is a default option that permits a better visualisation of the position of the elements that were created. To enable this functionality, the class removed previously is added. When the user wants to upload an HTML and CSS file, it is first asked the user to select the two files, one at a time. Then, both files are read, and their content is sent to the server. When the server sends its response, the CSS file is interpreted using a library named CSSJSON[3]. With these results, the chatbot can add all the elements on the HTML file with the corresponding CSS code of the element. The process of uploading an image is similar to the operation of uploading the HTML and CSS file, but instead of sending a text containing the content of the file, it is sent a file with an image.

5.3.2.9 Save and Open

When the user wants to Save the current project, the HTML and CSS codes of the current webpage are sent to the server as well as the name for the project. When the user requests to Open a project,

[3]`https://github.com/aramk/CSSJSON`



This is an example

command again

change text

What is the text for h1?

If you no longer desire to continue this action, you can type the command cancel this action

This is an example

Editing text to This is an example

margin

Changing margin to the default value of 1em.

change color to #00ff00

Editing color to #00ff00

Figure 5.6: Changing the colour property of an Element

the project that the user wants to open is sent to the server. When the server responds, the chatbot can easily recreate the webpage using the same functions of uploading an HTML and CSS code.

5.3.2.10 Help

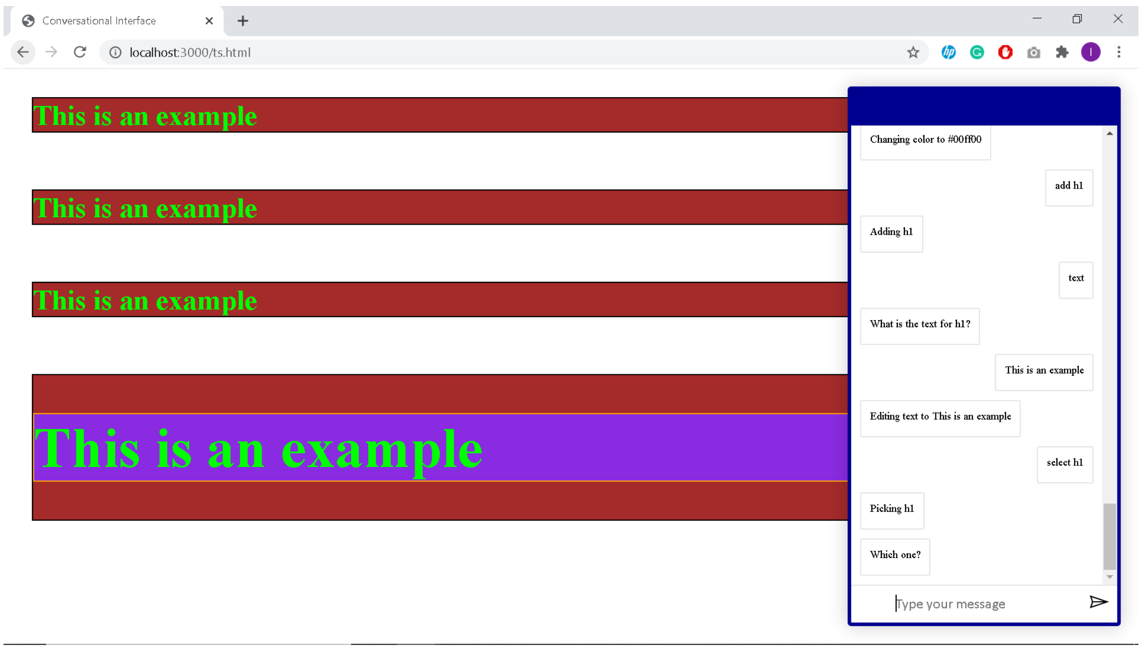
It was also developed a Help functionality that allows the user to ask for aid and the answer obtained from the chatbot depends on the element which is being edited. This functionality was implemented using the Template Method pattern, which permits each category of elements to have their own help function.

5.3.2.11 Speech Recognition

Finally, instead of writing the commands, it was added the ability to specify them in an oral way using the Speech Recognition library in JavaScript. This functionality is enabled by clicking on the microphone button on the conversational panel. This functionality allows the user to have a more natural level of interaction with the chatbot.

5.4 Conclusions

This chapter described the details regarding the definition of the domain-specific language. It was discussed the implementation details of the chatbot, which contains nineteen possible intentions. This chapter also addressed some aspects that allow the chatbot to have some human characteristics in its conversation as the help functionality that varies according to the current element, the ways that the user has to write the same command and the possibility of interacting with the chatbot in



This is an example

This is an example

This is an example

This is an example

Changing color to #00ff00

add h1

Adding h1

text

What is the text for h1?

This is an example

Editing text to This is an example

select h1

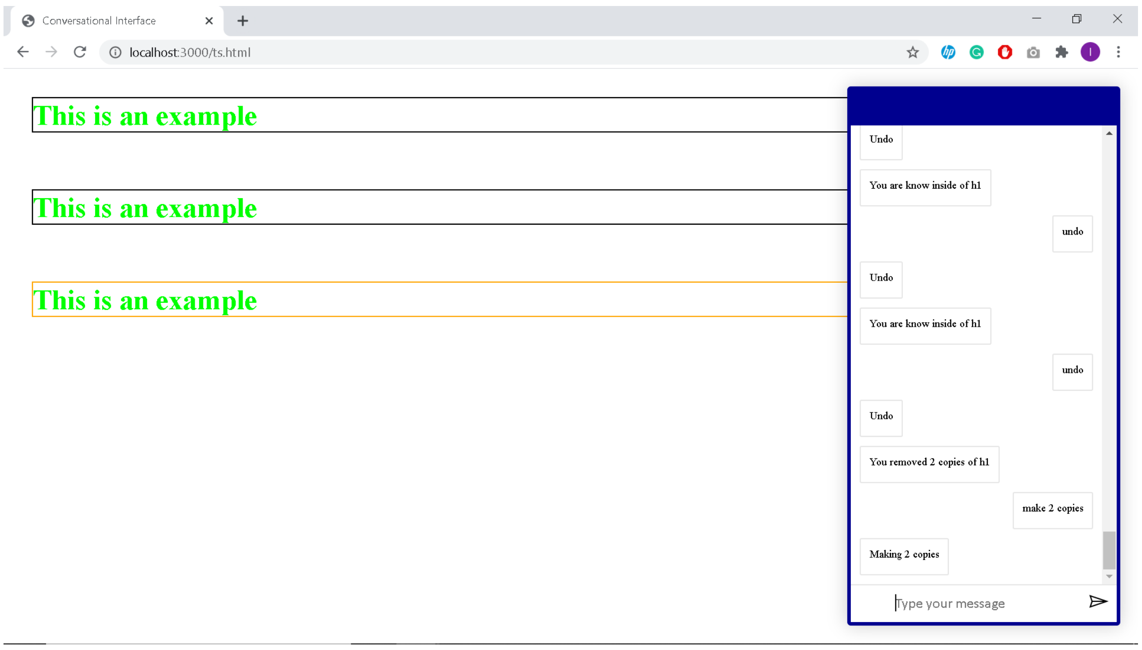
Picking h1

Which one?

Type your message

Figure 5.7: Colouring when selecting an element

an oral way. There were some difficulties while developing the domain-specific language and the tool. One of these difficulties was to develop a tool that would not have a repetitive interaction with the users.



This is an example

This is an example

This is an example

Undo

You are know inside of h1

undo

Undo

You are know inside of h1

undo

Undo

You removed 2 copies of h1

make 2 copies

Making 2 copies

Figure 5.8: Making copies of an element

Chapter 6

Validation

This chapter describes how the domain-specific language and the developed tool were validated, and the results gathered in Section 6.1. Section 6.2 describes the threats of the validation methodology that was utilised. Finally, Section 6.3 contains a number of final comments over the methodology used and the results obtained.

6.1 Validation Methodology

This section explains how the domain-specific language and the developed tool were validated. In the first place, a questionnaire was sent to programmers and designers to validate the domain-specific language. Afterwards, a controlled experiment was executed to evaluate the tool that was developed. With these two steps, we seek to answer the following research questions:

RQ1: *Are users able to create a complex webpage using a conversational interface?*

RQ2: *Is it possible to specify an easy to understand domain-specific language aiming at generating webpages?*

RQ3: *Would front end developers use a conversational interface to develop webpages?*

RQ4: *Would back end developers use a conversational interface to develop webpages?*

Section 6.1.1 explains how the domain-specific language was validated and presents the obtained results. Section 6.1.2 describes how the tool was evaluated and demonstrates the results that were gathered.

6.1.1 Domain-Specific Language Validation

A questionnaire was developed to determine if the domain-specific language contained the most common elements in a webpage as well as the properties that change the appearance of the elements. With this questionnaire, it was also intended to identify what commands the participants would use in a conversational interface and if these commands were present or had an equivalent in the developed domain-specific language.

6.1.1.1 Experimental Paramaters

The questionnaire that was prepared aimed at determining if the language that was specified contained the HyperText Markup Language (HTML) elements that the users usually create as well as the Cascading Style Sheet (CSS) properties. Another objective was to identity what commands the participants would use.

Experiment: A questionnaire was sent to the participants. The participants were asked to write commands that would replicate a certain webpage.

Participants: Ten answers were obtained to this questionnaire. Nine of the participants had a background in computer science, and the remaining one had a background in design. Due to the short amount of time available to get the feedback, the participants were mainly acquaintances or indicated/suggested by acquaintances.

Duration: The users were not required to complete the task in a certain amount of time.

Procedure: The users were divided into four groups. Each group had to replicate a certain webpage.

Environment: All of the experiments were executed in a remote environment. The users were not observed doing this experiment.

Data: All the commands written by the participants were gathered.

6.1.1.2 Task

The participants needed to analyse a webpage and write commands that the chatbot would be able to comprehend in order to generate the webpage. Appendix B (p. 83) shows the questionnaire that was sent to the participants. In this scope, four similar surveys were created. The only difference between the four inquiries was the webpage that was requested to be replicated. The main reason to create four questionnaires with one question each instead of one study with four questions was to reduce the time needed to fill it. The four webpages that were asked to be replicated are shown in Appendix C (p. 85). With the combination of these four webpages, a more extensive list of commands was identified, turning this task more accessible and more efficient when compared with the result if just one webpage was analysed by all the participants.

6.1.1.3 Results

Ten responses were received to this questionnaire, and each one of them was analysed. This analysis was done by interpreting each one of the commands that were written by the participants and identifying the elements that were used as well as the properties that change the appearance of the elements. Finally, it was also determined:

1. If the functionality or element or CSS property was possible to be executed;
2. If there was an equivalent functionality or element or CSS property;
3. If it was not possible to execute that functionality or element or CSS property using the domain-specific language described in Section 5.2, p. 33.

From the answers given to the questionnaire, 21 elements were identified. The following list specifies these elements.

1. **Paragraph or Text:** The domain-specific language contains this element;
2. **Div:** The domain-specific language contains this element;
3. **Input:** The domain-specific language contains this element;
4. **Title or Headings:** The domain-specific language contains this element;
5. **Button:** The domain-specific language contains this element;
6. **Br or Separator:** The domain-specific language does not contain this element;
7. **A or Links:** The domain-specific language does not contain this element;
8. **Footer:** The domain-specific language contains this element;
9. **Container:** The domain-specific language contains this element;
10. **Image:** The domain-specific language contains this element;
11. **Navbar:** The domain-specific language contains this element;
12. **Header:** The domain-specific language does not contain this element;
13. **List:** The domain-specific language does not contain this element;
14. **Card:** The domain-specific language does not contain this element;
15. **Carousel:** The domain-specific language does not contain this element;
16. **List Item:** The domain-specific language contains this element;

17. **Menu Item:** The domain-specific language does not contain this element;
18. **Section:** The domain-specific language does not contain this element;
19. **Slider:** The domain-specific language does not contains this element;
20. **Span:** The domain-specific language contains this element;
21. **Thumbnail:** The domain-specific language does not contain this element;
22. **Unordered List:** The domain-specific language contains this element;

The following list details the 22 properties that change the appearance of the elements that were identified from the answers given to the questionnaire.

1. **Change element text** The domain-specific language contains this property;
2. **Type:** The domain-specific language contains this property;
3. **Colour:** The domain-specific language contains this property;
4. **Display:** The domain-specific language does not contain this property;
5. **Width:** The domain-specific language contains this property;
6. **Background Colour:** The domain-specific language contains this property;
7. **Name on Navbar:** The domain-specific language contains this property;
8. **Text Align:** The domain-specific language contains this property;
9. **Links on Navbar:** The domain-specific language contains this property;
10. **Font Size:** The domain-specific language contains this property;
11. **Height:** The domain-specific language contains this property;
12. **Border Radius:** The domain-specific language does not contains this property;
13. **Href:** The domain-specific language does not contains this property;
14. **Round Image:** The domain-specific language does not contains this property;
15. **Align Items:** The domain-specific language does not contains this property;
16. **Flex:** The domain-specific language does not contains this property;
17. **Flex Direction:** The domain-specific language does not contains this property;
18. **Float:** The domain-specific language does not contains this property;
19. **Image on Navbar:** The domain-specific language contains this property;

20. **Margin:** The domain-specific language contains this property;
21. **Source:** The domain-specific language contains this property;

The following list contains the 17 functionalities that were identified from the answers given to the questionnaire.

1. **Add or Create elements:** The domain-specific language contains this functionality;
2. **Define where the elements are created:** The domain-specific language does not contain this functionality. This functionality allows specifying where the elements are created;
3. **Specify the value for the CSS property while creating an element:** The domain-specific language does not contain this functionality. This functionality allows detailing the value of the property while creating the element. In the domain-specific language, the user can specify the properties that he wants to change while creating the element;
4. **Several instructions on the same command:** The domain-specific language does not contain this functionality. It is only possible to execute one command at a time. However, the user can specify the properties that he wants to change while creating the element;
5. **Define id's and classes:** The domain-specific language does not contain this functionality;
6. **Define columns or set element side by side:** The domain-specific language contains an equivalent to this command which is "same row". The "same row" command permits the user to change the position of the children of an element to the same row;
7. **Bot asks if the user wants a certain element:** The domain-specific language does not contain this functionality;
8. **Create elements on the navbar:** The domain-specific language does not contain this functionality;
9. **Exit or Element is complete or Done:** The domain-specific language contains this functionality;
10. **Save Project:** The domain-specific language contains this functionality;
11. **Change the appearance of the elements in the navbar:** The domain-specific language does not contain this functionality;
12. **Invert position of two elements:** The domain-specific language does not contain this functionality;
13. **Actions that require JavaScript:** The domain-specific language does not contain this functionality;

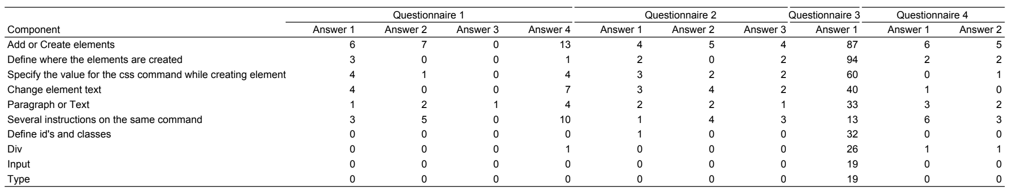
14. **Loading a template without removing the existing code:** The domain-specific language does not contain this functionality;

15. **Open Project:** The domain-specific language contains this functionality;

16. **Rows:** The domain-specific language contains this functionality. The user is able to change the position of the children of an element to the "same row" or to the "same column";

17. **Select a Element:** The domain-specific language contains this functionality.

Appendix D (p. 93) shows the number of times that each participant utilised each one of the components and in Figure 6.1 it is shown the ten most utilised components. Figure 6.2 (p. 55) shows a box and whisker plot of the occurrences of the components mentioned before.



	Questionnaire 1				Questionnaire 2			Questionnaire 3	Questionnaire 4	
Component	Answer 1	Answer 2	Answer 3	Answer 4	Answer 1	Answer 2	Answer 3	Answer 1	Answer 1	Answer 2
Add or Create elements	6	7	0	13	4	5	4	87	6	5
Define where the elements are created	3	0	0	1	2	0	2	94	2	2
Specify the value for the css command while creating element	4	1	0	4	3	2	2	60	0	1
Change element text	4	0	0	7	3	4	2	40	1	0
Paragraph or Text	1	2	1	4	2	2	1	33	3	2
Several instructions on the same command	3	5	0	10	1	4	3	13	6	3
Define id's and classes	0	0	0	0	1	0	0	32	0	0
Div	0	0	0	1	0	0	0	26	1	1
Input	0	0	0	0	0	0	0	19	0	0
Type	0	0	0	0	0	0	0	19	0	0

Figure 6.1: Ten most used components

As it can be seen, in the case of some components, the maximum number of uses by an user is much larger than the associated average number. Therefore, some of these cases can be outliers. To determine if they are outliers, it was used the Grubbs' test[1]. The following components have a significant outlier:

1. Paragraph or Text;

[1] https://www.graphpad.com/quickcalcs/grubbs2/

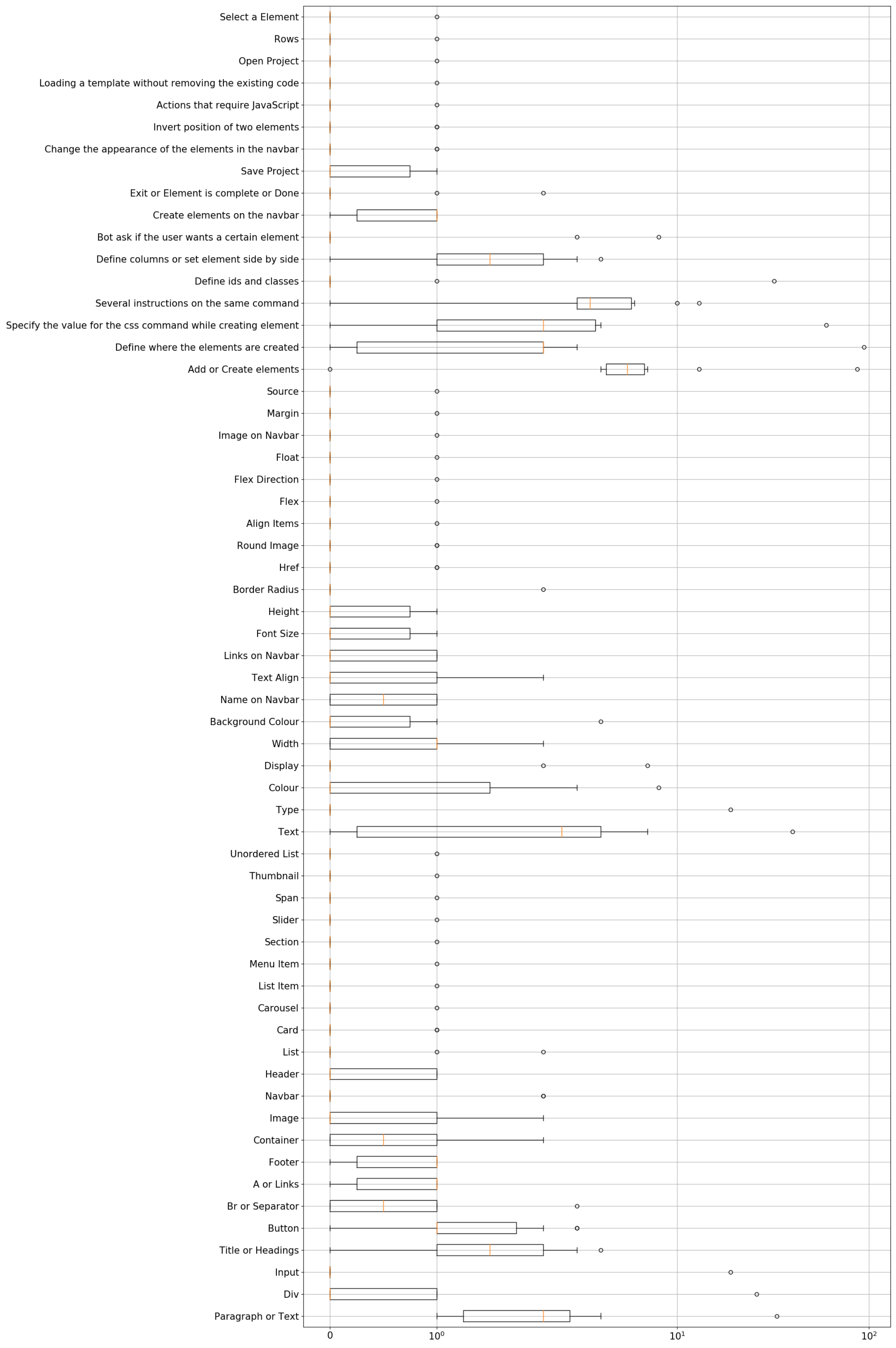


Figure 6.2: Box and Whisker plot for all the components

2. Div;
3. Br or Separator;
4. Input;
5. Change element text;
6. Type;
7. Colour;
8. Display;
9. Background Colour;
10. Border Radius;
11. Add or Create Elements;
12. Define where the elements are created;
13. Specify the value for the CSS command while creating an element;
14. Define id's and classes;
15. Bot asks if the user wants a certain element.

This suggests that in some cases, almost only one participant utilised these components. Therefore, in these cases, if these elements are eliminated, then the number of occurrences of those components get sharply reduced.

By examining Table 6.2 (p. 55), it can be seen that the most used elements can be created using the domain-specific language. It also can be verified that the most used CSS properties by the participants are present in the domain-specific language. The most used functionality, creating elements, is already included in the domain-specific language.

Almost all of the participants used commands that contain several instructions structured in a single command. However, the developed framework does not support this possibility. Some of the participants also specified the value for the CSS property while creating an element, but this is not supported by the tool. However, it is permitted to indicate what are the CSS properties to be changed while creating an element. There are also some commands that almost all the participants used that have a similar command in the tool such as the definition of columns or setting elements side by side or setting an element on the left and another on the right. The equivalent to any of these actions in the tool is the command "same row". This command modifies the position of the children of one element allowing the children of the element to be on the same row.

It must be noticed that none of the participants used commands to remove elements. On the other hand, only one of the participants defined a command to allow going back to another element as the "select" command in the tool.

6.1.2 Tool Validation

An experiment as created to evaluate the developed tool. The objective of this test was to determine if the domain-specific language was easy to understand. Other goals of this experiment were to determine if front end developers and back end developers would use a conversational interface to develop prototypes of webpages.

6.1.2.1 Experimental Parameters

In the first place, we developed this experiment and tested it with an user. This allowed identifying some problems and correct them. The following parameters were adopted:

Experiments: The experiment consisted of replicating two webpages. The guide for this experiment can be seen in Appendix E (p. 97)

Participants: The number of participants was eleven. One of the requirements was that they needed to have a background in computer science or design. Ten of the participants had a background in computer science, and the remaining one had a background in design. Due to the short amount of time available to get the feedback, the participants were mainly acquaintances or indicated/suggested by acquaintances.

Duration: There was a maximum duration of 60 minutes for the experience and a maximum duration of 20 minutes for each problem. These two timeouts were set to provide an appropriate time to conclude both problems and not to strain the participants.

Procedure: All the participants did the same experience. Before starting the experiences, the participants were required to do two tasks in order to learn how to use the tool. These two tasks focused on creating elements, changing their appearance, adding images, changing from one element to another and making copies. Afterwards, the participants needed to replicate two websites. During the experiment, it was provided to the participants a file with all the commands that can be executed in the tool as well as the elements that can be created and the properties that can be changed. This file can be seen in Appendix F (p. 103). Due to some limitations of the tool, the expected results of the two problems differ slightly regarding the images shown to replicate. The expected result of each problem can be seen in Appendix G (p. 109).

Environment: All of the experiments were executed in a remote environment. A video call software was utilised to explain the experiment to the participants and to observe and record the actions they executed in the tool. Using this method enabled the participants to ask questions about the problems.

Data: The time that each participant took was recorded as well as the actions that they executed in the tool.

Post-test: After completing the two problems, the users need to fill a questionnaire about the experiment. This questionnaire can be seen in Appendix H (p. 113).

6.1.2.2 Tasks

As none of the participants used the tool before, it was developed a tutorial to introduce them to the more common commands. This tutorial included two tasks:

Task 1: Copying and pasting the commands that were written in the guide in order to replicate an image. This task objective is to show how the tool works, how to create elements and how to change their appearance.

Task 2: The participants needed to upload two images and change the image on the navbar and the source of the image on the body of the webpage. This task main goals were to change the element that the user is currently on using the “select” command and show how to upload images to the webpage. Another goal was to familiarise the participants with the “help” command and with the file that was given to them. Figure 6.3 shows the result of this task.

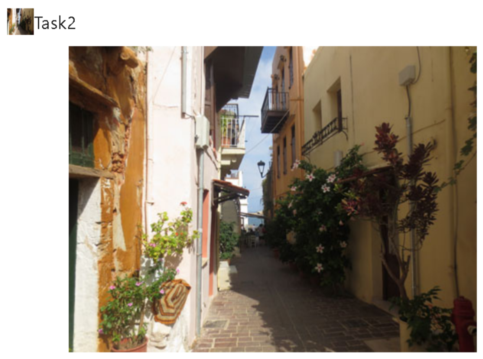


Figure 6.3: Webpage asked to upload in the second task of the tutorial

After completing this tutorial, the participants needed to replicate two webpages. Appendix G (p. 109) shows the two webpages that were expected to be created. The next paragrahs provide a brief explanation of what was required to do in each of the problems:

Problem 1: The participants need to create a navbar and change their name and links. Afterwards, it was required to create a container with one heading, one paragraph and eight buttons. The participants could create one button and then make seven copies of this button. Afterwards, it was needed to create a container with a button and a paragraph. The participants needed to change the property “text align” to centre the elements inside both containers.

Problem 2: First, the participants were required to create a div. Then, they needed to create another div, inside of the previous one, with an image, a paragraph and one more div. In this last div, they needed to add two buttons and a paragraph and use the property “same row” to align the elements in the same row. Afterwards, they needed to create two copies of the div that contains the image. Next, it was required to use the property “same row” in the first div created and make two copies of this div. Then, it was needed to create a footer with two div’s and use the property “same row”. Finally, the left side div on the footer needed to have two paragraphs and the right side div needed to have one paragraph.

6.1.2.3 Results

By analysing the eleven recordings of the participants that interacted with the tool, it was possible to determine the time and the number of actions that the participants required to complete each of the two problems. These results can be seen in the Appendix I (p. 117). In this appendix, it is also shown how many commands of each type the participants used to complete each of the problems. Afterwards, it was calculated the average time and the average number of commands that were required to complete each of the two problems. These average values are shown in Table 6.1.

Mean	**Problem 1**	**Problem 2**
Time (min:ss)	11:10	12:14
Number of Commands	35.63	53.81
Number of Add Commands	10.18	16.45
Number of Remove Commands	0.72	0.45
Number of Change Commands	12.27	13.63
Number of Undo Commands	2.09	3.36
Number of Redo Commands	0.00	0.27
Number of Select Commands	0.81	0.45
Number of Make Commands	1.27	2.81
Number of Enter Commands	0.27	0.54
Number of Exit Commands	6.81	13.9
Number of Cancel Commands	0.18	0.27
Number of Help Commands	1.00	1.45
Number of Clear Commands	0.00	0.18

Table 6.1: Mean time and mean number of commands taken to complete each problem

As expected the “add”, “change” and “exit” commands where the most utilised by the participants. This is because the elements of the webpage must be created, and their appearance can be changed. To move in the hierarchy, the most common command is “exit” because this element allows the users to go from an element to its parent. After executing a “exit” command, if a new element is created, this new element is positioned before the element regarding which the user executed the “exit” command. This interaction permits creating a webpage from top to bottom. Users utilised the commands “remove” and “undo” do delete elements. The “undo” command could be used to remove elements if the last action was the creation of an element. Two users restarted the second problem from scratch using the command “clear”. The “open”, “save”, “upload”, “upload image”, “download”, “hide borders” and “show borders” do not appear in Table 6.1 (p. 59) because it was not required to use any of this commands to replicate the webpages.

The answers to the questions in the questionnaire about the experience are given in Figure 6.4 (p. 62).

By analysing these graphs, it can be concluded that most of the participants that developed front end indicated that they would prefer using another tool. One reason that may explain this type of answer is that it is more flexible in developing a prototype by writing code than using this tool. This occurs because the participants are not allowed to create all HTML elements and change all the CSS properties. Furthermore, the participants are certainly used to the way in which they usually develop prototypes and might not be interested in learning a new tool.

In these graphics it is also clear that the participants that do not develop front end answered that they would utilise this tool to develop a prototype. This preference can be related to the fact that they are not used to develop web prototypes, and this tool gives them an easy and quick way to develop them.

The participants also answered that the language was not hard to understand, which was one of the main objectives of this work. Furthermore, they indicated that there was some conversational degree provided by the tool (no answer below sometimes) which is a good result because this tool was supposed to correspond to a conversational interface. Finally, the users said that the chatbot was giving them useful information about how the tool works and about possible commands that can be executed.

6.2 Validation Threats

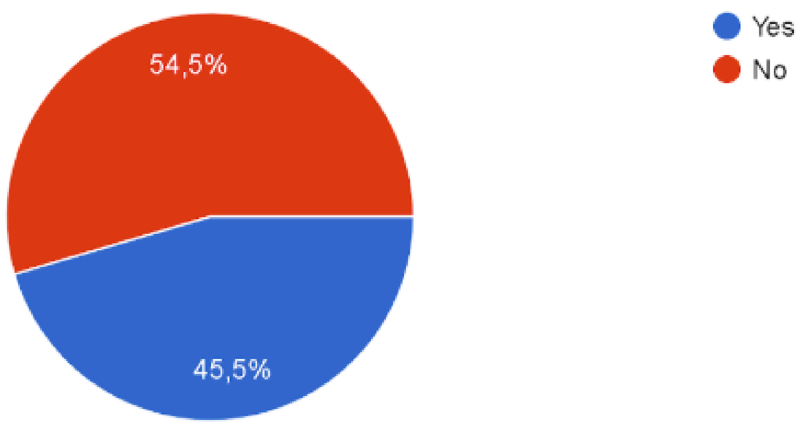
To test a tool that aims at being used by persons, the best way to validate it is to do an experiment. However, this method has some flaws. Section 6.2.1 details the threats that jeopardise the internal validity and Section 6.2.2 describes the threats that jeopardise the external validity. Section 6.2.3 describes the threats that jeopardise the construct validity.

6.2.1 Internal Validity

Threats to internal validity can influence the independent variable with respect to causality. Therefore, the conclusion about a possible relationship between a treatment and a result can be affected

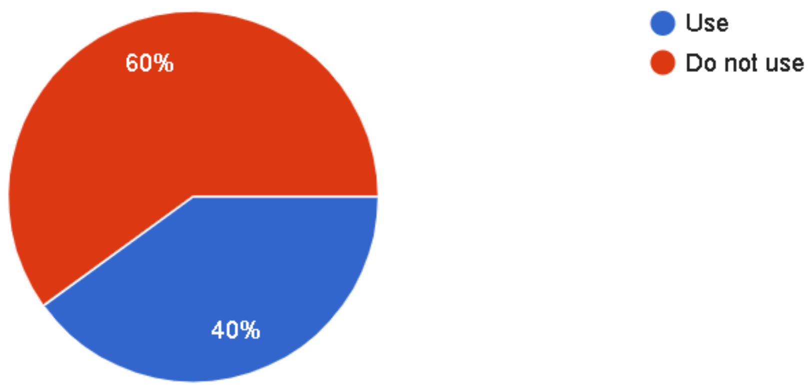
Do you normally develop front end?

11 respostas



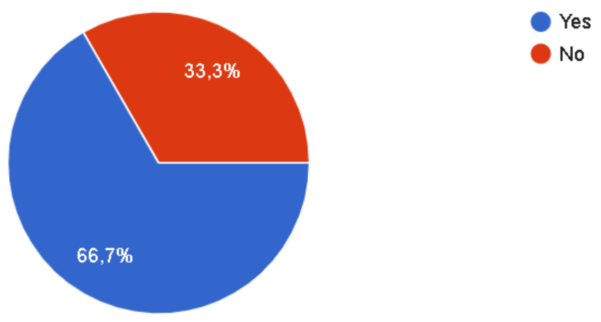
If yes, would you prefer to use this tool or to code?

5 responses



If you answered "No" to the first question, would you developed a prototype using this tool?

6 respostas



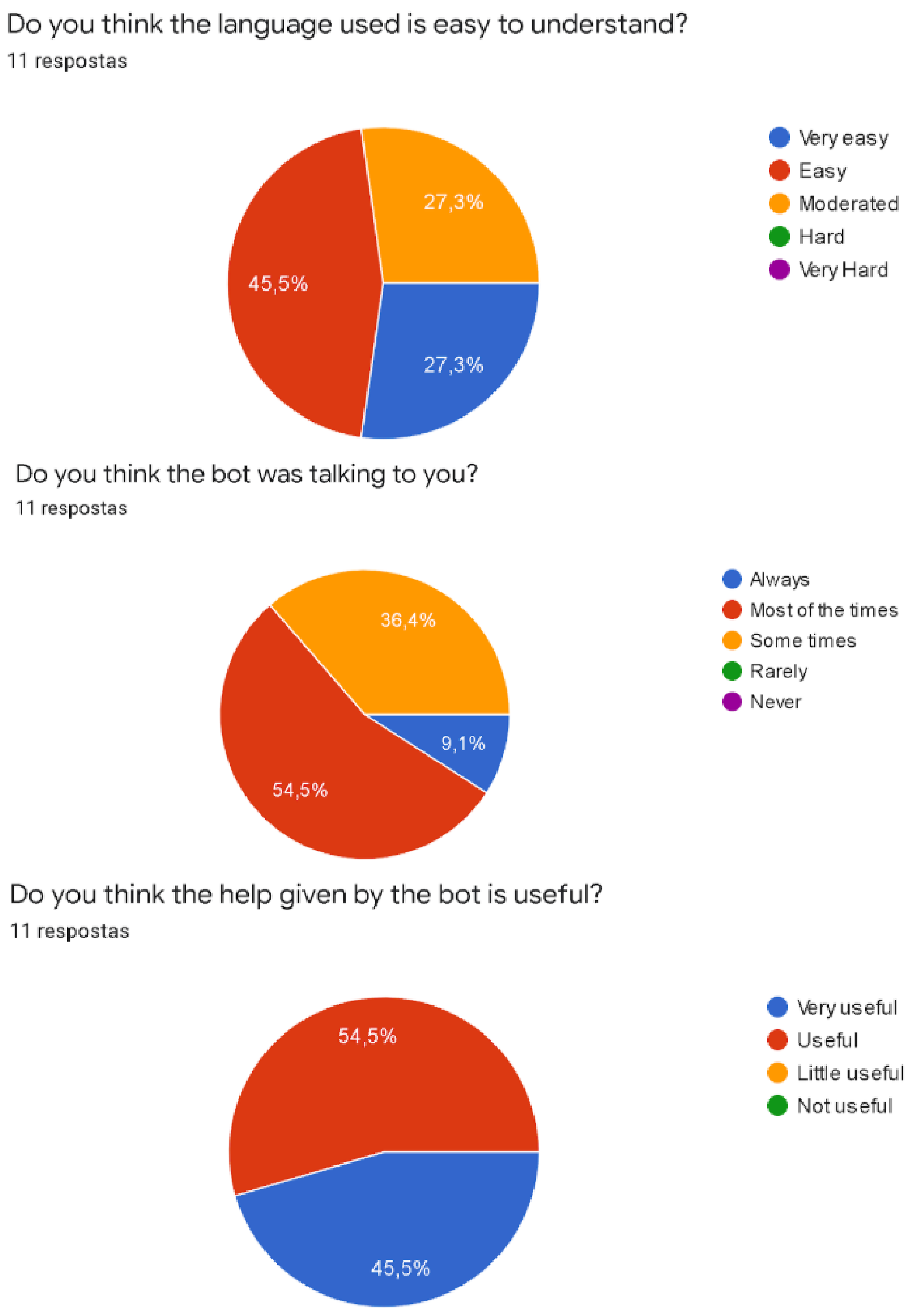


Figure 6.4: Results of the questionnaire about the experience

[WRH+12].

Environment: All the experiments were done in a different environment because they were executed in a remote way. This can mean that the time the participants took to complete the problems could be influenced by distractions. However, this threat was mitigated by recording the user actions during the experiment. Future experiments should try to control these variables in a more rigorous manner;

6.2.2 External Validity

External Validity concerns the degree to which the findings of a study can be generalised to different populations or settings.

Sample Characteristics: The participants may not have the skill set required to use this tool because it is required some knowledge on webpage development. However, 10 of the participants had a background in computer science, and the remaining participant had a background in webpage design. The participants were students at the Faculty of Engineering of the University of Porto. Future experiments should try to increase the number of participants and have a higher degree of heterogeneity in terms of the background of the participants;

1. **Sample Size:** The number of participants may not be sufficient to obtain a representative set of results. The size of the sample was rather small, something that must be taken into account when interpreting the findings of this study. In the future, this tool should be tested using a higher number of participants;
2. **Complexity of the Problems:** The complexity of the two problems may not be adequate. However, taking into consideration the average time and the average number of commands taken to complete each of the problems, more difficult tasks can be time-consuming for the participants;
3. **Miscomprehension of the Problems:** In this type of questionnaires, it is possible that the participants can misunderstand what is asked to do in written instructions. In this case, this issue was addressed, allowing participants to ask questions about the problems before starting solving them.

6.2.3 Construct Validity

Construct Validity is related with the suitability of the conclusions made on the observations or measurements, primarily if a test measures the intended construct.

1. **Researcher expectations:** The evaluator and creator of this experiment were the same person. Therefore, the participants could be influenced on how to reach the final result. In the future, an experiment should be done in which the evaluator of the experiment is different from the creator.

6.3 Conclusions

This section detailed the validation processes used to evaluate the domain-specific language and the conversational interface. The results obtained from the questionnaire to validate the domain-specific language and from the experiment that was made were also presented. It was concluded that the domain-specific language contains the most used elements that were utilised by the participants as well as the most common CSS properties. It was also verified that some of the commands used by the participants were specified in the domain-specific language. The participants of the experiment felt that the language used was not hard to understand. Finally, the participants typically not involved in front end tasks were more prone to use a tool like this one in the future.

Chapter 7

Conclusions and Future Work

During this work, it was explained that one possible way to developed webpages is starting by drawing a hand-drawn sketch of the webpage and then transforming it into a prototype. This prototype is evaluated by the client, and if one of the requirements is missing, the whole process must be repeated. The repetition of this process means that it is consumed more time and resources than it was initially planned. Therefore, it was developed a tool that allows the designers to create the prototype immediately, aiming at easing the process of creating prototypes.

In line with this global objective, in this work it was first explained the background concepts of a conversational interface, live software development and web development. Afterwards, it was described the current state of the art regarding the interaction in the computer-aided design software, the conversational interface and works that transform hand-drawn mockups to prototypes. Later, it was defined the problem as well as the adopted solution and what this work aimed at accomplishing. Following, the implementation details of the solution described previously were detailed.

Section 7.1 describes the main difficulties faced during the development of this thesis. Section 7.2 enumerates the main contributions of this work. Section 7.3 answers the proposed research questions as well as the hypothesis. Finally, Section 7.4 introduces some improvements that could be done in the future.

7.1 Main Difficulties

There were several difficulties while researching and developing this tool. The questionnaire that at the end was used to validate the domain-specific language was, in fact, intended to be used to create the domain-specific language itself. The initial objective was to obtain a number of answers from designers so that, based on them, the domain-specific language could be created. However, it was only received one response until May thus turning it impossible to use it as initially intended. Therefore, the domain-specific language was mainly developed by associating common actions that programmers use to commands that can be executed in the tool, and it was finally subjected to validation.

In addition, it was also challenging to develop a framework that would enable someone to improve it in a natural way in the future. Another difficulty was to make the conversation non-repetitive.

7.2 Main Contributions

The main contributions of this work are the specification of a domain-specific language and a conversational interface that combined allow the generation of webpages. The most common interactions that developers use to write code were identified as well as the most used elements. Then, these two components were added to the domain-specific language. Afterwards, some commands that can help creating webpages were included. Later, it was developed a conversational interface using this domain-specific language. The developed tool includes functions designed for each one of the commands present in the domain-specific language.

7.3 Summary Research Questions and Hypothesis

After detailing how the domain-specific language and the tool were developed as well as the results that were obtained during their validation, the hypothesis and research questions detailed in (*cf.* Chapter 4, p. 27) can be answered.

RQ1: *Are users able to create a complex webpage using a conversational interface?*

The webpages that were requested to replicate in the experiment were two complex webpages because both of them required to create elements inside other elements and to modify the appearance of some elements. Given that the users were able to replicate both of the webpages successfully, it can be said that a conversational interface can be used to create complex websites.

RQ2: *Is it possible to specify an easy to understand domain-specific language aiming at generating webpages?*

One of the questions asked to the participants after completing the experiment was: "Do you think the language used is easy to understand?". As the results of this question were at least Moderate, it can be concluded that the domain-specific language is not hard to comprehend.

RQ3: *Would front end developers use a conversational interface to develop webpages?*

The majority of the participants who said they usually developed front end answered that they would not use this tool to develop prototypes. One of the reasons for this is that they may be used to the frameworks in which they develop prototypes and do not want to move to a new framework.

RQ4: *Would back end developers use a conversational interface to develop webpages?*

The majority of the participants who said they usually do not develop front end answered that they would use this tool to develop prototypes, if they needed. One of the reason can be because they rarely develop front end, and this tool gives them a faster and easier way to develop prototypes.

Hypothesis: *"A conversational interface that allows the generation of complex webpages using an easy to understand domain-specific language is more comfortable to utilise than the current forms of developing a prototype."*

One of the objectives of the experience used to validate the tool was to verify if it was possible to generate complex websites with a conversational interface. The participants successfully replicated the webpages of both problems. They also answered in the questionnaire of the experience that the domain-specific language was not difficult to understand. The majority of participants that do not develop front end answered that they would use this tool to develop a prototype. This can mean that it is easier to use the tool than develop a prototype using the most common forms to develop them. However, most participants that develop front end indicated that they would not use the tool. One reason for this answer can be due to the current limitations of the tool. Another reason can be because they are used to the way they develop prototypes and do not want to move to a new tool from scratch.

7.4 Future Work

In the future, more elements can be added to the domain-specific language as well as more properties to change the appearance of the elements. Furthermore, some functionalities that were mentioned by some participants in the validation process can be added in the domain-specific language, such as enabling to specify the value of the property that the user wants to change while creating the element. Another two possible functionalities to add is to create several elements in the same command or to specify the number of copies while creating the element.

It is also possible to improve the help provided by the chatbot by including a command that enables the user to see all the available commands, the properties that were changed in the current element and the hierarchy of the developed webpage. It is also possible to improve how it is seen the element that the user is currently on and how the messages appear in the conversational panel.

Another aspect that can be improved is the use of default values as it was done in the margin and padding properties. Some examples are to add a default text if the user did not say he wanted a text when creating a heading or to having a default colour for the properties colour and background colour.

Finally, the experiment that was conducted to validate the tool can be and should be repeated but with a higher number of participants, a higher degree of heterogeneity in terms of participant background and in an environment where the experiment variables can be controlled in a more rigorous manner in order to have more representative feedback. This would ultimately enable improving the developed tool turning it more useful to the developers.

References

[AKW15] Sameera A Abdul-Kader and JC Woods. Survey on chatbot design techniques in speech conversation systems. *International Journal of Advanced Computer Science and Applications*, 6(7), 2015. Cited on page 15.

[AMG00] Stephen C Arnold, Leo Mark, and John Goldthwaite. Programming by voice, vocal-programming. In *Proceedings of the fourth international ACM conference on Assistive technologies*, pages 149–155, 2000. Cited on pages 24 and 25.

[ARC$^+$19] Ademar Aguiar, André Restivo, Filipe Figueiredo Correia, Hugo Sereno Ferreira, and João Pedro Dias. Live software development: tightening the feedback loops. In *Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming*, pages 1–6, 2019. Cited on page 9.

[Beg04] Andrew Begel. Spoken language support for software development. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, pages 271–272. IEEE, 2004. Cited on page 25.

[Bel18] Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 1–6, 2018. Cited on pages 19, 21, and 27.

[BFdH$^+$13] Sebastian Burckhardt, Manuel Fahndrich, Peli de Halleux, Sean McDirmid, Michal Moskal, Nikolai Tillmann, and Jun Kato. It's alive! continuous feedback in ui programming. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pages 95–104, 2013. Cited on page 9.

[BLC90] Timothy J Berners-Lee and Robert Cailliau. Worldwideweb: Proposal for a hypertext project. 1990. Cited on page 10.

[BS17] Stefan Bieliauskas and Andreas Schreiber. A conversational user interface for software visualization. In *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 139–143. IEEE, 2017. Cited on page 14.

[Cho03] Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003. Cited on page 7.

[CK12] Anthony E Cozzie and Samuel King. Macho: Writing programs with natural language and examples. 2012. Cited on page 25.

[dSF19] João Carlos da Silva Ferreira. Live web prototypes from hand-drawn mockups. 2019. Cited on pages 19, 24, and 27.

[Fel99] Susan Feldman. Nlp meets the jabberwocky: Natural language processing in information retrieval. *ONLINE-WESTON THEN WILTON-*, 23:62–73, 1999. Cited on page 7.

[JAB$^+$19] Vanita Jain, Piyush Agrawal, Subham Banga, Rishabh Kapoor, and Shashwat Gulyani. Sketch2code: Transformation of sketches to ui in real-time using deep neural network. *arXiv preprint arXiv:1910.08930*, 2019. Cited on pages 19, 20, and 27.

[KDMB17] Lorenz Cuno Klopfenstein, Saverio Delpriori, Silvia Malatini, and Alessandro Bogliolo. The rise of bots: A survey of conversational interfaces, patterns, and paradigms. In *Proceedings of the 2017 conference on designing interactive systems*, pages 555–565, 2017. Cited on page 16.

[LDF20] André Sousa Lago, João Pedro Dias, and Hugo Sereno Ferreira. Conversational interface for managing non-trivial internet-of-things systems. In *International Conference on Computational Science*, pages 384–397. Springer, 2020. Cited on page 16.

[LDG$^+$17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. Cited on page 20.

[Lid98] Elizabeth D Liddy. Enhanced text retrieval using natural language processing. *Bulletin of the American Society for Information Science and Technology*, 24(4):14–16, 1998. Cited on page 7.

[PRZH00] David Price, Ellen Rilofff, Joseph Zachary, and Brandon Harvey. Naturaljava: a natural language interface for programming in java. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 207–211, 2000. Cited on page 25.

[RAMI17] AM Rahman, Abdullah Al Mamun, and Alma Islam. Programming challenges of chatbot: Current and future prospective. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 75–78. IEEE, 2017. Cited on pages 16 and 25.

[RCHB18] Lucas Rosenblatt, Patrick Carrington, Kotaro Hara, and Jeffrey P Bigham. Vocal programming for people with upper-body motor impairments. In *Proceedings of the Internet of Accessible Things*, pages 1–10. 2018. Cited on pages 24 and 25.

[REBIM10] MZ Rashad, Hazem M El-Bakry, Islam R Isma'il, and Nikos Mastorakis. An overview of text-to-speech synthesis techniques. *Latest trends on communications and information technology*, pages 84–89, 2010. Cited on page 9.

[Rob19] Alex Robinson. Sketch2code: Generating a website from a paper mockup. *arXiv preprint arXiv:1905.13750*, 2019. Cited on pages 19, 22, and 27.

[SA07] Bayan Abu Shawar and Eric Atwell. Different measurement metrics to evaluate a chatbot system. In *Proceedings of the workshop on bridging the gap: Academic and industrial research in dialog technologies*, pages 89–96, 2007. Cited on page 16.

[Tan13] Steven L Tanimoto. A perspective on the evolution of live programming. In *2013 1st International Workshop on Live Programming (LIVE)*, pages 31–34. IEEE, 2013. Cited on page 9.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012. Cited on page 63.

[YD16] Dong Yu and Li Deng. *AUTOMATIC SPEECH RECOGNITION.* Springer, 2016. Cited on page 5.

Appendix A

Domain-Specific Language Configuration file

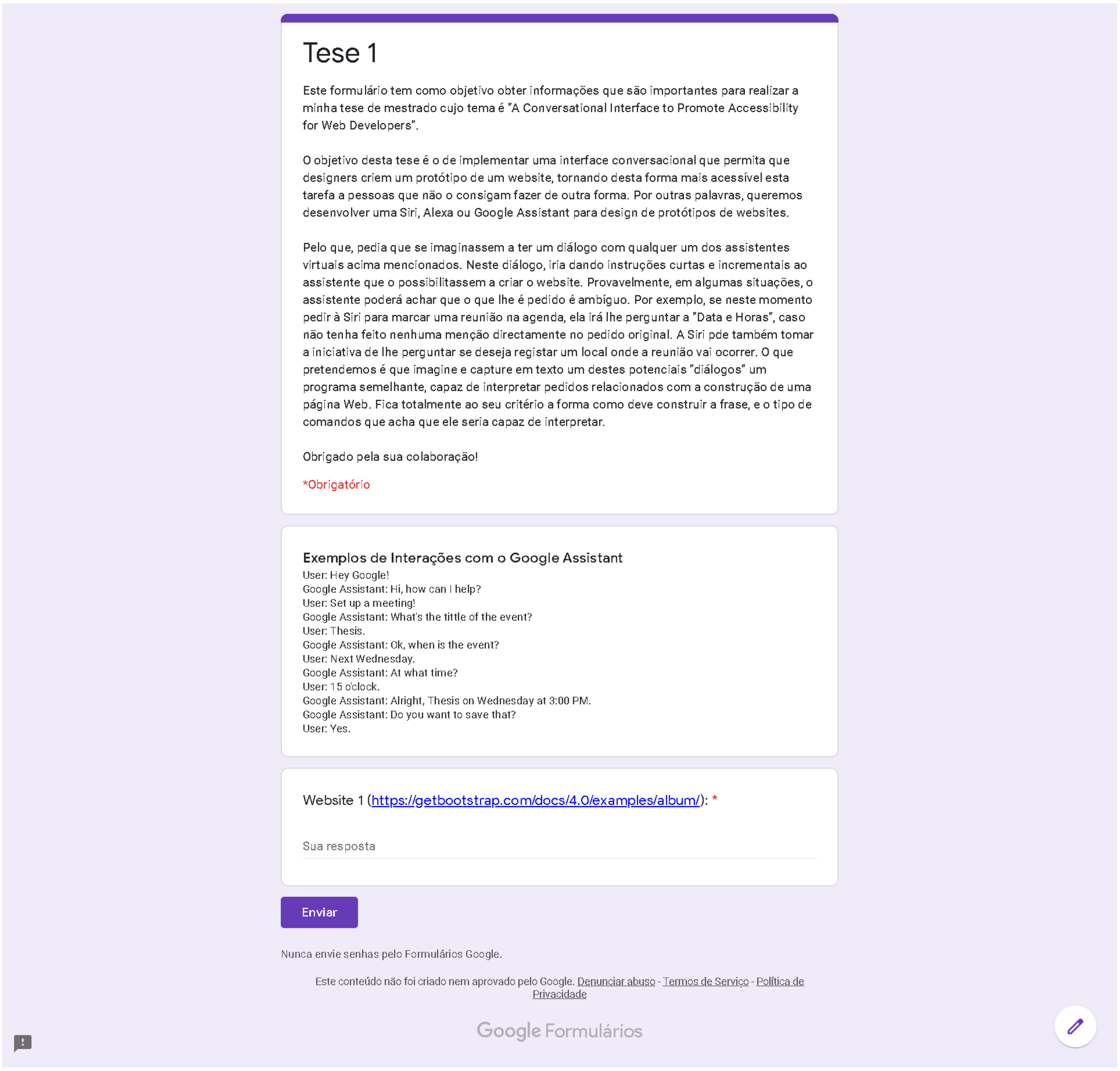
This appendix includes the file where the domain-specific language is detailed.

```
{
    "languages": ["en"],
    "entities": [
        {
            "name": "element",
            "language": "en",
            "values": [
              {"text": "h1", "options": ["H1"]},
              {"text": "h2", "options": ["H2"]},
              {"text": "h3", "options": ["H3"]},
              {"text": "h4", "options": ["H4"]},
              {"text": "h5", "options": ["H5"]},
              {"text": "h6", "options": ["H6"]},
              {"text": "navbar", "options": ["Navbar"]},
              {"text": "button", "options": ["Button"]},
              {"text": "div", "options": ["div"]},
              {"text": "container", "options": ["Container"]},
              {"text": "image", "options": ["Image"]},
              {"text": "input", "options": ["Input"]},
              {"text": "p", "options": ["P"]},
              {"text": "unordered list", "options": ["Unordered List"]},
              {"text": "ordered list", "options": ["Ordered List"]},
              {"text": "list item", "options": ["List Item"]},
              {"text": "label", "options": ["Label"]},
              {"text": "span", "options": ["Span"]},
              {"text": "footer", "options": ["Footer"]},
              {"text": "form", "options": ["Form"]}
            ]
        },
        {
            "name": "attributeL",
            "language": "en",
            "values": [
              {"text": "links", "options": ["Links"]}
            ]
        },
        {
            "name": "attributeT",
            "language": "en",
            "values": [
              {"text": "name", "options": ["Name"]},
              {"text": "border", "options": ["Border"]},
              {"text": "border style", "options": ["Border Style"]},
              {"text": "type", "options": ["Type"]},
              {"text": "text", "options": ["Text"]},
              {"text": "text align", "options": ["Text Align"]}
            ]
        },
        {
            "name": "attributeU",
            "language": "en",
            "values": [
              {"text": "logo", "options": ["Logo"]},
              {"text": "source", "options": ["Source"]}
            ]
        },
        {
            "name": "attributeV",
            "language": "en",
            "values": [
              {"text": "margin", "options": ["Margin"]},
              {"text": "padding", "options": ["Padding"]},
              {"text": "height", "options": ["Height"]},
              {"text": "width", "options": ["Width"]},
              {"text": "border width", "options": ["Border Width"]},
              {"text": "font size", "options": ["Font Size"]}
            ]
        },
        {
            "name": "attributeH",
            "language": "en",
            "values": [
              {"text": "color", "options": ["Colour", "colour", "Color"]},
              {"text": "background color", "options": ["Background Color", "background colour", "Background colour"]},
              {"text": "border color", "options": ["Border Color", "border colour", "Border Colour"]}
            ]
        },
        {
            "name": "attribute",
            "language": "en",
            "values": [
              {"text": "same row", "options": ["Same Row"]},
              {"text": "same column", "options": ["Same Column"]}
            ]
        },
        {
            "name": "value",
            "language": "en",
            "regex": "((((([0-9])+(\\.[0-9]+)?((r?em)|(px))))(\\s((([0-9])+(\\.[0-9]+)?((r?em)|(px))))){0,3})|^0\\%$|^100\\%$|^[0-9]{1,2}\\%$|^[0-9]{1,2}\\.[0-9]{1,3}\\%$"
        }
    ],
    "intents": [
        {
            "language": "en",
            "intent": "add",
            "commands": ["add %element%", "create %element%", "append %element%"],
            "answers": ["Adding {{ element }}", "Creating {{ element
                }}"],
            "questions": [
              {
                "entity": "element",
                "question": {"en": "What element do you want to add
                   ?"}
              }
            ]
        },
        {
            "language": "en",
            "intent": "select",
            "commands": ["select %element%", "pick %element%"],
            "answers": ["Selecting {{ element }} {{ ordinal }}", "
               Picking {{ element }} {{ ordinal }}"],
            "questions": [
              {
                "entity": "element",
                "question": {"en": "What element do you want to
                   select?"}
              }
            ]
        },
        {
            "language": "en",
            "intent": "enter",
            "commands": ["enter", "go in", "get in"],
            "answers": ["Entering"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "exit",
            "commands": ["exit", "go out", "get out"],
            "answers": ["Exiting"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "attributeT",
            "commands": ["%attributeT% %text%"],
            "answers": ["Changing {{ attributeT }} to {{ text }}", "Editing {{ attributeT }} to {{ text }}"],
            "questions": [
              {
                "entity": "text",
                "question": {"en": "What is the text of {{ attributeT }}?"}
              }
            ]
        },
        {
            "language": "en",
            "intent": "attributeV",
            "commands": ["%attributeV% %value%"],
            "answers": ["Changing {{ attributeV }} to {{ value }}", "Editing {{ attributeV }} to {{ value }}"],
            "questions": [
              {
                "entity": "value",
                "question": {"en": "What is the value of {{ attributeV }}?"}
              }
            ]
        },
        {
            "language": "en",
            "intent": "attributeU",
            "commands": ["%attributeU% %url%"],
            "answers": ["Changing {{ attributeU }} to {{ url }}", "Editing {{ attributeU }} to {{ url }}"],
            "questions": [
              {
                "entity": "url",
                "question": {"en": "What is the url of {{ attributeU }}?"}
              }
            ]
        },
        {
            "language": "en",
            "intent": "attributeH",
            "commands": ["%attributeH% %hashtag%"],
            "answers": ["Changing {{ attributeH }} to {{ hashtag }}", "Editing {{ attributeH }} to {{ hashtag }}"],
            "questions": [
              {
                "entity": "hashtag",
                "question": {"en": "What is the hexadecimal of {{ attributeH }}?"}
              }
            ]
        },
        {
            "language": "en",
            "intent": "attributeL",
            "commands": ["%attributeL% %number%"],
            "answers": ["Changing {{ attributeL }} to {{ number }}", "Editing {{ attributeL }} to {{ number }}"],
            "questions": [
              {
                "entity": "number",
                "question": {"en": "How many {{ attributeL }}?"}
              }
            ]
        },
        {
            "language": "en",
            "intent": "attribute",
            "commands": ["%attribute%"],
            "answers": ["Changing {{ attribute }}", "Editing {{ attribute }}"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "make",
            "commands": ["make %number%"],
            "answers": ["Making {{ number }} copies"],
        "questions": [
          {
            "entity": "number",
            "question": {"en": "How many copies?"}
          }
        ]
    },
    {
        "language": "en",
        "intent": "download",
        "commands": ["download %text%"],
        "answers": ["Downloading files as {{ text }}", "Transfering files as {{ text }}"],
        "questions": [
          {
            "entity": "text",
            "question": {"en": "What name to give to the files?"}
          }
        ]
    },
    {
        "language": "en",
        "intent": "save",
        "commands": ["save %text%"],
        "answers": ["Saving {{ text }}"],
        "questions": [
          {
            "entity": "text",
            "question": {"en": "What name to give to the project?"}
          }
        ]
    },
    {
        "language": "en",
        "intent": "open",
        "commands": ["open %text%"],
        "answers": ["Opening {{ text }}"],
        "questions": [
              {
                "entity": "text",
                "question": {"en": "What project do you want to open
                   ?"}
              }
            ]
        },
        {
            "language": "en",
            "intent": "showBorders",
            "commands": ["show borders"],
            "answers": ["Showing Borders"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "hideBorders",
            "commands": ["hide borders"],
            "answers": ["Hiding Borders"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "undo",
            "commands": ["undo"],
            "answers": ["Undo"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "redo",
            "commands": ["redo"],
            "answers": ["Redo"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "clear",
            "commands": ["clear"],
            "answers": ["Clearing workspace"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "upload",
            "commands": ["upload"],
            "answers": ["Uploading files"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "uploadI",
            "commands": ["upload image"],
            "answers": ["UploadingI"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "remove",
            "commands": ["remove", "delete"],
            "answers": ["Removing element", "Deleting element"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "cancel",
            "commands": ["cancel"],
            "answers": ["Cancelling operation"],
            "questions": []
        },
        {
            "language": "en",
            "intent": "help",
            "commands": ["help"],
            "answers": ["Help"],
            "questions": []
        }
    ]
}
```

Appendix B

Domain-Specific Language Questionnaire

The objective of this questionnaire is to ask participants to identify and write commands that can be used to create a webpage. Therefore, four surveys were created, each of them having a different link to the webpage to be analysed. This appendix contains one of the four questionnaires.



Tese 1

Este formulário tem como objetivo obter informações que são importantes para realizar a minha tese de mestrado cujo tema é "A Conversational Interface to Promote Accessibility for Web Developers".

O objetivo desta tese é o de implementar uma interface conversacional que permita que designers criem um protótipo de um website, tornando desta forma mais acessível esta tarefa a pessoas que não o consigam fazer de outra forma. Por outras palavras, queremos desenvolver uma Siri, Alexa ou Google Assistant para design de protótipos de websites.

Pelo que, pedia que se imaginassem a ter um diálogo com qualquer um dos assistentes virtuais acima mencionados. Neste diálogo, iria dando instruções curtas e incrementais ao assistente que o possibilitassem a criar o website. Provavelmente, em algumas situações, o assistente poderá achar que o que lhe é pedido é ambíguo. Por exemplo, se neste momento pedir à Siri para marcar uma reunião na agenda, ela irá lhe perguntar a "Data e Horas", caso não tenha feito nenhuma menção directamente no pedido original. A Siri pde também tomar a iniciativa de lhe perguntar se deseja registar um local onde a reunião vai ocorrer. O que pretendemos é que imagine e capture em texto um destes potenciais "diálogos" um programa semelhante, capaz de interpretar pedidos relacionados com a construção de uma página Web. Fica totalmente ao seu critério a forma como deve construir a frase, e o tipo de comandos que acha que ele seria capaz de interpretar.

Obrigado pela sua colaboração!

*Obrigatório

Exemplos de Interações com o Google Assistant

User: Hey Google!
Google Assistant: Hi, how can I help?
User: Set up a meeting!
Google Assistant: What's the tittle of the event?
User: Thesis.
Google Assistant: Ok, when is the event?
User: Next Wednesday.
Google Assistant: At what time?
User: 15 o'clock.
Google Assistant: Alright, Thesis on Wednesday at 3:00 PM.
Google Assistant: Do you want to save that?
User: Yes.

Website 1 (https://getbootstrap.com/docs/4.0/examples/album/): *

Sua resposta

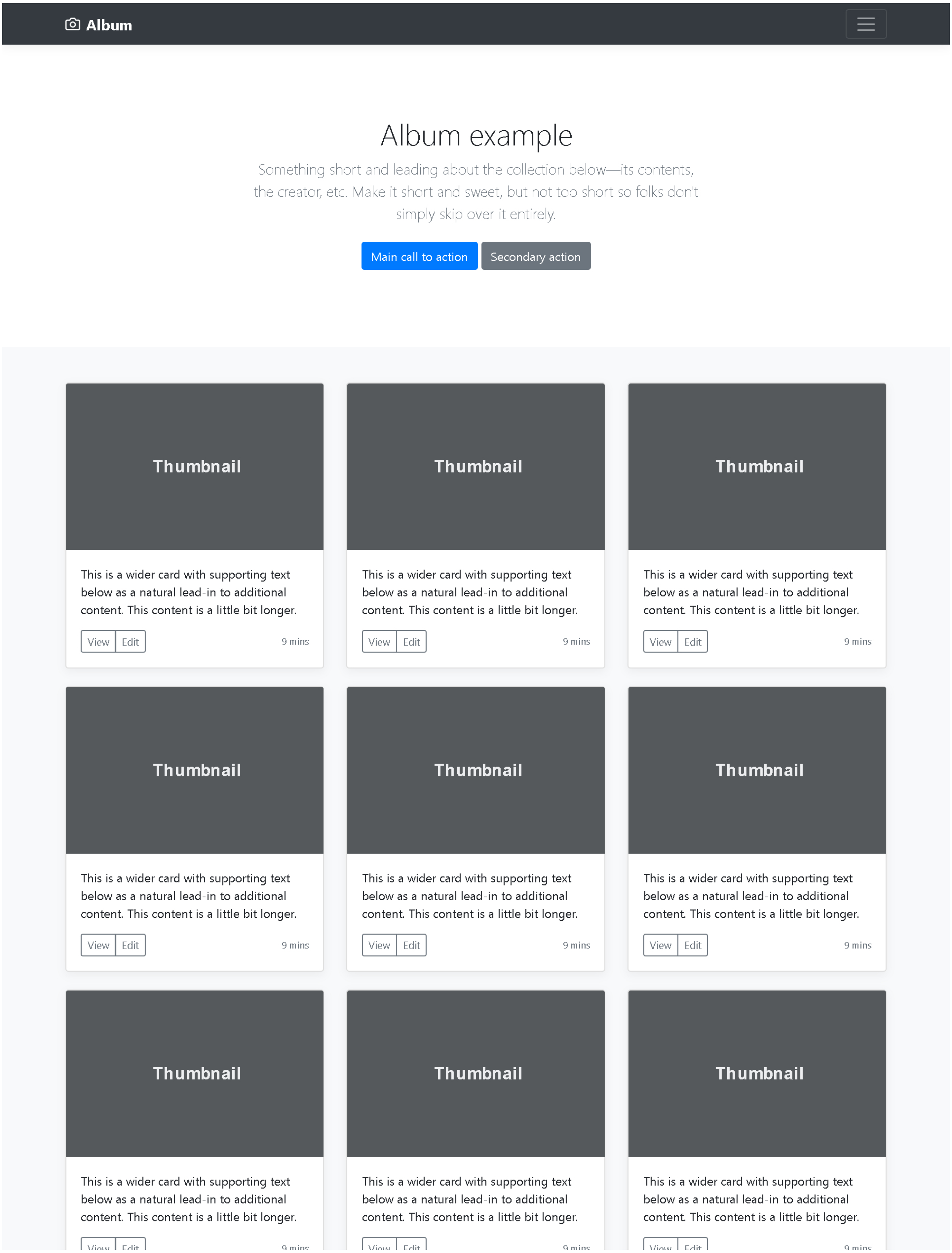
Enviar

Nunca envie senhas pelo Formulários Google.

Appendix C

Webpages to Replicate in the Questionnaires

Following is shown the four webpages that were requested to replicate in the questionnaire to validate the domain-specific language.



Album example

Something short and leading about the collection below—its contents, the creator, etc. Make it short and sweet, but not too short so folks don't simply skip over it entirely.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View Edit 9 mins

View Edit 9 mins

View Edit 9 mins

Album example is © Bootstrap, but please download and customize it for yourself!

New to Bootstrap? Visit the homepage or read our getting started guide.

Back to top

Company name

Features Enterprise Support Pricing Sign up

Pricing

Quickly build an effective pricing table for your potential customers with this Bootstrap example. It's built with default Bootstrap components and utilities with little customization.

Free

$0 / mo

10 users included
2 GB of storage
Email support
Help center access

Sign up for free

Pro

$15 / mo

20 users included
10 GB of storage
Priority email support
Help center access

Get started

Enterprise

$29 / mo

30 users included
15 GB of storage
Phone and email support
Help center access

Contact us

© 2017-2018

Features

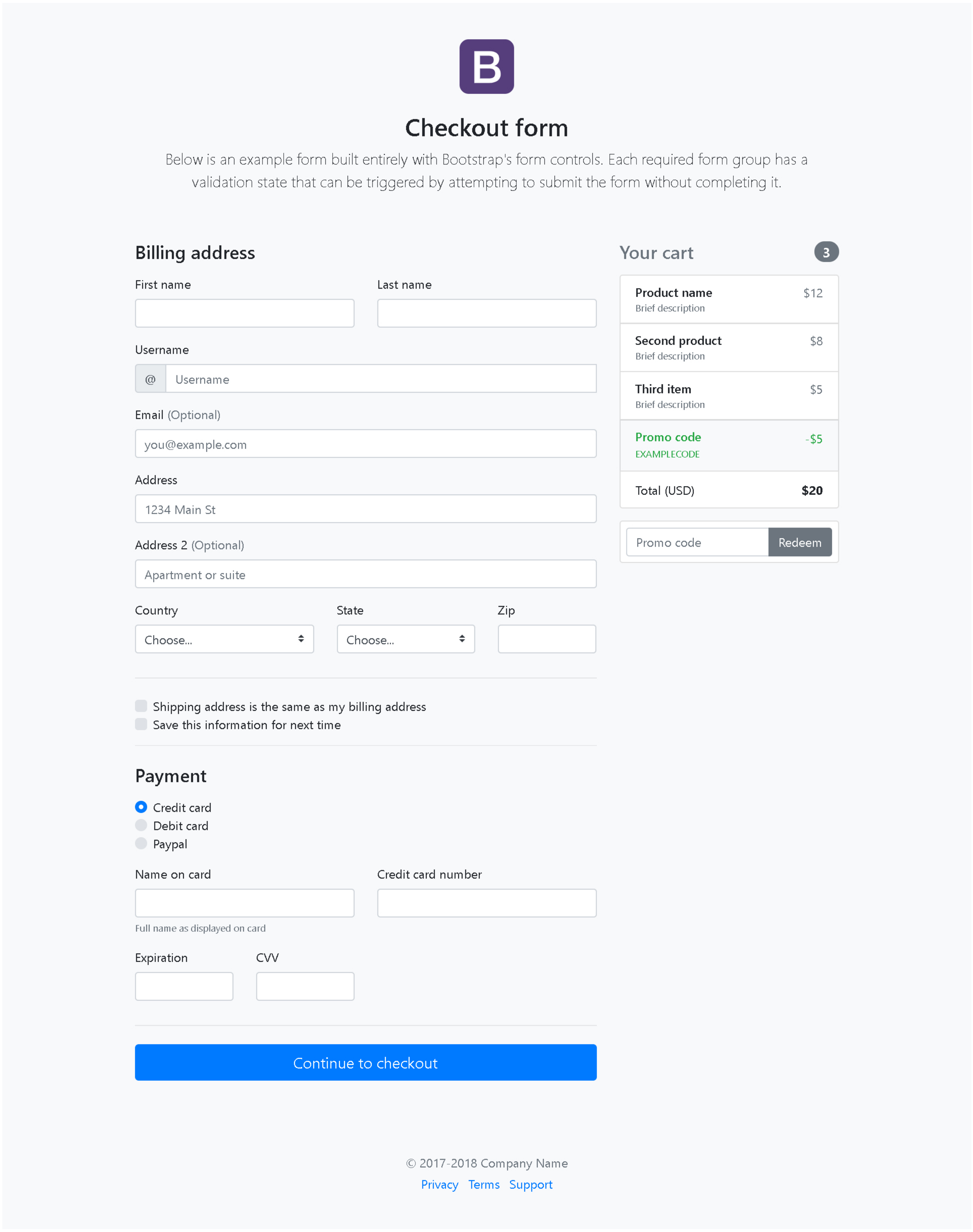
Cool stuff
Random feature
Team feature
Stuff for developers
Another one
Last time

Resources

Resource
Resource name
Another resource
Final resource

About

Team
Locations
Privacy
Terms



Checkout form

Below is an example form built entirely with Bootstrap's form controls. Each required form group has a validation state that can be triggered by attempting to submit the form without completing it.

Billing address

First name

Last name

Username

@ Username

Email (Optional)

you@example.com

Address

1234 Main St

Address 2 (Optional)

Apartment or suite

Country: Choose...

State: Choose...

Zip

☐ Shipping address is the same as my billing address

☐ Save this information for next time

Payment

● Credit card

○ Debit card

○ Paypal

Name on card

Full name as displayed on card

Credit card number

Expiration

CVV

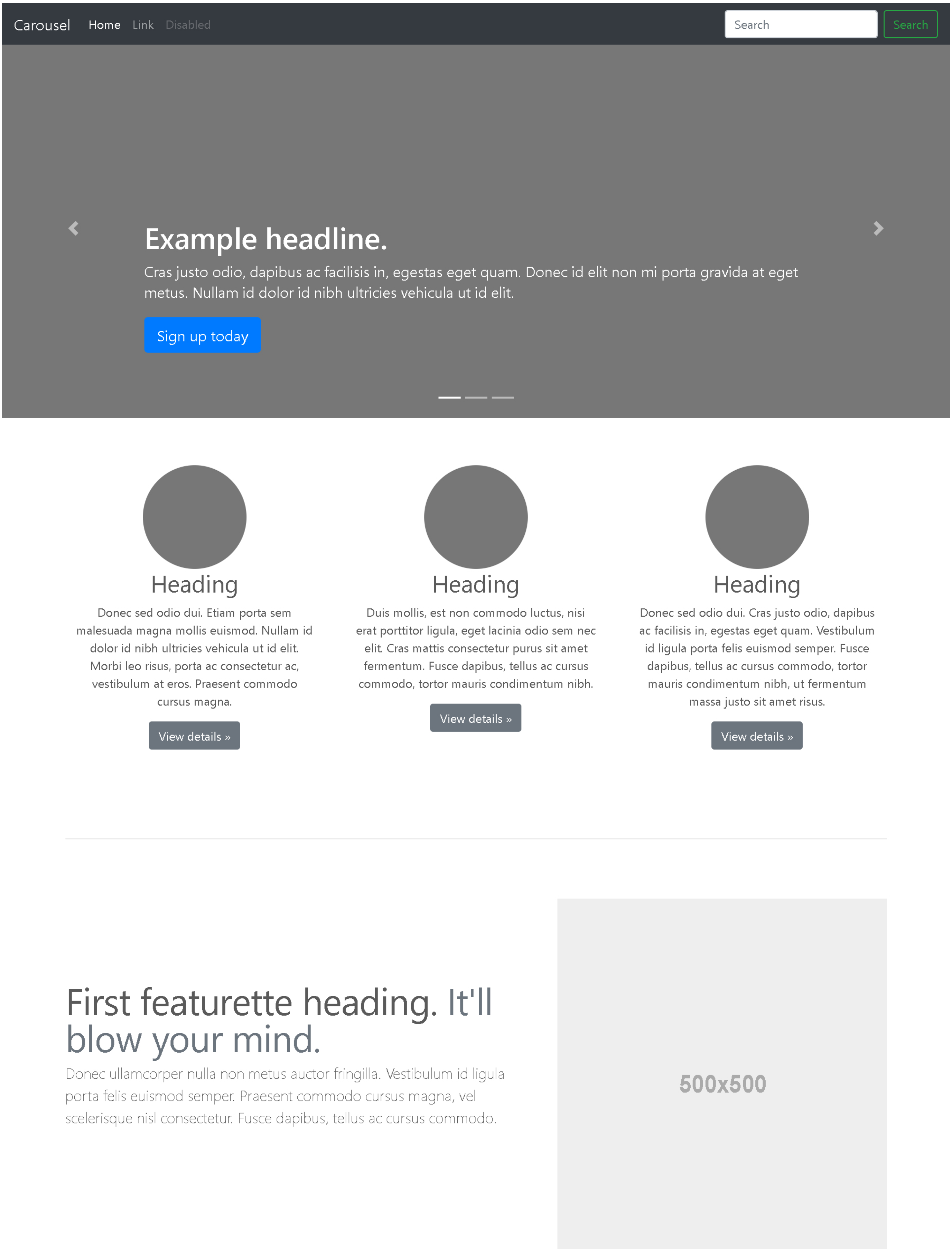
Continue to checkout

Your cart 3

Product name Brief description	$12
Second product Brief description	$8
Third item Brief description	$5
Promo code EXAMPLECODE	-$5
Total (USD)	**$20**

Promo code | Redeem

Privacy Terms Support



Example headline.

Cras justo odio, dapibus ac facilisis in, egestas eget quam. Donec id elit non mi porta gravida at eget metus. Nullam id dolor id nibh ultricies vehicula ut id elit.

Heading

Donec sed odio dui. Etiam porta sem malesuada magna mollis euismod. Nullam id dolor id nibh ultricies vehicula ut id elit. Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Praesent commodo cursus magna.

Heading

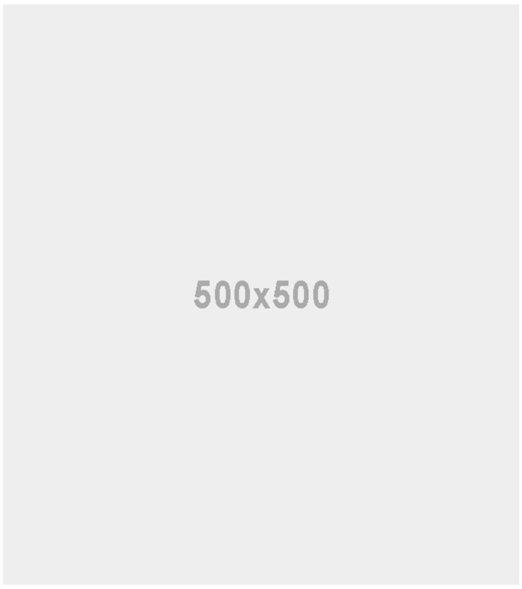
Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit. Cras mattis consectetur purus sit amet fermentum. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh.

Heading

Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

First featurette heading. It'll blow your mind.

Donec ullamcorper nulla non metus auctor fringilla. Vestibulum id ligula porta felis euismod semper. Praesent commodo cursus magna, vel scelerisque nisl consectetur. Fusce dapibus, tellus ac cursus commodo.

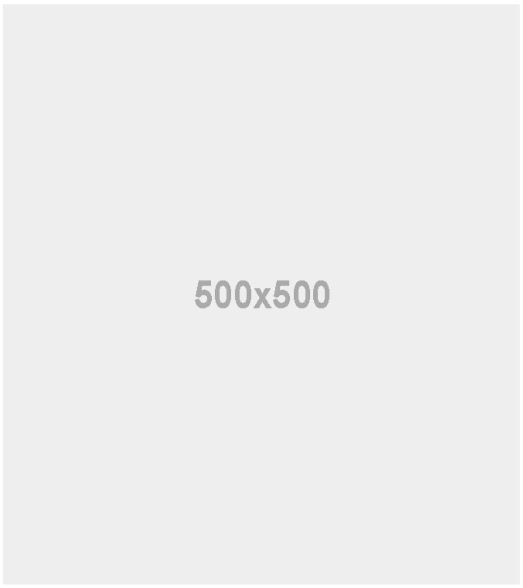


Oh yeah, it's that good. See for yourself.

Donec ullamcorper nulla non metus auctor fringilla. Vestibulum id ligula porta felis euismod semper. Praesent commodo cursus magna, vel scelerisque nisl consectetur. Fusce dapibus, tellus ac cursus commodo.

And lastly, this one. Checkmate.

Donec ullamcorper nulla non metus auctor fringilla. Vestibulum id ligula porta felis euismod semper. Praesent commodo cursus magna, vel scelerisque nisl consectetur. Fusce dapibus, tellus ac cursus commodo.



Back to top

Appendix D

Questionnaires Results

This appendix contains the number that each participant utilised each one of the components.

	Questionnaire 1				Questionnaire 2			Questionnaire 3	Questionnaire 4	
Component	Answer 1	Answer 2	Answer 3	Answer 4	Answer 1	Answer 2	Answer 3	Answer 1	Answer 1	Answer 2
Paragraph or Text	1	2	1	4	2	2	1	33	3	2
Div	0	0	0	1	0	0	0	26	1	1
Input	0	0	0	0	0	0	0	19	0	0
Title or Headings	1	2	1	3	0	1	1	4	2	2
Button	1	2	0	3	1	0	1	1	3	1
Br or Separator	0	0	0	0	0	1	1	3	1	1
Footer	1	1	0	1	1	1	0	0	1	1
Navbar	1	1	0	1	1	0	1	0	1	1
A or Links	1	1	0	2	0	1	0	0	1	0
Container	1	0	0	1	1	0	0	0	2	0
Image	0	0	0	0	0	0	0	0	2	2
Header	0	1	1	1	0	1	0	0	0	0
List	0	0	1	0	0	0	0	0	2	0
Card	0	1	0	0	0	1	0	0	0	0
Carousel	0	0	0	0	0	0	0	0	0	1
List Item	0	0	0	1	0	0	0	0	0	0
Menu Item	0	0	0	0	0	0	0	0	1	0
Section	1	0	0	0	0	0	0	0	0	0
Slider	0	0	0	0	0	0	0	0	1	0
Span	0	0	0	0	0	0	0	1	0	0
Thumbnail	0	1	0	0	0	0	0	0	0	0
Unordered List	0	0	0	1	0	0	0	0	0	0
Change element text	4	0	0	7	3	4	2	40	1	0
Type	0	0	0	0	0	0	0	19	0	0
Colour	0	0	0	2	0	0	0	8	3	0
Display	0	0	0	2	0	0	0	7	0	0
Width	2	1	0	1	1	0	0	0	1	1
Background Colour	0	0	0	1	0	0	0	4	1	0
Name on Navbar	0	0	0	1	1	1	1	0	1	0
Text Align	2	1	0	1	0	0	0	0	1	0
Links on Navbar	0	0	0	0	1	1	1	0	1	0
Font Size	0	0	0	1	1	1	0	0	0	0
Height	0	0	0	1	0	0	0	0	1	1
Border Radius	0	0	0	0	0	0	0	2	0	0
Href	0	1	0	1	0	0	0	0	0	0
Round Image	0	0	0	0	0	0	0	0	1	1
Align Items	0	0	0	0	0	0	0	1	0	0
Flex	0	0	0	0	0	0	0	1	0	0
Flex Direction	0	0	0	0	0	0	0	1	0	0
Float	0	0	0	0	0	0	0	1	0	0
Image on Navbar	1	0	0	0	0	0	0	0	0	0
Margin	0	0	0	0	0	0	0	1	0	0
Source	0	0	0	0	0	1	0	0	0	0
Add or Create elements	6	7	0	13	4	5	4	87	6	5
Define where the elements are created	3	0	0	1	2	0	2	94	2	2
Specify the value for the css command while creating element	4	1	0	4	3	2	2	60	0	1
Several instructions on the same command	3	5	0	10	1	4	3	13	6	3
Define id's and classes	0	0	0	0	1	0	0	32	0	0

Define columns or set element side by side	3	2	0	2	1	1	1	0	4	2
Bot ask if the user wants a certain element	0	0	8	0	0	3	0	0	0	0
Create elements on the navbar	1	0	1	1	1	0	1	0	1	1
Exit or Element is complete or Done	1	0	2	0	0	0	0	0	0	0
Save Project	0	0	0	0	0	1	1	0	1	0
Change the appearance of the elements in the navbar	0	0	0	0	1	1	0	0	0	0
Invert position of two elements	0	0	0	0	0	0	0	0	1	1
Actions that require JavaScript	0	0	0	1	0	0	0	0	0	0
Loading a template without removing the existing code	0	0	0	0	0	0	1	0	0	0
Open Project	0	0	0	0	0	1	0	0	0	0
Rows	0	1	0	0	0	0	0	0	0	0
Select a Element	0	0	0	0	0	0	0	0	1	0

Appendix E

Experiment Guide

This appendix contains the guide used for the experimentation and a text file that was provided in the experiment. This file details the commands that the participants can execute on the chatbot.

A Conversational Interface for Webpage Code Generation

This chatbot allows designers to create a prototype of a webpage without having to resort to hand-draw mockups by recognizing the intention of the commands entered. These commands can be written or spoken.

As this tool is novel and it takes some time to get used to the domain-specific language, let's start with a tutorial.

At any time, you can enter the command “help” to receive advice from the chatbot. This advice will vary according to the situation you are. The chatbot will also give some information without you asking.

Tutorial

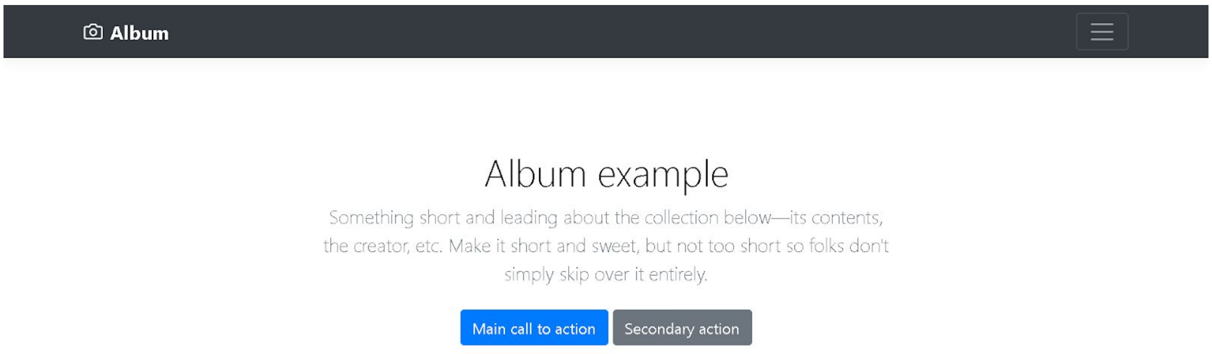
Task 1

In this task, you will learn how to create several elements and change their appearance by copying and pasting the following commands:

1. "add a navbar"
2. "change name to Album"
3. "change links" ("add a navbar with name links" would have been equivalent to these three first commands)
4. “three”
5. “Follow on Twitter”
6. “Add on Facebook”
7. “Email me”
8. “exit”
9. “add container”
10. “change padding” (“add container with padding” would have been equivalent to the 9th and 10th commands)
11. “add h1”
12. “change text to Album example”
13. “exit”
14. “add p with text”
15. “Something short and leading about the collection below—its contents, the creator, etc. Make it short and sweet, but not too short so folks don't simply skip over it entirely.”
16. “exit”
17. “add p”

18. “add button with text margin color background color”
19. “Main call to action”
20. “#ffffff”
21. “#007bff”
22. “make 1 copy”
23. “change text to Secondary action” (“text Secondary action” would have been a equivalent to this command)
24. “change background color to #6c757d” (“background color #6c757d” would have been equivalent to this command)
25. “exit”
26. “exit”
27. “text align center”
28. “save project as task1”

The final result should be similar to the following image.



Album

Album example

Something short and leading about the collection below—its contents, the creator, etc. Make it short and sweet, but not too short so folks don't simply skip over it entirely.

Task 2

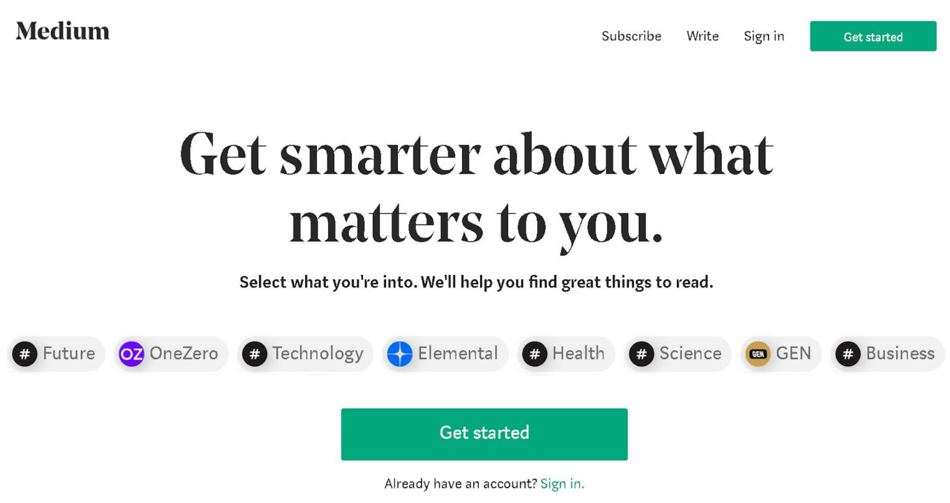
In this task, you will learn how to handle projects and how to load an already developed website to the chatbot. You will also be asked to upload images to the website and to get the final HTML and CSS code.

1. **Upload** the HTML file name tutorial2.html and the CSS file tutorial2.css;
2. **Upload** the image navbar.jpg;
3. **Select** the navbar;
4. **Change** the navbar image to the one previously uploaded;
5. **Upload** the image task2.jpg;
6. **Select** the image and **change** it to the one previously uploaded.

Problem 1

In the following steps, you will be asked for you to create a part of the Medium initial webpage.

1. Try to replicate the webpage shown below:



Get smarter about what matters to you.

Select what you're into. We'll help you find great things to read.

Already have an account? Sign in.

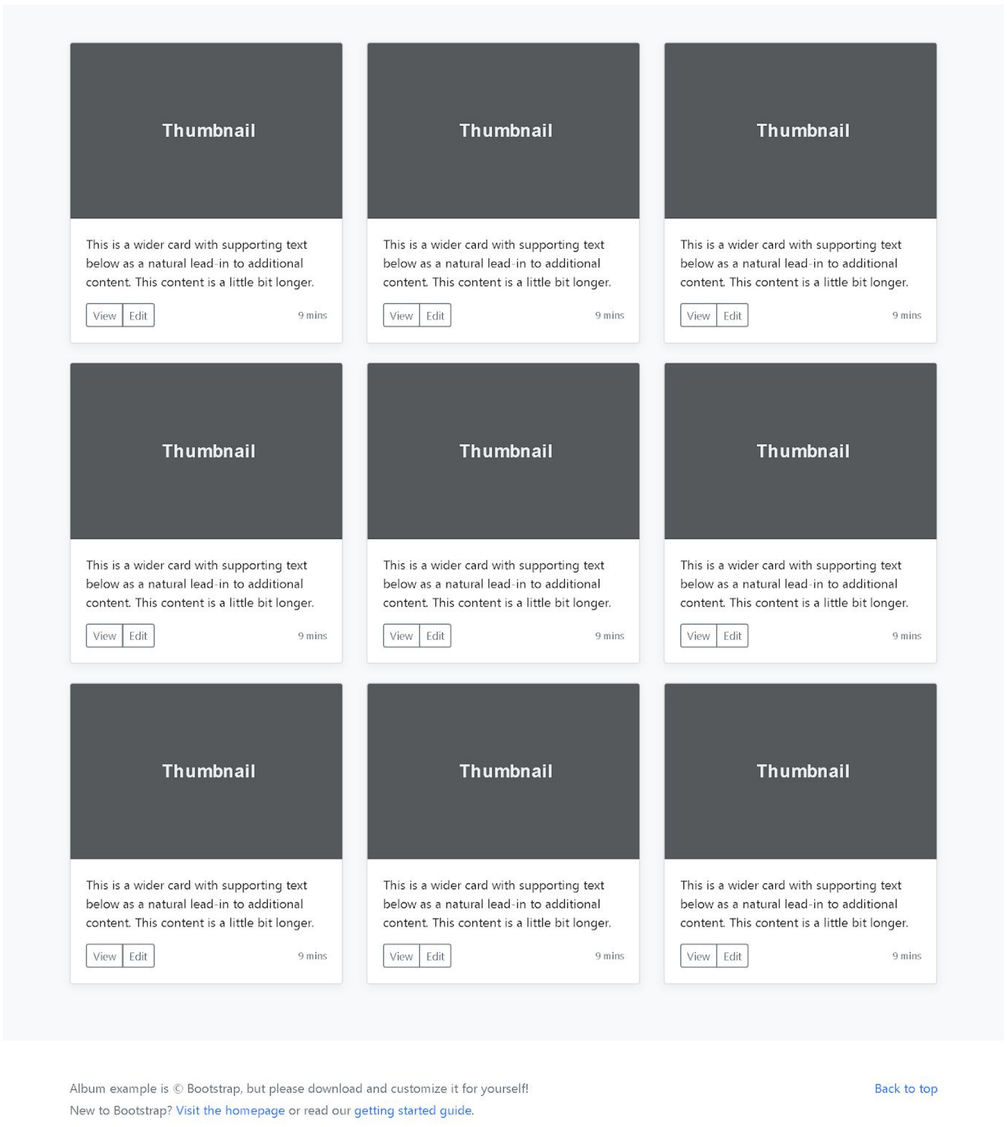
2. **Download** the HTML and CSS code as problem1.

Observations: The Get started button in the navbar should be equal as the other links. The grey buttons can have the same text and they should not have an image. The buttons should also be rectangular.

Problem 2

In the next steps, you will be asked to continue an existing project by adding new elements to the webpage.

1. **Open** the project you saved in Task 1;
2. Try to replicate the webpage shown below;



This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Album example is © Bootstrap, but please download and customize it for yourself!
New to Bootstrap? Visit the homepage or read our getting started guide.

Back to top

3. **Download** the HTML and CSS code as problem2.

Observations: The image source can be img.jpg; The text inside the cart is “This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.” The texts in the footer are “Album example is © Bootstrap, but please download and customize it for yourself!” and “New to Bootstrap? Visit the homepage or read our getting started guide.”

Questionnaire

https://forms.gle/hCQnsSRXoDM4qSTA9

Appendix F

File with the Commands

This appendix contains the text file containing the commands that the users can execute as well as the elements that can be created. This file was given to the participants of the experiment.

```
# Commands

- Add <Element>: Creates an element;
- Remove: Deletes an element;
- Select <Element>: Selects an element;
- Make <Number>: Creates copies of the element;
- Enter: Changes the element that is currently being modified;
- Exit: Changes the element that is currently being modified;
- Cancel: Invalidates the command that is being executed;
- Undo: Reverses the last command executed;
- Redo: Reverses the action of the undo;
- Clear: Removes all the elements;
- Hide Borders: Borders are removed on all the elements;
- Show Borders: Borders are revealed on all the elements;
- Download <Name>: Downloads a HTML and CSS code;
- Upload: Uploads a HTML and CSS code;
- Upload Image: Uploads an image;
- Save <Name> Saves the HTML and CSS code in the server;
- Open <Name>: Opens a HTML and CSS code that is located in the
  server;
- CSS Commands: Commands that change the appearence of the elements;

## Add

```

```
Allows the user to create an element. If an element is not provided,
     the chatbot will ask.

- Add, Create, Append

## Remove

Deletes the element (the one with orange border).

- Remove, Delete

## Select

Allows the user to change the selected element. If an element is not
     provided, the chatbot will ask.

If there are more than one element that can be selected, the chatbot
     colors all the possible elements and asks which one to select. (
    The user must respond with an ordinal).

- Select, Pick

## Make

Allows the user to make x copies of the selected element. If a
    number is not provided, the chatbot will asks.

- Make

## Enter

Changes the selected element to the last element added (the element
    with yellow background-color).

- Enter, Go in, Get in

## Exit

Changes the selected element to the parent of the current element.

```

```
- Exit, Go out, Get out

## Cancel

Allows the user to stop the current action. For example: the user
    says "add", the chatbot will ask What element do you want to add
    ?" and the user now says "Cancel" to not add any element.

- Cancel

## Undo

Allows the user to reverse to the previous state. For example, if
    the last action of the user was to add an element but now he
    realizes that it is not as he wants. He can utilise undo to
    recover the previous state.

- Undo

## Redo

Allows the user to reverse the action of the undo.

- Redo

## Clear

Allows the user to remove all the elements.

- Clear

## Hide Borders

Allows the user to remove all the borders that are shown to help to
    visualize the elements.

- Hide borders

## Show Borders

```

```
Allows the user to reveal all the borders to help to visualize the
   elements.

- Show borders

## Download

Allows the user to download a html and css file.

- Download

## Upload

Allows the user to upload a html and css file.

- Upload

## Upload Image

Allows the user to upload an image.

- Upload image

## Save

Allows the user to save a project in the server.

- Save

## Open

Allows the user to open a project that is located on the server

- Open

## CSS Commands

Allows the user to change the appearence of the selected element.

### Possible CSS Commands
```

```

- text ...: allows to change the text inside of the element (only
   for H1, H2, H3, H4, H5, H6, Button and P elements).
- name ...: allows to change the name in the navbar (only for navbar
    element).
- logo ...: allows to change the logo in the navbar (only for navbar
    element).
- links: allows to change the links in the navbar (only for navbar
   element).
- source ...: allows to change the url in the image (only for image
   element).
- type ...: allows to change the type of the input (only for input
   element).
- text align ...: allows to change the text align of the element.
- color ...: allows to change the color of the element.
- background color ...: allows to change the background color of the
    element.
- margin ...: allows to change the margin of the element and as the
   default value of 1em.
- padding ...: allows to change the padding of the element and as
   the default value of 1em.
- border ...: allows to change the border of the element.
- border width ...: allows to change the border width of the element
   .
- border style ...: allows to change the border style of the element
   .
- border color ...: allows to change the border color of the element
   .
- width ...: allows to change the width of the element.
- height ...: allows to change the height of the element.
- font size ...: allows to change the size of the text.
- same row: the childs of the current element will be on one row.
- same column: the childs of the current element will be on one
   column.

If the user did not provided a value for the command, the chatbot
   will ask for it.


## Elements
```

```

- navbar
- h1
- h2
- h3
- h4
- h5
- h6
- p
- button
- div
- container
- image
- footer
- form
- input
- unordered list
- ordered list
- list item
- span
- label

## Multiple Commands

When adding a new element it is possible to execute multiple
    commands.

Examples:

- If the user enters the command 'add navbar with name logo links',
    the bot will ask for the name, the url of the logo and how many
    links and the text for each link.
- If the user enters the command 'add h1 with margin text', the bot
    will ask for the value of margin and what is the text.

# Voice Recognition

You can click in the microphone button and say the command you want
    to perform that the bot will try to understand what you said.
```

Appendix G

Expected results of the Experiment Problems

This appendix contains the expected results of the two problems that were asked to resolve in the experience.

Medium

Subscribe Write Sign in Get started

Get smarter about what matters to you.

Select what you're into. We'll help you find great things to read.

Future Future Future Future Future Future Future Future

Get started

Already have an account? Sign in

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View View 9 mins

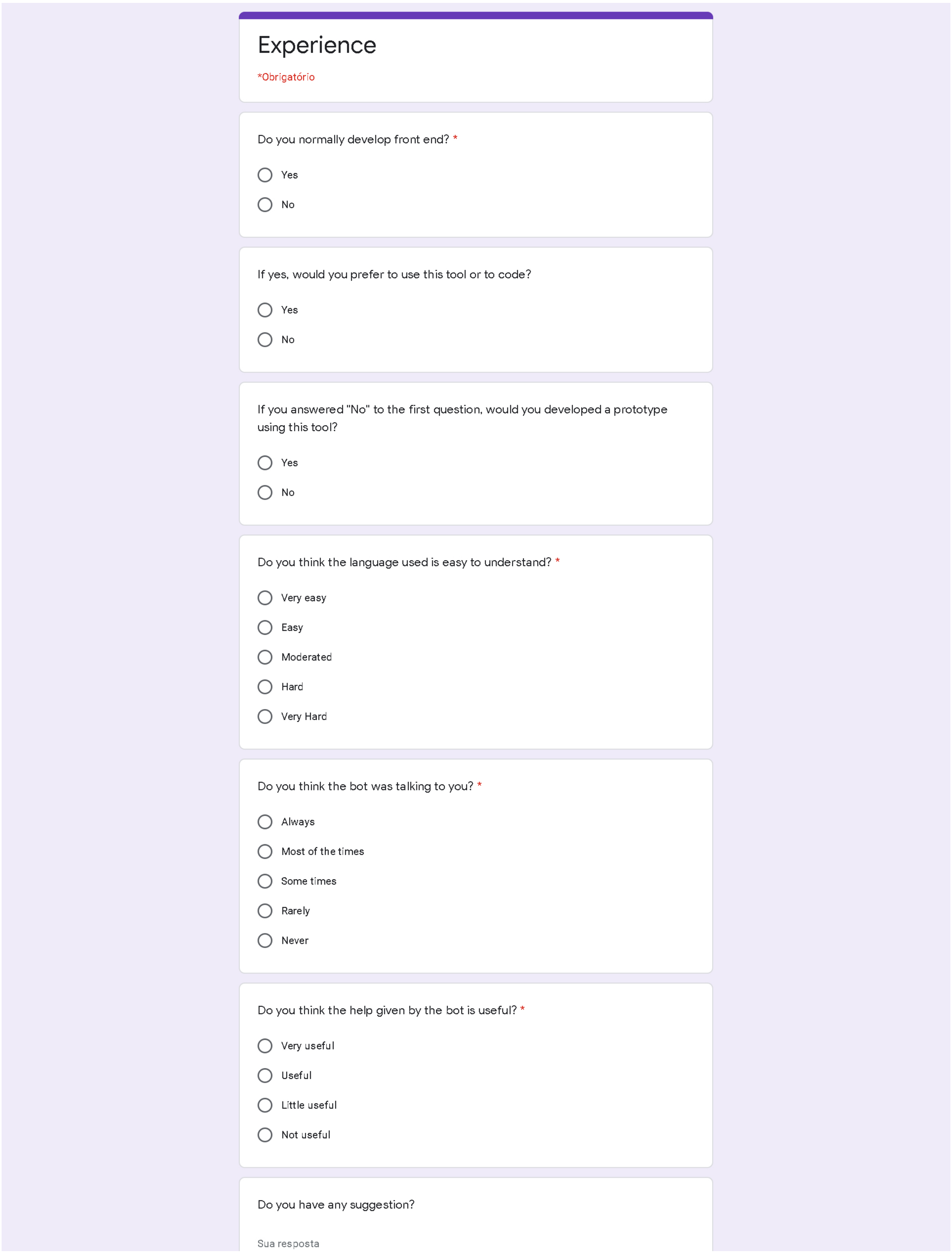
Album example is © Bootstrap, but please download and customize it for yourself!
New to Bootstrap? Visit the homepage or read our getting started guide.

Back to top

Appendix H

Experiment Questionnaire

This appendix contains the questionnaire that was sent after the participants completed the experiment.



Experience

*Obrigatório

Do you normally develop front end? *

- ◯ Yes
- ◯ No

If yes, would you prefer to use this tool or to code?

- ◯ Yes
- ◯ No

If you answered "No" to the first question, would you developed a prototype using this tool?

- ◯ Yes
- ◯ No

Do you think the language used is easy to understand? *

- ◯ Very easy
- ◯ Easy
- ◯ Moderated
- ◯ Hard
- ◯ Very Hard

Do you think the bot was talking to you? *

- ◯ Always
- ◯ Most of the times
- ◯ Some times
- ◯ Rarely
- ◯ Never

Do you think the help given by the bot is useful? *

- ◯ Very useful
- ◯ Useful
- ◯ Little useful
- ◯ Not useful

Do you have any suggestion?

Sua resposta

Enviar

Appendix I

Experiment Results

This appendix contains one table and two figures. The first one details the time and the number of commands the each participant required to solve each problem of the experiment. The figures tables contain the number of commands of each type that the each participant needed to complete the first and second problem of the experiment.

	Problem 1		Problem 2	
Participant	**Time (min:ss)**	**Number of commands**	**Time (min:ss)**	**Number of commands**
1	11:17	38	10:07	45
2	11:28	38	10:13	73
3	10:19	47	9:50	62
4	10:57	35	14:13	46
5	12:23	26	13:17	50
6	8:23	29	10:21	55
7	11:37	44	19:34	51
8	13:21	30	10:13	47
9	13:15	34	13:08	52
10	8:58	45	11:24	58
11	10:51	26	12:15	53

Table I.1: Time and number of commands that each participant required to complete each problem

	Add	Remove	Change	Undo	Redo	Select	Make	Enter	Exit	Cancel	Help	Clear	Count
User 1	11	1	14	3	0	0	1	0	6	0	2	0	38
User 2	11	1	14	2	0	0	1	0	7	2	0	0	38
User 3	14	2	11	6	0	2	1	1	9	0	1	0	47
User 4	11	0	10	2	0	1	1	1	7	0	2	0	35
User 5	10	2	6	0	0	0	1	1	6	0	0	0	26
User 6	9	0	11	0	0	0	1	0	7	0	1	0	29
User 7	8	1	23	2	0	1	2	0	7	0	0	0	44
User 8	9	0	10	0	0	0	1	0	7	0	3	0	30
User 9	8	0	16	1	0	1	1	0	5	0	2	0	34
User 10	12	1	12	6	0	4	3	0	7	0	0	0	45
User 11	9	0	8	1	0	0	1	0	7	0	0	0	26

Figure I.1: Number of commands of each type that the each participant needed to complete the Problem 1

	Add	Remove	Change	Undo	Redo	Select	Make	Enter	Exit	Cancel	Help	Clear	Count
User 1	14	0	11	1	0	0	3	0	13	0	3	0	45
User 2	17	0	20	12	1	0	3	2	15	2	1	0	73
User 3	21	0	16	4	0	2	3	0	15	0	0	1	62
User 4	13	0	13	3	1	0	3	0	12	0	1	0	46
User 5	16	2	9	2	0	1	3	1	16	0	0	0	50
User 6	18	0	14	2	0	1	2	0	13	0	5	0	55
User 7	16	3	15	0	0	0	4	0	12	1	0	0	51
User 8	15	0	11	1	0	0	3	1	14	0	2	0	47
User 9	16	0	17	2	0	0	2	0	14	0	1	0	52
User 10	20	0	10	6	0	1	3	2	15	0	0	1	58
User 11	15	0	14	4	1	0	2	0	14	0	3	0	53

Figure I.2: Number of commands of each type that the each participant needed to complete the Problem 2