TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ALEKSANDAR SVETOSLAVOV TONKIN

WEB INTERFACE FOR NATURAL LANGUAGE PROCESSING ENGINES

Master of Science thesis

Examiner: Asst. prof. Petri Ihantola
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering May 4th 2016

ABSTRACT

ALEKSANDAR SVETOSLAVOV TONKIN: Web Interface for natural language processing engines
Tampere University of technology
Master of Science Thesis, 45 pages
May 2016
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Asst. prof. Petri Ihantola

Keywords: natural language processing tasks, natural language processing engines, part-of speech tagging, interface, web application, web development

This thesis presents a graphical user interface that simplifies the use of natural language processing engines. Until now the use of the implemented part-of-speech task and other natural language processing tasks were possible only through command line interface. This restriction of the current interface was the reason, because of which the users were required to have both programming and scripting knowledge and experience. In addition to that, the preparation of the input data, used to feed the engine, and output data from the tasks had to be handled manually.

The problem was solved by developing a web-based application and integrating OpenNLP engine and its part-of-speech tagging task. Hence, the chosen solution provides portability and accessibility from various locations. The user interface simplifies the working process for users with little or no technical knowledge and experience. Moreover, the application facilitates and guides the users through the task's process flow. It also allows them to automatically preprocess their data to a format required as an input for the engine. After that, the users can simply follow the stages for the rest of the task and use the engine. Moreover, the application saves the data at the end of every stage of the task, which does not enroll the user to execute the whole task at once. Also the data used as an input and output from every stage of the task is stored automatically on the server, which provides reusability. And last, the application has modular structure, which provides the ability to extend the amount of tasks and engines according to the needs of the users.

The application provides simple interface, automated process and file handling. However the speed and accessibility of the application depends on the connection and the load on the server. The current software can be expanded with additional engines and tasks, but only if they support the current file and system structure.

In the future some parts of the user interface and the back-end structure could be improved, as well as some more complexed tasks and different engines could be implemented.

PREFACE

I would like to thank assistant professor Petri Ihantola for his great help and mentoring for the finalization of this thesis. Also I would like to pay my gratitude to all the colleagues from Lionbridge Oy for the opportunity to do both the project and this thesis, and my family for their moral support.

Tampere, 25.05.2016

Aleksandar Svetoslavov Tonkin

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Natural language processing is an area in computer science that is closely related to disciplines such as machine learning, artificial intelligence, linguistics, statistics and information theory. The main idea behind natural language processing is to make computers generate, understand and analyse written and spoken languages, as well as or maybe even better that any human can.

There are many different methods used for natural language processing, which can be accessed through so called natural language processing engines or toolkits. A toolkit is a set of tools, such as software libraries, documentations or scripts, with defined and specific purpose. They are often used to execute the tasks of the users' code. Therefore, if one needs to use many of these engines, then this person is required to have at least some programming or scripting skills. This requirement creates a huge gap when it comes to the usage of these engines by nontechnical users, such as linguists. And considering that this might appear a challenging task for such kind of users to interact with command line user interface or dealing with scripts, appears the need for a software with user interface that will allow processing natural languages.

To solve the aforementioned issues, an application was developed, that aimed to meet the requirements of the nontechnical users. This software gives the users the freedom to use the engines and access their tasks without the need of any programming or scripting skills. The application provides a flexible process flow that can be paused and continued at any step, as well as the natural language data, used as an input, is processed and stored automatically.

The work in this thesis is divided in seven chapter. Chapter 2 is introductory chapter to the user interface types that take place in the current project. Moreover, it defines the problem and the criteria and sets the scope of this thesis. Chapter 3 provides background information about natural language processing, OpenNLP engine and its supported tasks, which take part in the current solution. Chapter 4 explains the background on web applications, gives an overview on user interface design and introduces client-server architecture. Chapter 5 explains the structure of the application, its user interface, how the process is divided into stages and how they work. In chapter 6 the solution of the problem is being evaluated and it is observed how the criteria were fulfilled. The last chapter provides a summary of the current project and how some issues, that are currently available, can be fixed in the future.

2. PROBLEM AND CRITERIA

This chapter will introduce the user interface types that take part in this thesis. Moreover, it will define the problem when it comes to using natural language processing engines and accessing their tasks as well as the solution criteria will be set. In the end of this chapter, the labor division, for the project, will be explained and the scope of this thesis will be set.

2.1 User Interfaces

User interface (UI) in general describes the space where humans are interacting with machines. According to the area of use there are different types of UIs. In software engineering user interface can be described as the junction between a user and a computer program. User interface is a part from any software and it is constructed to provide an insight view of the software to the users. In the most abstract understanding, a good user interface is attractive, simple to use, responsive in terms of speed, easily understandable and consistent among all its screens [20].

In computer science, user interfaces can be divided into two groups: command line interface (CLI) and graphical user interface (GUI). CLI is maybe the oldest interface method and it was the primary way of interacting with computers until the invention of the video display terminal. This is a character-based method that requires the use of keyboard [20]. The users have to type their commands and as response the computer prints out the result back on the screen. In order to properly use CLI, users have to learn and remember a huge amount of commands. The most famous examples of CLIs are the command prompt in Windows and the terminal in Linux operating systems. In contrast to CLI, GUI allows the users to graphically interact with the system by using windows, icons and menus, which can be manipulated by various input device, for instance mouse or keyboard. GUI uses more resources than CLI, due to that fact it has to load more visual components and devices. GUI provides the users with some advantages, such as simplicity in the process of learning and using, visualization of the executed actions and multitasking [5]. On the other hand CLI still remains preferred user interface for the execution of some tasks or by experienced users when interacting with a software, because in many cases it provides more concise and powerful means of control. Figure 1 provides a visual example about one difference between CLI and GUI. On the left side of the figure is an example of two copy commands done by using CLI and the right side shows drag and drop copy method in GUI environment. As you can see CLI requires just one command line to copy a file, while GUI needs both source and destination windows open in order to execute the same action. However, non-technical users prefer the GUI method as more convenient. Users can visually navigate to both locations and produce the same operation without the need

to remember any commands or location addresses, which leads to smaller possibility of making errors when typing the command in CLI or copying the file in a wrong location.

Figure 1. *CLI versus GUI example*

Mentioning UI it is also important to introduce the application programming interface (API). While UI refers to an interface used by the users to interact with application, API is an interface that provides a way one application to interact with or control another. API is a set of libraries and tools, used when building a GUI application. In some cases one can think of them as they are building blocks that a developers has to combine with theirs in order that application to communicate with others [21]. In others, API refers to a group of requirements on how applications can communicate between. But in the end, all that API does is to "reveal" some of the program's internal functionality to the outside world in a limited manner, which allows data sharing and using the program's functions. Advantage of using API's is that one does not need to know details on how the external application have been created, but instead only use its functionality.

2.2 Natural language processing engine's problem

This thesis has the task to provide a solution for several problems concerning the access and usage of natural language processing engines. The one problem involves the users that has to perform their tasks by using these engines. The targeted group of users can be categorized as commons users. They do not possess any specific technical skill, such as administration, programming or scripting. The problem that they face, when using the natural language processing language engines, is that they need to spend time on learning how to program or create scripts beforehand in order to use the tasks. This is caused, due to the reason that the main way of interacting with these toolkits is through command line interface. Even though, these engines also provide API functionality, there are no available GUI applications at this point. Another problem is the portability and accessibility. In order to use these engines, the users are constrained to have them deployed on their local working stations. In this case the access to these toolkits is limited and they cannot be reached from a remote location.

2.3 Criteria definition

The project in this thesis aims to facilitate and guide the workflow when using and accessing natural language processing engines and their tasks through an application. It is to simplify the working process so as to require no prerequisite technical knowledge, say, skills in programming, installation, or command-line interaction. Its interface should provide expandability to any number of engines and tasks according the needs of the users. The application should be independent of operating systems and easily supported and accessed by the users, without the need of individual installations on their working computers, from various locations. The interface should provide flexibility in the workflow as needed by the users.

A good way to evaluate the application design is by using heuristic evaluation. This is a method that identifies usability problems in the process of user interface design by using usability principles called heuristics. Heuristic evaluation method is widely used nowadays, especially for projects that experience lack of resources such as time or budget for additional interface testing. This method has been developed by Jakob Nielsen, who has come up with a set of heuristics called "10 Usability Heuristics for User Interface Design" [36]. Let us now take a look at these heuristics. The first one is "*Visibility of system status"* and it refers to system visibility. It should inform the users what is happening with the use of some kind of feedback. *Match between system and the real world* states that the system should follow the real-world convention and information should be displayed in natural and logical way. *User control and freedom* explains that a system should always provide clear means of leaving, going backwards or forwards. *Consistency and standards* focus on the design where the same elements should be visualized in the same way. *Error prevention* heuristic refers to error prevention and states that it is always good to have an error message to prevent the error, than actually experiencing it and then notifying the user. *Recognition rather than recall* concerns the user's memory load. When designing, it is better to make more actions and options visible, rather than asking users to remember their locations. *Flexibility and efficiency of use* heuristic describes the need of interaction between unexperienced and experience users. *Aesthetic and minimalist design* concerns the interface in a sense that it need to be designed as simple as possible. *Help users recognize, diagnose, and recover from errors* principle focuses on the error messages that should provide a detail explanation of the errors and a possible solution. The last heuristic ("*Help and documentation*") focuses on the documentation of the interface that should be always present even if it is not being used.[36]

A web application, with clients and servers working over the Internet, will obviously provide for solutions for many of the above considerations. Lionbridge [8], the company to commission the application, required the use of Microsoft ASP.NET framework [9] and C# language [10] for the server-side coding. This is due to the application was to be deployed on a server running a Windows operating system [11]. Moreover, at least the

OpenNLP natural language processing engine [1] was to be accessible through the application, as well as its part-of-speech tagging task has to be implemented. Both OpenNLP and its task will be explained in the following chapter.

2.4 Scope

The practical work performed for this thesis was done in the cooperation of Grigorij Ljubin Saveski [4]. The project labor itself was divided into three parts: client-side, server-side and integration of OpenNLP engine [1] [2]. The client-side refers to the graphical user interface that interacts with the user, in our case this is the web browser. The server-side refers to all the operations performed on a remote server. Figure 2 presents an abstract structure of the application for the current problem. This thesis focuses on the GUI design that affect the client side (browser), what choices have been made during the design process, and also some server-side functionality will be presented, because the workload there was shared. Moreover, this thesis will provide some brief theoretical background on natural language processing, OpenNLP engine and its supported tasks, needed to understand the user interface. For more detail information about natural language processing refer to Grigorij Ljubin Saveski [4] work, which scope focuses on these matters.

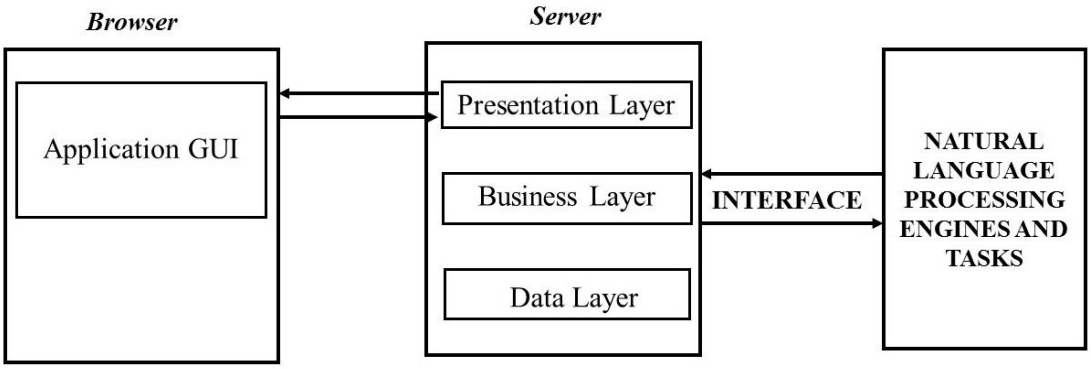

Figure 2. *Structure of the application*

3. NATURAL LANGUAGE PROCESSING AND OPENNLP

This chapter will provide background on what is natural language processing, its tasks and some natural language processing approaches. Moreover, the engine used in the project will be introduced as well as the tasks it supports and some example of how they can be accessed.

3.1 Natural language processing

Natural languages refer to one type of communication between humans. The problem, that machines are facing when it comes to understanding speech and text, is that people do not always speak or write plainly. In many languages there are words that have more than one meaning and according to the context the overall meaning is being changed too. These levels of ambiguity in languages cause the raising of many problems in natural language processing.

Humans use different linguistic points of view to analyse the data when it comes to languages. According to Liddy [18], by using different "levels of language" approaches (phonological, morphological, lexical, semantical, or syntactical) numerous meanings can be observed, for instance what is written, spoken or what the context of the data is. Moreover, for the successful creation of a human-like language system (so far this has not been accomplished), one must create such a system that uses multiple levels simultaneously, like humans usually do [18]. Liddy [18] explains the difference between natural language processing and natural language understanding. A machine is able to understand, when it is capable to achieve several different goals. The first of them is that the system should be able to paraphrase an input. Secondly, it should be able to translate the input in various languages without any problems. The third goal requires for the machine to be able to answer questions based on the text. And finally it should be able to summarise it. However, while natural language processing has achieved some progress in the first three goals, the forth one is still being an obstacle towards natural language understanding [18], which in fact remains its final goal.

In natural language processing, the parts used for solving the issues are known as natural language processing tasks. The main role of these tasks is to handle the given data text or corpora (large textual data collection) in various ways, like extracting a meaning or serving as subtasks for solving larger ones [4] [6]. More information about some tasks will be provided later in the next Section 3.2. The use of these tasks is possible through the natural language processing engines, such as OpenNLP [1], Natural Language Toolkit (NLTK) [12] or Stanford NLP [13]. Some of the widely used tasks supported from such engines

are part-of-speech tagging, tokenizing, name entity extraction, sentence detection, parsing and chunking. Many of these tasks are based on different paradigms, such as machine learning or pattern recognition [6] [15]. In order for a natural language processing task to be executed, a set of steps, called stages, has to be followed. Often it can be observed that some of these stages are being recurrently executed, for instance the training stage that is regularly followed by an evaluation stage (Figure 3). The following paragraphs will provide a brief introduction to these and some other stages. According to the process or the task requirements, additional stages can be added in the flow.

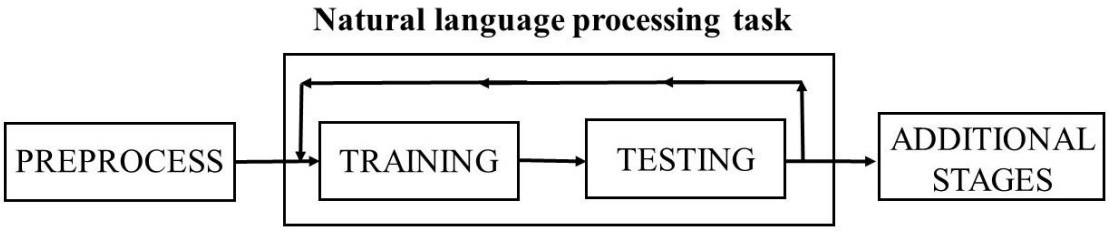

Figure 3. *Repeatedly executed stages in tasks*

Training a model for a task provides a task language model, which purpose is to make predictions based on the tasks. After this stage is completed, the accuracy of the model depends on the information used during this stage. There exists different forms of training. Two of the most famous ones are supervised and unsupervised [15] [16], where the former one is caused by using annotated data and latter one by using unannotated data. Nevertheless, there are some tasks or engines with supervised training that requires some manipulations over the input data before the actual training stage.

When such tasks or engines are in scope, this means that they accept only a particular data template as an input. Therefore, for the process to continue without any errors, a data set with this specific format has to be provided. However, the available training data may not be formatted properly, may contain errors or unnecessary extra information. In this case a preprocessing stage has to be executed before the training. As a result the training stage will receive formatted data stripped from any additional items, which will ensure better prevention of errors [4].

The evaluation stages is used to test the model and to ensure that it meets the requirements of the task. There exists different techniques that can be used in this stage to obtain the precision of the model, such as cross validation [15]. The idea behind this type of validation is that it depends on the input. The data is being divided into groups, where all the groups are being used to train a model, except for one, that is being used for testing. After each group is trained, it is being evaluated. The precision is calculated as average of the testing scores of all groups. The method used in the current thesis divides the data into two sets, namely training and testing data. The size of these sets is usually set beforehand. However, in this particular case it may vary due to the freedom of the users to

choose the proportions they desire during runtime. During the testing stage, the model is being evaluated by calculating the ratio between the amount of correct predictions and the total amount of predictions. This stage is followed either by continuing with another stage or going back to the training stage if the model's precision level has not been reached.

There exists roughly four approaches to handle natural language processing and their tasks. They are symbolic, statistical (probabilistic), connectionist and hybrid [18]. As mention before, this thesis focuses on part-of-speech tagging task and therefore we will briefly pay attention on the statistical method, since it is the one that is being used. This approach do not require any linguistic knowledge, because it uses statistical methods to set the rules when training a model for particular task [4]. Many probabilistic methods are used in natural language processing, but only one will be considered, namely maximum entropy. The principle of this method states that if there is more than one choice and it is not clear which one is more possible to happen, then both should be considered with the same probability (uniform distribution) [4].

To make it easily to understand let us apply the maximum entropy principle to a natural language processing task. Let this task be part-of-speech tagging task. The first thing required is a large set of data or a corpus that will be used for training of the model for the current task. Let us now say that there are few words that in the current data are met more than twice. Moreover, let us set these words to have more than one meaning and to be tagged as a noun, a verb or an adjective. After training the model with this rule, each of these words should have the same probability, when it is encountered. However, if in another piece of the input these words has been set preferably as nouns and verbs, then this second rule on the other hand will increase the probability for these two tags. Therefore, once the training is over there are two rules for these words. Since the model was created by using maximum entropy, every time when one of these words is encountered, into account will be taken the probabilities from these two rules and they will be tagged in a way that satisfies both rules and also as uniform as possible. However, in natural language processing this kind of probabilistic rules are not all that is taken into account when dealing with maximum entropy models. Another rules can be tied to the context, for instance previously assigned tags or the order of the words [4].

3.2 OpenNLP

Apache OpenNLP is a natural language processing toolkit based on machine learning that was developed to serve as a sophisticated framework for the purpose of natural language processing tasks and in the process of building more complex natural language processing systems [1]. It provides its users with the ability to create models for multiple natural language processing tasks [1]. Moreover, OpenNLP includes the use of maximum entropy framework for some of its tasks [1] [2] [7].

OpenNLP engine development process was initiated in 2000 by Jason Baldridge and Gann Bierner. In one of the web sites that provides information about the engine [3] it is stated: "OpenNLP, broadly speaking, was meant to be a high-level organizational unit for various open source software packages for natural language processing; more practically, it provided a high-level package name for various Java packages of the form opennlp.*". The first implantation of OpenNLP has started as Grok parsing toolkit [3] for natural language processing and offered OpenNLP Java API (application programming interface) [1] [2].

OpenNLP engine supports several different natural language processing tasks: sentence detection, tokenization, name finder, document categorizer, part-of-speech tagging, chunking, parsing and coreference resolution, which will be briefly explained in the current chapter as well as some examples will be given about how the training data should look like and what is the outcome from executing them. Some of these tasks are required to be executed beforehand in order more complex tasks to take turn. The relation between OpenNLP tasks is shown in Figure 1.

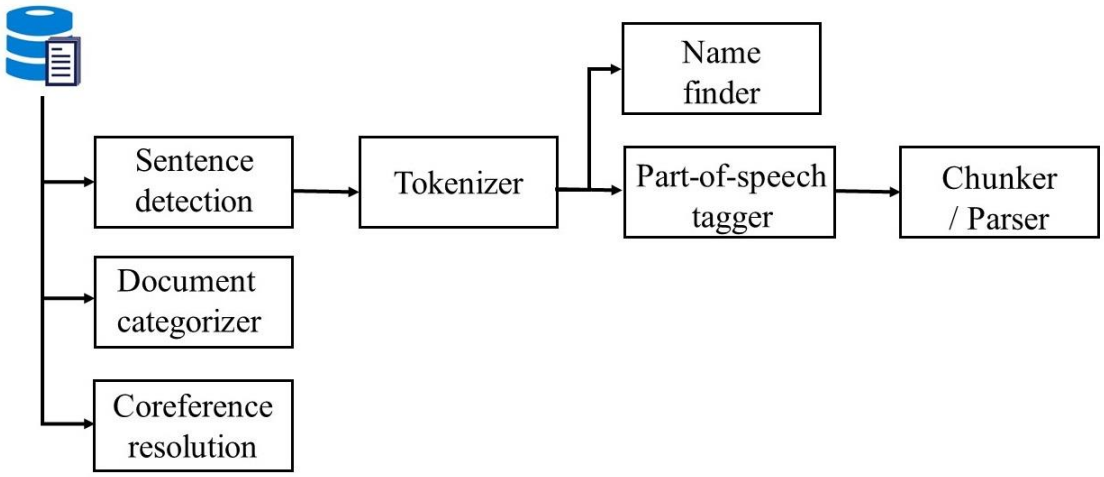

Figure 4. *Correlation between the tasks in OpenNLP*

All of these modules provide both application programming interface and command line interface [2], which can be used for training and, if necessary, for evaluating the tasks. By using these tasks one could build more sophisticated system. Moreover, OpenNLP provides a great amount of external packages, some of which are specific to certain language such as English, Spanish or German and others are used to store language rules. However, these packages are only available for a small amount of languages. There are also packages that can be used to convert a large amount of corpora to a format accepted by OpenNLP.

Sentence detector is the first task in OpenNLP, which provides the users with the functionality to identify sentences in a text by detecting that a particular punctuation character marks their end, as well as it can modify the whole text by separating each sentence on a

new line. Maximum entropy model is being used to evaluate if a particular punctuation character signify the end of the sentence. Therefore, the sentence detector task is based on rules and not on the actual content of the sentence [1]. In order for the Sentence detector to work properly a model for this task has to be trained. The data for the model needs to be converted to the "OpenNLP Sentence Detector training format" [1]. This format is simply one sentence per line and the end of the file is marked with an empty line. After that the model can be used on some random data to detect the sentences. The following example represents how this tasks actually works. The following text is given to the model as input:

"Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group. Rudolph Agnew, 55 years old and former chairman of Consolidated Gold Fields PLC, was named a director of this British industrial conglomerate. "

After detecting the sentences, the produced output would look like:

"Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group. Rudolph Agnew, 55 years old and former chairman of Consolidated Gold Fields PLC, was named a director of this British industrial conglomerate. "

The output after executing the task appears in a form where each sentence is printed on a new line [1].

Tokenizer splits the given input into tokens. The tokens usually include words, punctuation characters and numbers. This is a two-stage process, where first the text has to be divided using sentence detector, and then each sentence is being tokenized individually. There are three versions for OpenNLP tokenizer: whitespace, simple and learnable tokenizer. The former tokenizer identify non whitespace sequences as tokens. The latter one accepts as tokens the sequences of the same characters and the last one detects token boundaries based on probability model [1]. The tokenizer task also provide a detokenizing feature, which purpose is to reconstruct a string from tokenized sequence [1].

The training data for the tokenizer model, similar to sentence detector task, has its own format. This format is one sentence per line and the tokens are separated either by whitespace or by the tag <SPLIT> or by both.

Let us now train a model with the following example data:

Pierre Vinken<SPLIT>, 61 years old<SPLIT>, will join the board as a nonexecutive director Nov. 29<SPLIT>.

Mr. Vinken is chairman of Elsevier N.V.<SPLIT>, the Dutch publishing group<SPLIT>.

Now let us use the model over the same example sentences, but in this case they are stripped from the tags. The given output uses the whitespace tokenizer implementation:

"Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 . Mr. Vinken is chairman of Elsevier N.V. , the Dutch publishing group ."

After the execution of this task, the result is presented in such a form where every token (including the punctuation characters) are followed by whitespace [1].

Name finder is a task, which purpose is to identify names and numbers in the provided textual data. In order to perform the task, a model, trained over some corpora, is needed so that it can distinguish the name for the current language. Moreover, to train a name finder model, the sentence detector and tokenizer tasks has to be executed beforehand [1]. The following example visualize how name finder is used. Let us have these two sentences as an input text:

"Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 . Mr . Vinken is chairman of Elsevier N.V. , the Dutch publishing group ."

Then the output from the name finder task would be:

"<START:person> Pierre Vinken <END> , 61 years old , will join the board as a nonexecutive director Nov. 29 . Mr . <START:person> Vinken <END> is chairman of Elsevier N.V. , the Dutch publishing group ." [1].

As you can see only the names of people are tagged. This result is caused by the content of the data set used during the training stage.

Document categorizer task is used to classify input data into categories, which needs to be predefined by the users. The task is based on maximum entropy framework and a trained model, over some data, is required for the task to be performed. The input has to be pre-classified into groups, so later to be used over any input given to the model [1]. To make this sound more clear let's explain it with an example. The first thing that is required to train a model. The data format that OpenNLP accept is one document per line. This document consist the category and the text separated by whitespace. The following example consists of data and two categories GMIncrease and GMDecrease that are predefined by the users.

GMDecrease Major acquisitions that have a lower gross margin than the existing network also had a negative impact on the overall gross margin, but it should improve following the implementation of its integration strategies .

GMIncrease The upward movement of gross margin resulted from amounts pursuant to adjustments to obligations towards dealers .

After the model is created, let us use it on the same data but without the categories. When the first sentence is read it will be categorized as GMDecreased and the second one as GMIncreased.The classifications for the training data is user-specific, therefore there are no pre-build models that exist. The training data has to be created according the requirements for the task and the documents that are targeted.

Part-of-speech tagger tasks iterates through the input text token-by-token and predicts a tag for each of them based on maximum entropy part-of-speech tagging [4]. It is important to mention that the tagger uses probability model, where the prediction of a tag over another one depends by the token and the context. Tag dictionaries are optional to use and need to be provided by the users. These dictionaries are files that contain all available tags that are specific for every individual token. By using them the algorithm speed might get increased and also the number of possible tags for a token might get reduced [1], which will avoid a token to be assigned with an inappropriate tag.

The OpenNLP part-of-speech tagger uses the Penn Treebank tag set [1] to perform the task over the tokens. Before using the tagger, a model need to be trained beforehand. The training input needs to be correctly tokenized, annotated and formatted. Every sentence has to be on a separate line and to contain its tokens along with their tags. The combination of token and tag need to be split only with underscore such as "*token_tag*". Another important issue is that all the assigned tags in the training data has to be correct. An individual model has to be created for every language and relevant data in the same language has to be provided [1].

The following example explains how OpenNLP part-of-speech tagger works. An example sentence for training can be:

About_IN 10_CD Euro_NNP ,_, I_PRP reckon_VBP ._.

As you can see there is the token, followed by underscore and the tag, which is an abbreviation of the corresponding part of speech for the current language. For instance "PRP" tag means personal pronoun and "VBP" is Verb, non-3rd person singular present. For more information about all abbreviations of the tags refer to Penn Treebank tag set [31]

After the model is create, let us provide the following sentence that needs to be tagged.

"Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 . Mr. Vinken is chairman of Elsevier N.V. , the Dutch publishing group ."

As a result the part-of-speech tagger will output the following:

“Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS old_JJ ,_, will_MD join_VB the_DT board_NN as_IN a_DT nonexecutive_JJ director_NN Nov._NNP 29_CD ._. Mr._NNP Vinken_NNP is_VBZ chairman_NN of_IN Elsevier_NNP N.V._NNP ,_, the_DT Dutch_NNP publishing_VBG group_NN ._.” [1] .

Chunker task function is to group the various syntactical elements from the input data without specifying their internal structure or role in the sentence. To perform the task first a tokenized sentence has to be provided as an input. After that part-of-speech tagger is used to tag the words of the sentence and in the end they are split into syntactic groups, such as nouns, verbs, adjective. However, the data also needs to be properly formatted. For the chunker task training data format it is required that every word is on a separate line followed by two tags – the first one is its part-of-speech tag and the second is the chunk tag [1]. An example of training data can be:

```
He        PRP  B-NP
reckons   VBZ  B-VP
the       DT   B-NP
current   JJ   I-NP
account   NN   I-NP
deficit   NN   I-NP
will      MD   B-VP
narrow    VB   I-VP
to        TO   B-PP
only      RB   B-NP
#         #    I-NP
1.8       CD   I-NP
billion   CD   I-NP
in        IN   B-PP
September NNP  B-NP
.         .    O
```

As you can see the first column holds all the words in a sentence, the second column are the appropriate part-of-speech tags and the third one are the chunker tags. The chunker tags holds the name of the chunker type, for instance “I-NP” stands for “noun phrase words” and “I-VP” for “ verb phrase words”. The chunker tags are two types B-CHUNCK (first word of the chunk) and I-CHUNK (each other word in the chunk) [1].

After the models is trained, let us have the following sentence that is already tagged by part-of-speech tagger:

“Rockwell_NNP International_NNP Corp._NNP 's_POS Tulsa_NNP unit_NN said_VBD it_PRP signed_VBD a_DT tentative_JJ agreement_NN extending_VBG its_PRP$ contract_NN with_IN Boeing_NNP Co._NNP to_TO provide_VB structural_JJ parts_NNS for_IN Boeing_NNP 's_POS 747_CD jetliners_NNS ._.”

The produced output after chunking it would look like:

“[NP Rockwell_NNP International_NNP Corp._NNP] [NP 's_POS Tulsa_NNP unit_NN] [VP said_VBD] [NP it_PRP] [VP signed_VBD] [NP a_DT tentative_JJ

agreement_NN] [VP extending_VBG] [NP its_PRP$ contract_NN] [PP with_IN] [NP Boeing_NNP Co._NNP] [VP to_TO provide_VB] [NP structural_JJ parts_NNS] [PP for_IN] [NP Boeing_NNP] [NP 's_POS 747_CD jetliners_NNS] ._."

As a result the tokens, along with their tags, have been grouped by using square brackets [1].

Parser is used to divide the input data string into tokens, which afterwards are group according to their syntactic relation. The purpose of this tool is mainly for demonstration and testing. The best way for using it is through the command line interface. After the process is over it is possible for the parse tree to be printed on the screen if needed. When using the parser to train the model, a part-of speech tagger model will be also created so that the input can be tagged and parsed simultaneously. To achieve better accuracy for the result, the default tagger can be replaced with another trained on a larger corpus of data [1]. As an example let us have the following input sentence:

"The quick brown fox jumps over the lazy dog."

After using the parser model on that input, the output would be:

"(TOP (NP (NP (DT The) (JJ quick) (JJ brown) (NN fox) (NNS jumps)) (PP (IN over) (NP (DT the) (JJ lazy) (NN dog))) (. .))) " [1] .

In the result the words are grouped in a parse structure according to their relation on syntactical level. The tags in front of the words are actually part-of speech tags, while the tags that stands alone near the opening brackets are groups' phrases, for instance "NP" is noun phrases and "PP" stands for preposition phrases.

Coreference resolution is the last supported task by the OpenNLP engine. It purpose is to links multiple mentions of an entity in a document together [1]. However, the OpenNLP implementation is currently limited to noun phrase mentions only [1], because the functionality of this task is still in process of development.

OpenNLP provides a variety of natural languages processing tasks that are supported both by command line user interface and application programming interface. Part-of-speech tagger is one of the basic tasks and it is often uses as prerequisite in the developing process of more complexed algorithms, such as language dictionaries or spell-checkers. Tokenizer and sentence detector provide the base for more complexed tasks. Therefore, it can be concluded that OpenNLP is a powerful framework that can be used in the building process of more sophisticated natural language processing systems.

4. WEB APPLICATIONS

Web applications are Internet-based application programs, stored on a remote server and delivered over the HTTP [24]. They are client-server applications, where the client (front-end) is browser-based and the server (back-end) is known as web server. Web applications rely on a browser-like applications to render them and in most cases they are cross-browser compatible.

The term cross browsing refers to the ability of a web application to be properly rendered by all browsers. The major cause for this problem is due to different browsers, their versions, the operating system they work on, the screen size and the browser bugs [46] [47]. Cross browser applications focus on providing such a solution that preserves its functionality and layout when used by multiple browsers. In order for such applications to support cross browsing they have to be tested if they are rendered without any problems on the targeted group of browsers [46] [47]. Although, most of the browsers have similar features, small difference are expected, because of the rendering engines or the code interpreters they use, which may affect the overall of the application. Therefore, during the development and the testing of the web applications, they have to be adapted to work with different browsers.

There are many different languages that can be used in the developing process of web applications. However, there is no one best language for this task [37]. When creating web applications they are usually divided into two front-end and back-end. The most basic languages used for front-end web development are HTML [26] and CSS [26]. HTML stands for Hyper Text Markup Language and it is the backbone of any website. The HTML code provides the base framework of the website view. It consist of markup tags, where each tag describes different content in the HTML document. These tags are always in pairs – opening and closing one. In order to display the page content properly a browser-based application needs to be used, that reads the code and determine how things has to be shown based on the tags.[38]. Cascading Style Sheets (CSS) is a front-end language used to control the presentation of the web document and adds to the web site a unique look. CSS is independent from the HTML and can be stored in external CSS stylesheet files. Moreover, CSS saves a lot of work, due to the fact it can control the view of multiple pages simultaneously. The term cascade come from the fact that multiple style sheets can be applied on the same web page.[38].

Another programming language that is quite common is JavaScript and it is used for client-side scripting [37]. Client-side scripting (also includes HTML and CSS) concerns any code that run on the client side. This describes any action that is done on the page without the need to post back to the server. JavaScript is a lightweight, cross-platform, object-oriented, scripting language. The one of its uses is to manipulate the content of HTML

pages and to transform them from static into dynamic. For example, JavaScript can be used to validate user's input before it is being sent to the server or to update parts of the page without the need to reload the whole page. Nowadays, JavaScript is also used for different non-browser environments, for example back-end programming. All browsers support it and it provides the users with flexibility and dynamism over there web pages without the need to send requests to server. Advantage of JavaScript is that there are many available libraries that can provide functionality or extend current one [27].

The above mentioned programming languages are used for front-end development, except JavaScript that can also be used for back-end. For implementing the logic of the application on the server-side, languages such as C# [10], PHP [39] and Java [40], can be used. When developing web applications, the developers usually choose to use the ones they are experience with or the ones that will suits them best in their project. Figure 5 visualize the web application programming structure.

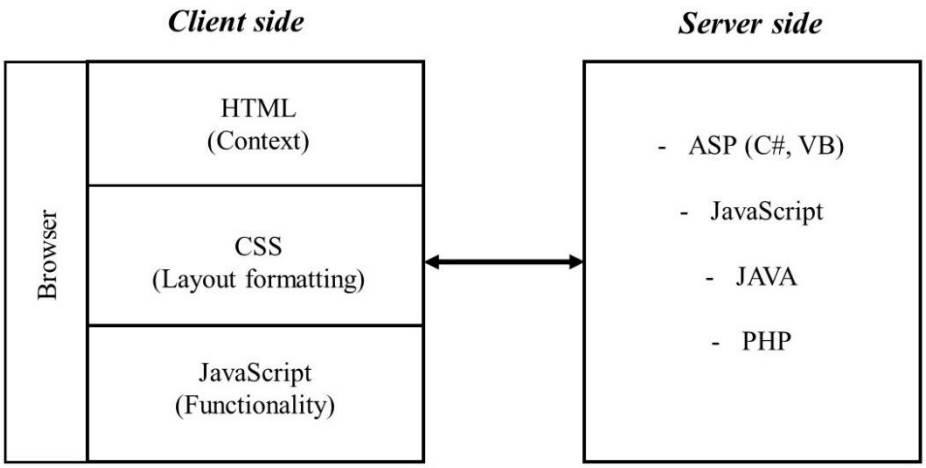

Figure 5. *Web application programming structure overview*

From the most abstract point of view, web applications consist of three layers – presentation, business, and data [41] [42]. These layers are illustrated in Figure 6. The presentation layer interacts with the users. It contains the presentation logic components, the user interface components and it provides the connection with the business layer. The business layer refers to the actual logic of the system. Also this layer may include parts that others can use as service interface. The last one is the data layer that gives the access to the data within the server as well as outside through some services similar to the ones in the business layer. The amount of the layers may vary according to the general purpose they are needed for.

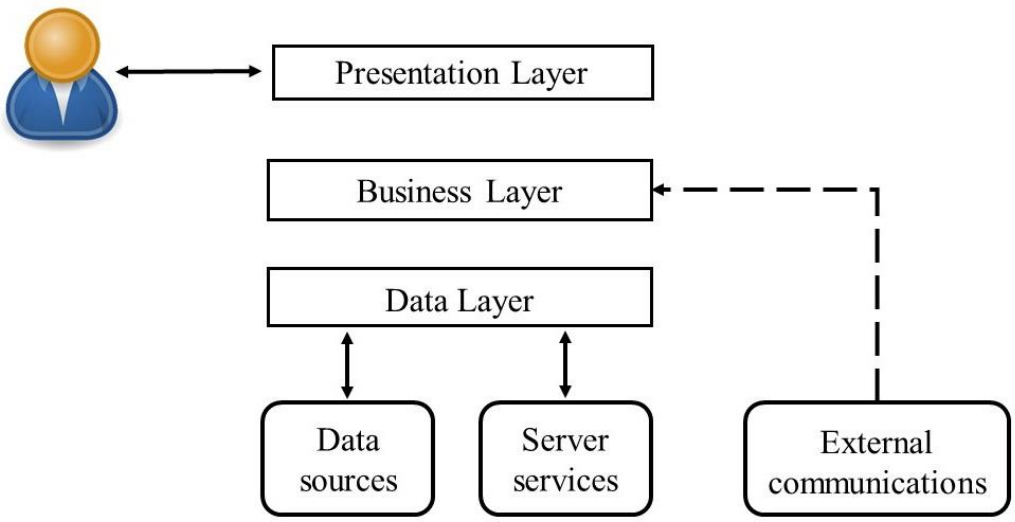

Figure 6. *Web application layers*

Web applications are mainly divided in to two types - static and dynamic. The first one consist of HTML files where each page is presented by a physical file and if they have the same elements, then these elements appear in the code for each page. In contrast are dynamic web server technologies, such as ASP, where pages are a build dynamically. Figure 7 presents the difference between static and dynamic pages.

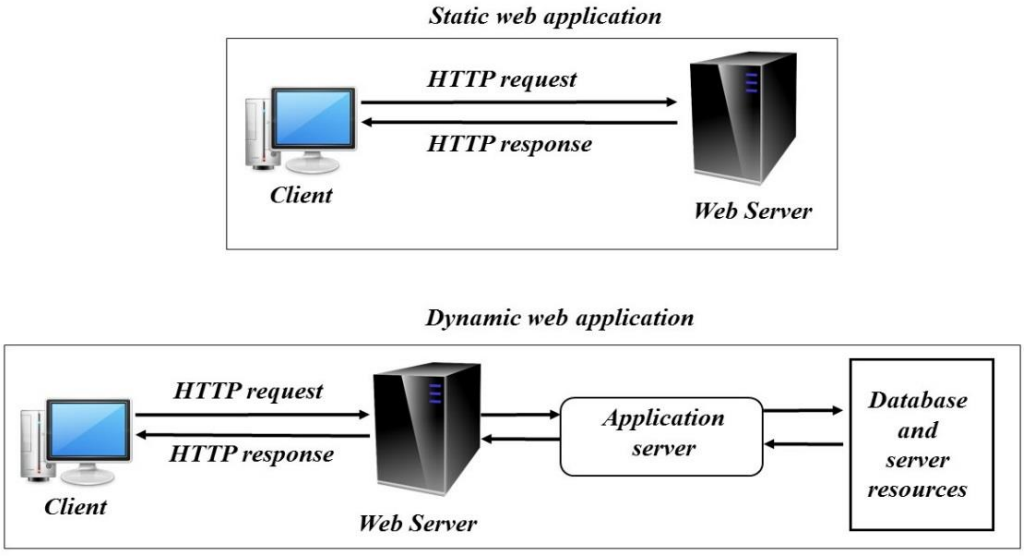

Figure 7. *Static versus Dynamic website*

With static web applications, the client makes a request for a resource by sending a request to the server. Then the server responds by locating the files on its local storage and sending them back to the client. In dynamic environment an application server software is installed on the same server where is installed the web server. The application server

has the ability to communicate with the databases and the other resources on the server. In this environment the client makes a request. Then the web server talks with application server, which is talking to the database and the resources. After that it is up to the web server to construct an HTML format in response, which is being sent to the client. The important difference compared to the static one is that the client is not aware of the application server, hence, does not require additional software. All that clients do is sending a request for html and then receiving an html.

Web applications often require some kind of authentication from its users, because of different reasons such as keeping track on users' progress or restricting access [44]. As already mentioned web applications are accessed via HTTP and this protocol is stateless [45]. This means that HTTP do not demand from the server to store information or status about the users during multiple requests and once the transaction is completed the connection between the client and the server is lost. A solution to this issue should be the use of a mechanism that will link each response and requests. There are different methods such as the use of HTTP cookies or session cookies. Both examples are almost similar. The main difference is that the first one have expiration date and they are stored on the client machine. Once they expire, they are being deleted from the client. The latter one, on the other hand, does not have expiration period and they are stored on the server. Once the user closes the browser the session cookies are being deleted [45]. Session cookies are usually created during the process of authentication. After that they are attach to each HTTP request and act as a temporary replacement of the user's credentials.

These two mechanism for authentication can also be used for storing and passing data between pages which is a widely used approach with web applications. Both have their advantages and disadvantages. For example, session cookies can store more data compared to HTTP cookies. Moreover, the session cookies reduce the bandwidth usage, because they pass only a reference when the page loads, while HTTP cookies have to pass all the data to the server first. However, the client-side cookies have advantage towards the serve-side ones. They last longer, even after the browser is closed, which is useful if one wants to store authentication data or some temporary settings.

4.1 User interface design

Joel Spolsky says "A user interface is well designed when the program behaves exactly how the user thought it would." [32]. There are different ways of describing what actually user interface design is. Some sources state that it is the design of machines, computers, software applications or websites, where the focus is aimed at the user's experience and interactions. Others explain it as a process where the idea of how a machine works is being interpreted into the way humans think, or as a process of creating a visual language that allows the users to interact with application. However, all these are correct and it can be concluded that user interface design combines them all [33].

There exists many different user interface design methodologies and there is no one that is perfect by itself. In order to achieve good interface multiple design approaches have to be applied. The two most used methods are the parallel and the iterative design [34] [35]. Although, they seems as different types, both design patterns start from version 0 concept, that describes the interface purposes, and often parallel design method is followed by iterative. Typical for parallel design process, as the name hints, is that during the initial step one or more designer (depends on the team), produce several multiple designs at the same time. Usually the preferred amount is between 3 and 5 alternative designs [35]. These designs do not require to consist of all the features and they had to be created quickly without sparing too much expenses on them. A good approach in this case is paper prototyping, where the designs are drawn on papers and they provides a rough wireframes on the interface. Once they are ready, it is time to put them to a user testing. The prototypes are given to users, where one user evaluate several versions. After this process is done the results are taken into account with all the prototypes and a single merged design is created. In the end the process continues with iterative design to improve more the merged design. Figure 8 provides a visual representation of the parallel design.

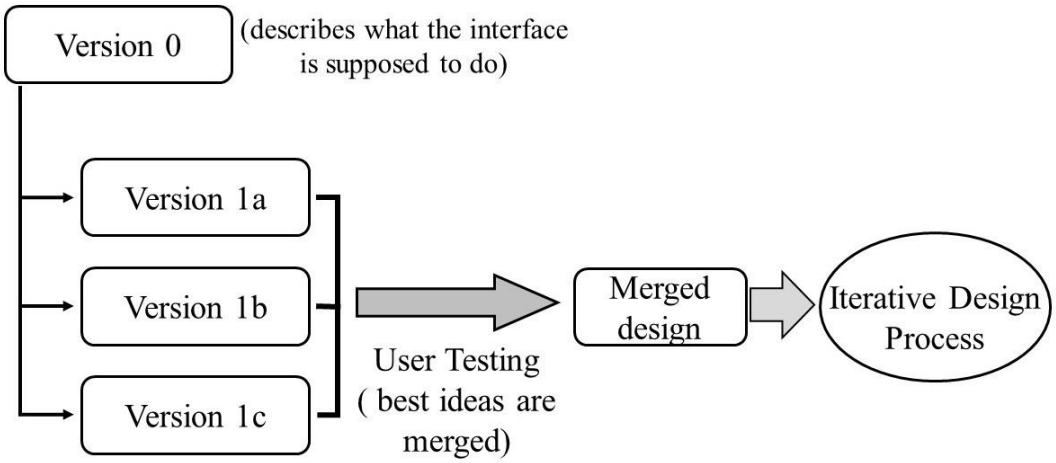

Figure 8. *Parallel design overview*

Let us now have a look at iterative design. It is cheap, simple and strong methodology. It is simple, because it provides a simple process model, cheap, because an iteration can be done for a short time, and strong, because you can do as many iteration as you can afford in means of budget [34]. The iteration design simply means that you create a design version, administrate usability evaluation, for instance user testing or heuristic evaluation, and then improve the next version based on the results from the testing. Figure 9 shows the cycle in iterative design.

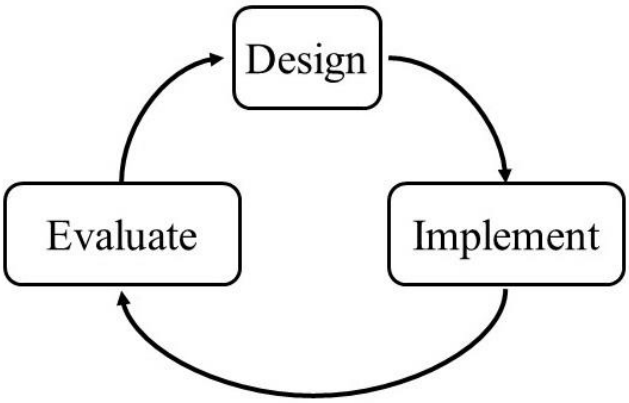

Figure 9. *Iterative design overview*

This look like the worst-case design due to the fact that the design flaws are found after all the three stages are executed. However it is one of the most common design methods, because the evaluation takes place on the market, where the actual users are giving feedback to the designers. This design method aims for short iterations, because the requirements are constantly evolving. During the implementation of the current design flaw fixes, the users are providing new set of requirements for the next iteration. This way the design and functional fixes are being created shortly before being implemented, which leads to a tight feedback loop, where a quick user testing can be performed and the new flaws can be fixed during the next iterations.

Important stage in the process of user interface design is the evaluation, during which the design is being examined and measured in terms of effectiveness, efficiency and satisfaction [48]. There exists different evaluations approaches, such as heuristic evaluation, software guidelines, cognitive walkthroughs, and usability testing [49]. As it was already mentioned, heuristic evaluation is method that uses principles called heuristics to identify usability problems. This method is usually performed by UI specialist, who have the task to study the interface in depth and look for properties that may lead to usability problems. Usability testing method is performed under real-world conditions, where the evaluation takes place during the use of the user interface [49]. However, this method is expensive and time-consuming. Another method is the software guidelines, where a set of guidelines is published and used during the evaluation. These guidelines provide recommendations about the user interface design content and this evaluation method can be performed even from people that are not UI specialists [49]. Cognitive walkthroughs is another method for evaluation of the user interface design. This method can be performed by the developers, who examine the interface in terms of typical users' usage. Then the results are compared with the users' goals and knowledge and the differences are noted as problems [49]. A way to compare the above-mentioned evaluation methods is to perform them over the same user interface design. However, there is no single best method and each of them has its advantages and disadvantages (Table 1). [49]

Table 1. *Comparison between different UI design evaluation methods*

	Advantages	**Disadvantages**
Heuristic evaluation	A method that identifies a big variety of UI design problems both serious and low-priority. Also it is a low cost method.	Requires UI knowledge and expertise. More than one evaluator is required.
Usability testing	Method that identifies serious problems and avoid the low-priority ones.	It is an expensive, time consuming method that require UI expertise
Software guidelines	Method that identifies recurring and general problems. It can be performed by non UI specialists.	Possibility to miss severe problems
Cognitive walkthroughs	Can be used by developers and it helps defining the users' goals	Time consuming method which requires task definition methodology. Possibility of missing recurring problems.

4.2 Thick and Thin client

When users are performing tasks by using an application or processing some data, it is not always mandatory that the results has to be stored on their local machines. It has become very common to use the term *client-server architecture* [22]. This kind of architecture refers to a network where client is communicating with a server in order to execute some task or service. In this order, we can define that a client is an application that run on the user's workstation and it is dependent by the server to perform operations. On the other hand, server is usually a powerful computer that is used to manage and perform different tasks and processes. When it comes to client-server architecture, the most important thing that has to be determent is if the client or the server will handle the workload. Therefore, there are two main types of architecture, namely thick and thin client.

Thin client usually refers to a stateless workstation without hard drive. It is can be referred as to a remote terminal. All the features with this kind of client are stored on a remote server. In many cases thin clients are browser-based applications. The most common

thing about them is that it is usually displayed in browsers, they can be cross platform and most of the processing is done by the server.

Thick or fat client refers to architecture that provides rich functionality as well as it is independent from the server, in contrast to thick client that is dependent [22].Thick client also is a machine that has its own storage device where to store and process data, since the most workload is done on the local machine.

Typically client-server applications usually consist of several layers, namely user interface, application logic, business logic, data access and external communication. The first one refers to what the users are presented with and interact with. The next layer concerns the communication between the users and the computers. The business layer include the logic on manipulation of the data. The data layer is concern with reading and writing the data to storage locations. And the external communication layer concerns the outside systems that take part in exchanging information resources. According to where these layers are utilized, it can be determent what kind of architecture is present (Figure 10).

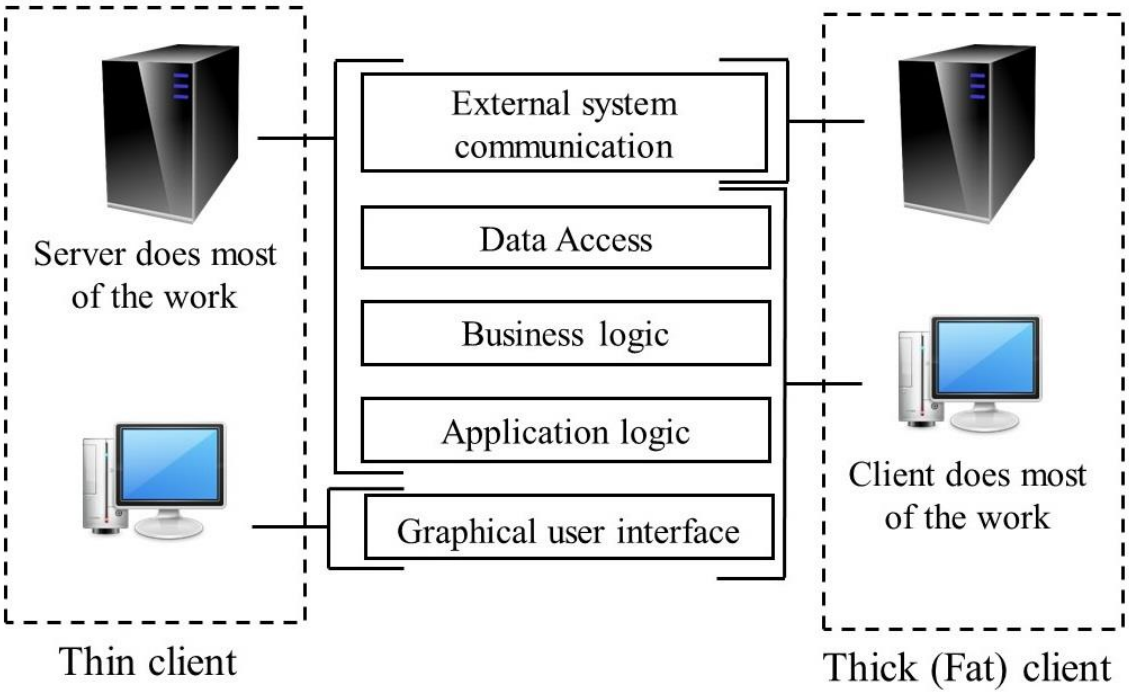

Figure 10. *Thick client vs. thin client responsibilities*

Even these two architectures are quite similar, because they both run on the user's machine and exchange data with the server over the network, they also have many differences. Some of these differences between the two architectures can be observed below (Table 2) [22] [23].

Table 2. *Comparison between thick and thin client architecture*

Thin client	Thick client
Application is easy to deploy and does not require additional installation	Application has to be deployed on every machine
Data is validated by the server after captured from the client	Data is validated by the client itself
If there is server timeout, the client cannot perform processes. Server communication should be present at all time	Continues communication is not required.
Portability. Application is probably on the server and can be accessed from any location	Application can be accessed only on the machine that it has been installed
Ability to use old equipment at client side.	Reduce workload on the server
Reduce security threats	Increased security issues

Because of all this advantages and disadvantages of each of these two kinds of applications, nowadays, a new kind has been created. It acts as a bridge between thin and thick client and it is called smart or rich client [23]. The tendency is to move from the traditional client-server architecture to web-based one and to provide the application over the web HTTP connection [24]. This kind of client applications can work both online and offline as well as automatically updated without the need of the user. Moreover, due to the fact they are web-based, they supports various amount of platforms and languages. Smart clients also provide rich graphical user interface similar to a common desktop application.

5. BUILDING UI FOR OPENNLP PART-OF SPEECH TAGGER

The GUI solution that is presented in this chapter aims to simplify and guide the working process of the users when they are interacting with natural languages processing engines and their tasks. Based on the criteria for portability and accessibility, it was concluded that the most suitable choice for the project will be the development of a web-based application. Hence, the application will be separated into two parts, namely client-side (front-end) and server-side (back-end). The client in this case will be the browser through which the users will access the application and the server will be used to process the data and interact with the natural languages processing engine. Therefore, it can be concluded that this application can be classified as a thin client, due to the fact that the most workload will be done by the server.

This chapter starts with an explanation on how the requirements for this project has evolved during the application development. Then an overview of the used technologies will be provided and explanation about the choices that have been made. After that, the GUI structure will be presented followed by the flexibility solution of the user interface. Then there is a section that describes how the application deals with the users and how the files are managed on the server. After that, there are several sections that explain the user interface and some functional matters of every stage in the application.

5.1 Requirements analysis

In the process of application development not all the requirements stay fixed during the whole process. Sometimes it happens that some requirements has to be dropped, others has to be adopted. There are different reasons for this to happen, such as unforeseen problems that can occur during the development, missing stakeholders or just the customer has come up with some new ideas on their mind during the working process.

In the current thesis the stated set of requirements corresponds to the ones set at the point when the application was already deployed. However, these requirements evolved from the initial ones due to some of reasons that will be explained below. Let us now observe the requirements and see the steps that resulted the current set.

In the initial staged of web development the first thing that had to be decided was the environment and the programming language that will be used. Since application is web-based it was obvious that HTML, CSS and JavaScript will be the ones that will take place for the client-side development. However, for the server-side there are many possibilities.

Due to our stakeholders' requirements, it was set that .NET environment and C# programming language should be used. The reason for this choice was that the server, where the application was targeted for deployment, uses Windows based operating system [11], such as Microsoft Windows Server. During the developing process these requirements stayed fixed and they didn't face any changes.

The next major requirement concerned the natural language processing engines. The initial idea was to implement the part-of-speech tagging task by using two different engines, namely OpenNLP [1] and SharpNLP [25]. However, this requirement faced an unforeseen problem that was determined during the developing process. During the initial research it seemed that OpenNLP engine will provide us with a challenge, due to the fact it was developed using Java programming languages [40], while SharpNLP integration seemed easier, due to the reason it uses the same framework as our application. However, during more extensive research it was concluded that the integration of SharpNLP engine will not be possible, because of the lack of documentation about its implementation and use. Moreover, the project was abandoned and there was not any support available. Therefore, as a result it was set that only OpenNLP part-of-speech tagging task has to be implemented.

The last major requirement focuses on the task process flow. As already mentioned, natural language processing tasks consist of stages. The main ones that usually take place are the creation of the language model and its evaluation for the current task. In order to create a part-of-speech model properly formatted data, like the one shown earlier, has to provided. Since the users do not have such kind data another step had to be executed beforehand, namely preprocessing the files. Therefore, an initial requirement was to create a stage that will handle this preprocessing step. Later in the developing process the users have decided that they would like to have a stage where they would be able to use the tagger on various inputs if they are satisfied with the results from the testing stage. As a result the process flow ended with four stages show in Figure 11.

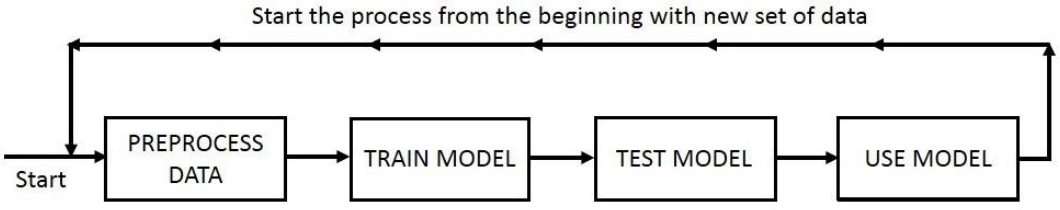

Figure 11. *Process flow overview*

In the first stage the users preprocess their data by filtering it and converting it to the necessary format for training. Then in the train stage the input data is used by the part-of-speech tagging task to train a model for the specific languages, followed by the testing stage where the accuracy of the model is determent based on its reaction over some data.

And the last stage is meant for industrial use, where the users are able to use the model with whatever data they wish to be tagged.

An abstract view of the application can be seen in Figure 12, where the parts of the application that interact with the users are presented as stages. They are based on the approach required for natural language processing models that was shown earlier.

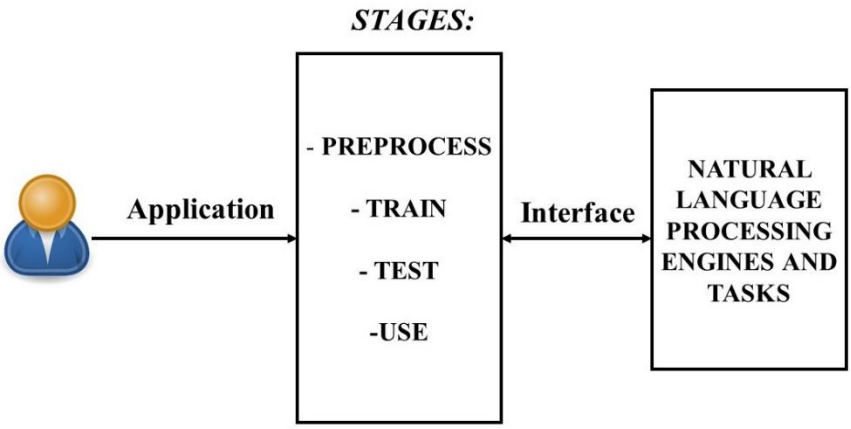

Figure 12. *Cooperation between the elements in the environment*

The stages has the task to handle and verify the information in several ways and to ensure that the engine will receive a proper formatted data. Although, the data provided from the user is not always properly formatted and needs manipulations, it has at least to be correctly tagged with its part-of-speech tags.

5.2 Implementation technologies

According to the technical specifications we were entitled to use C# and .NET framework, hence, we agreed with company that for this project the most suitable solution will be delivered by using ASP.NET. ASP is an object-oriented platform that allows to select among a variety of programming languages, such as C# or Visual Basic, to create dynamic web-based application, where the client makes a HTTP request and the server responds by dynamically constructing an html page and send it to the client.

ASP.NET provides three frameworks for web development, namely ASP.NET Web Forms, MVC and Web Pages [9]. Web Forms are used to create dynamic content web application, as the pages run as programs on web server when requested. While a page is running it can perform any task required by the web site. As a result a page is dynamically produced and send to the browser. The ASP.NET Web Forms file consist of tightly coupled view with the code behind. The ASP.NET MVC is used to develop large scale web applications. This model consist of three parts- model, view and controller. The model is the part that handles the logic for the application, the view handles how the data is displayed, and the controller handles the user interaction. MVC is an alternative to Web

Forms and it provides features like lightweight, testable framework, and also provides independence between the view and the logic of the application. The ASP.NET Web Pages are used to target developers that desire to create a simple web story. In these pages there is the HTML code and also the server-based one that is used to dynamically control the rendering process of the page [9].

In this thesis ASP.NET Web Forms were used with addition of the "master pages" concept that provides a template-based approach. There were few reason due to which it was decide to use this approach instead of MVC. The first reason is that the team for the project consist of two people – myself and Grigorij Ljubin Saveski [4]. Moreover, both of us didn't have enough practical experience with MVC and compared to Web Forms, it requires more programming and deeper knowledge of how this architecture works. Moreover, ASP.NET Web Forms provides more control over the client from the server since the view and the controller are one and the same. Also, the natural language processing tasks required specific stages, as well as control. Therefore, in case of expansion it will be simpler by adding the required stage with its functionality. In addition to the web forms we used the common for web development HTML, CSS and JavaScript languages.

As far as the engine is concerned, we had to find a way to connect the OpenNLP interface with ours. In many cases it is assumed that Java and .NET are mutually exclusive technologies. However, thanks to IKVM.NET [43] tool were able to achieve that. This tool has allowed us to compile the OpenNLP libraries down to DLL files. After that we had just to reference these files directly to our application and used them as they were .NET objects.

5.3 Structure of the GUI

Figure 13 shows the structure of the application. As mentioned earlier, this web application uses the "master-page" template approach. This means that all the common layout and functionality of the pages can be implemented only in the master page. By doing this the repeated code in the different web forms is being reduced and centralized at only one place, which makes it easy to alter if needed and will affect the final result among all pages. Moreover, the master page and the content pages cannot exist separately. In order for the users to access a specific content page, they send a request to the web server. Then the web server merge both master and content page and produce one page with combined layout.

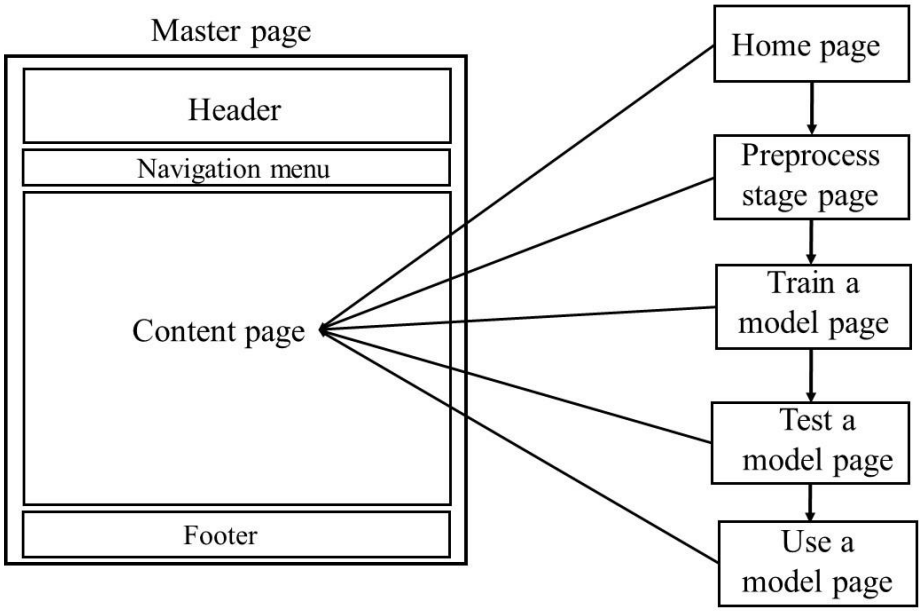

Figure 13. *Structure of the application*

The idea with the master page templates is that they consist of two parts, namely the master page content and a placeholders. The first one encapsulate all the common elements that should be visible, such as headers, menus and footers. Also in this section all the common style files and JavaScript files can be places. For example, since in our application uses jQuery, it is hardcoded in this page and there is no need to add it in any other file. The master page placeholders defines the place where the content of the actual pages should be placed. Every master page can contain as many as needed. This placeholders can be overridden by a content pages. This is very convenient, because every page can be placed in different file. In our application we have five content pages – home, preprocessing files, training, testing and using model. On the top of every file it is marked where the master page physical location is. After that they contain controls named "Content" that are linked to the master page placeholders. These controls enclose the specific content for every page. The graphical design of every content page consist of two things. The left side is the working area, where all the controls and messages appear, and the on the on the right side there is a section, which includes a brief manual what is the current page for and tutorial explaining how to use it. Moreover, in the structure of the application, there is a separate web form page that is not connected to the master page. Its interfaces is simple and consist only from images that visually show the steps for the whole process flow. This page was created as users' request, because usually it is easier for the users to see something than to read manuals.

5.4 Flexibility of the application

The process flow of part-of-speech tagging task in the most basic scenario can be considered straightforward (Figure 14). The first overview of the figure gives the idea that process is complexed. However, the bold rectangles represents the stages and all the starting arrows refers to the entry points. The bold oval shapes refers to the output files from the stages that are requirements for other stages and all other shapes are additional requirements or outputs. For instance, in training stage it is required to have a natural language engine and to provide the proper preprocess data file in order train a language model as an output.

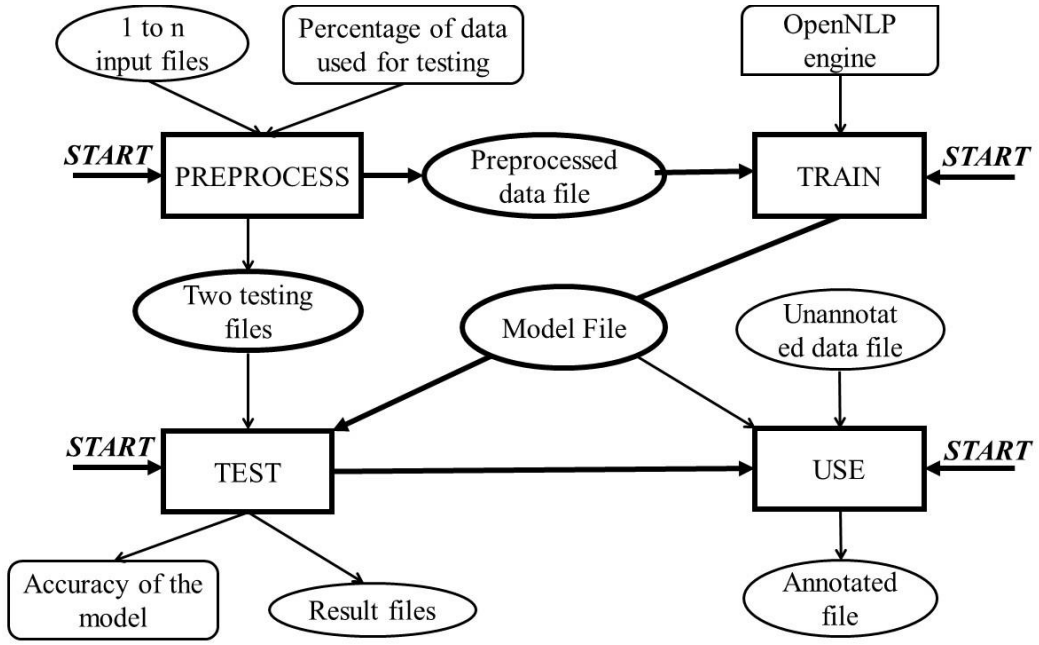

Figure 14. *Process flow stages with their inputs and outputs*

As you can see in Figure 14, the process is direct when it comes to the user to preprocess its own data, then train a languages model, evaluate the precision of the this model and in the end to use it. However, this process cannot always follow this order of execution. Due to an external reasons or just the need from the user side, this process might need to be interrupted at some point. Therefore, some flexibility is required that will provide multiple entry and exit points in the process stages. By using these access points, the users will be able to start or stop at any stage or just continue with the following one.

Possible situation with interrupting the process flow can occur if the user creates a language model, but during the testing stage it appears that the results are not satisfying. In this case the process is being paused and redirected back to the training stages where another old model is loaded into the engine and then tested with the current testing data. Then if the expected results are met, the process can continue to the next stage where the model will be used to tag wider variety of data.

The access points to every stages are handled by the graphical user interface of the application. However, with such flexibility a set of restrictions are required in order to control the actions of the user. This is also needed so the application to follow what stage was accessed and what were the conditions that occurred. As you can see in Figure 14, in order to execute a stage, users are required to provide the necessary input files. Furthermore, after the stage is executed, the applications marks this stage by saving some data that is required during the following stages. This data refers to the physical location of the output files on the server (the bold outline oval shapes in Figure 14). This way the process of creation and evaluation is being simplified, because these files can only be expected as results at a particular stage in the process.

Figure 15 provides a graphical overview of the solution to the flexibility problem explained above. As you can see this is achieve with a content page named "Home". It acts as starting point for the whole process flow, as well as an entry point to all the other stages.

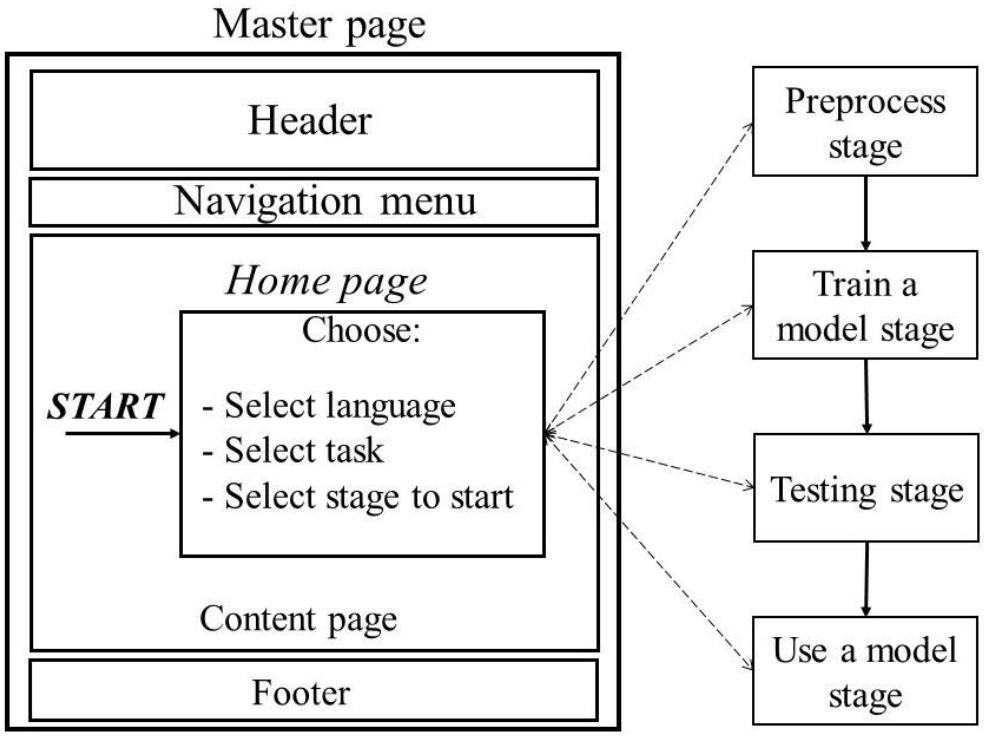

Figure 15. *GUI solution for the flexibility problem (Home content page)*

This page is the default one that is opened at the initial start of the application. To proceed to a specific stage, the users are required to provide some data that is being used in the process, namely the language, the task type and from which stage they want to start. By providing this data the users can continue to the chosen stage. However if they do not select one of these option, there are error messages that will be shown beneath these fields.

During the process, if the users desire to start their task from another stage, they are required to go back to the home page. This step backwards was chosen as step to control the actions of the user as well as an attempt to keep the files in some specific order that will be explained later.

5.5 Users' accessibility and files management

In order to provide some level of control on the users, there were several choices that have been made. The first one concerns the users that access the application in order to prevent anonymous entries. In Lionbridge, every individual user has its own domain account to login in the workstation onsite. Moreover, to use the server where the application is deployed, the users have to use the same credentials even if they access it remotely. Therefore, it was decided to use these domain accounts in order to authenticate in our application too. In the configuration file it was set that no anonymous access is allowed and the application uses windows authentication method to determine who is the user. When a user tries to access the application, his credentials are being check. If they appear in the corporate domain, then the user is allowed to use the web application. However if the user tries to access the application from a different machine then he will be prompt to enter corporate credentials in order to log in into the application. These usernames are used later when the folder structure is being created.

The second choice was done in order to follow the state of the stages in process flow. There are different ways to pass parameters between pages, such as URL variables, cookies or sessions. In our case we use session. There are a bit similar to cookies, again temporary storing data files, but the difference is that while cookies are stored on the client-side, sessions are stored on the server-side. Moreover, we chose sessions, because cookies can be deleted or alter on the client machine. Also the URL variables, can be also altered and the URL may lead to some unexpected behavior, like error. Compared to these two methods the session is being destroyed ones the browser is closed. The session variable have a lifetime during which the users can go to another websites and come back. However, the session in our case was set in way that if the users go back to the home page, their current one is deleted and a new one is created that is empty. This was done with the idea to prevent any free access to the different stages without the proper requirements to be fulfilled first, and particularly choosing the language and the task type which are vital for the folder structure.

The third decision that had to be taken was focused on how the files on the server will be managed. According to minor criteria from the users, they should be allowed to upload files from their local machines or to use files that are already on the server. The first problem that was faced here was about the file size. ASP.NET web application was deployed on Internet Information Server (IIS). They both have restrictions about the size of files that are being uploaded. And since our users may want to upload multiple files we had to alter these variables. The variable for ASP.NET is "*maxRequestLength*" and refers

to the size of the allowed input stream and it is measured in “KB”. The default value is 4 MB (4096 KB). The variable for IIS is named “*maxAllowedContentLength*”, it specifies the maximum length of the content in the request, calculate in bites and the default value is around 30 MB. Therefore, these two values were altered in the configuration file to allow the upload up to 1 GB combined size of files. Also, there is an error if users try to upload bigger files. A good idea for the files’ size might be to set an invisible variable that will store this data and by using JavaScript code to compare it with the calculated total size of all files added before the actual upload in order to prevent this error to occur.

The next step was to think of an understandable and easy to use file structure on the server that should be easily managed if accessed directly. After careful consideration and taking into account all the different stages and variables for the whole process flow, we came up with a structure that consists of four levels of folders that are nested as it can be seen in Figure 16.

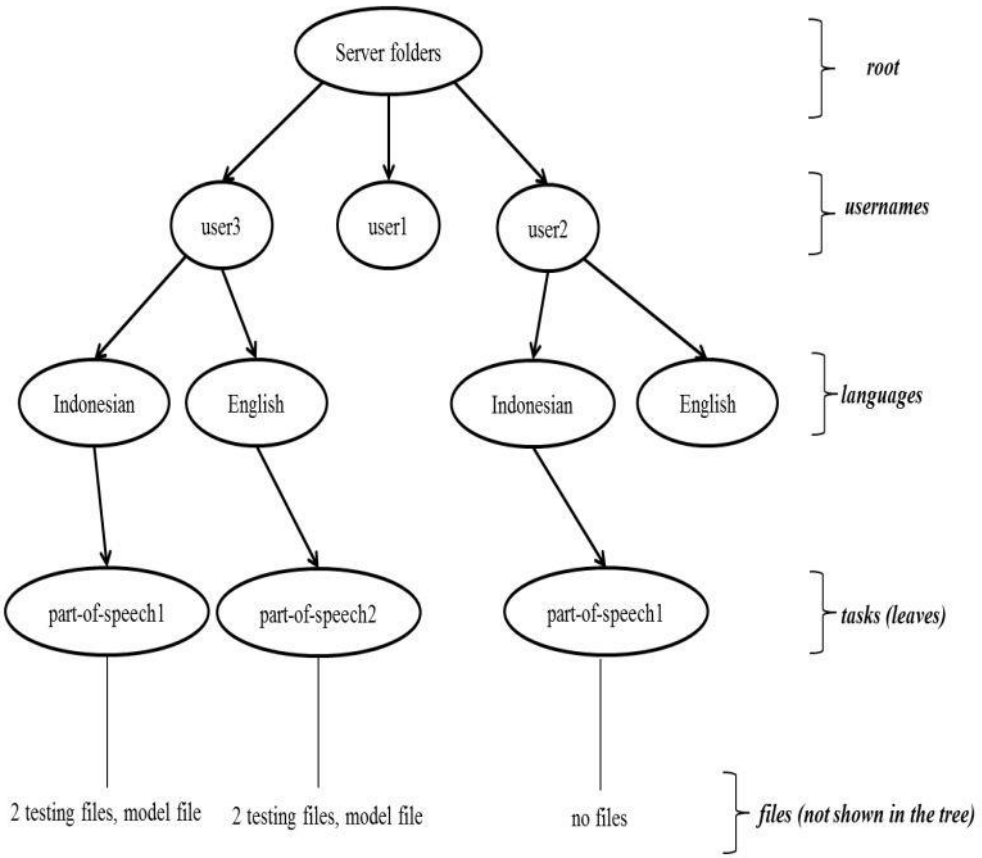

Figure 16. *Example of the storage folder structure based on reference [4]*

The top level is the server folder. It is usually deployed with application files. However, if it is missing there is no problem, because it will be created when the user starts the application and selects all the fields in the home page and then proceeds to the starting stage. This folder act as a default storage place. The username folder aims to structure the

work for every individual user and to make it easy to navigate through other users' folders. Initially a random string was used for naming. However, we decided that this was not a convenient approach, because the users are not going to remember all these characters. Hence, the name for this folder, as mentioned earlier, is taken by the application from the current workstation credentials that have been used to log in. Languages and task directories are required, because the amount of languages and tasks are greater than one. The names for both of them are acquired from data fields that has to be selected in the Home content page. The language folder usually contains only the name of the specified language for the task. On the other hand, a user can perform multiple tasks, or the same task many times for specific language. To prevent this folders to be overwritten, when a folder structure is created to the name of the task folder is appended a time stamp.

5.6 Preprocessing of input files

Preprocessing is the first stage in the process flow. It is a preliminary one and its purpose is to be used if the users do not have a properly formatted data that can be used directly for the purpose of training a part-of-speech model. On Figure 17 can be seen a brief overview of the content of the page. There are several sets of controls that are hidden and will be visible at a specific state of this stage. All this hiding and showing is achieved by using jQuery library. When the users initially start from this stage the controls that they can see are a checkbox for tag validation of the input files, buttons for selecting and uploading the input files, as well as a cancel button. There is also a table that will show the names of the selected files, their size and current status.

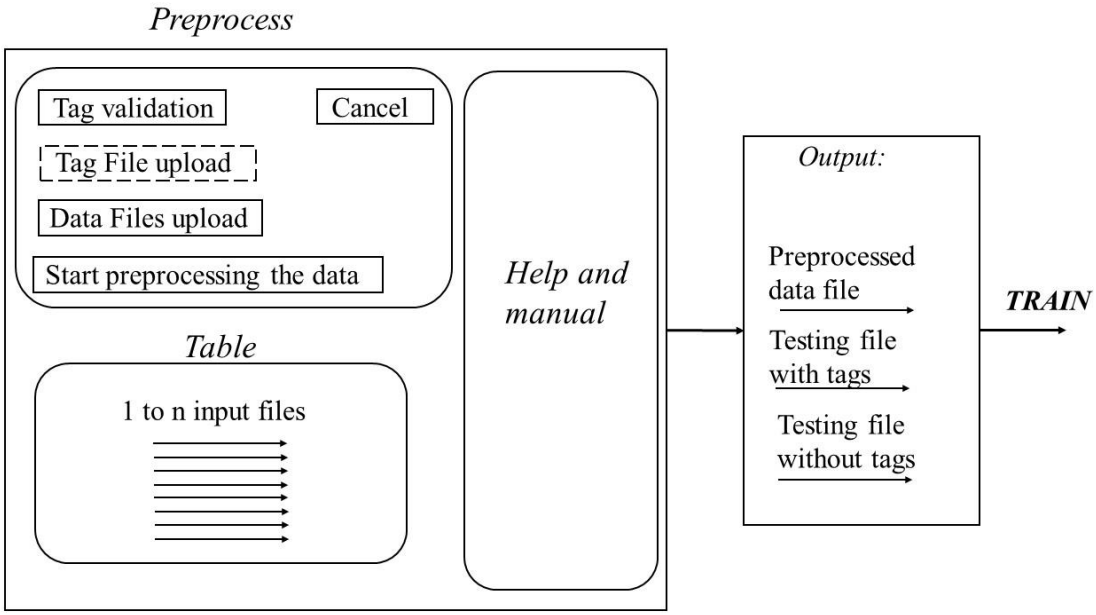

Figure 17. *Preprocessing stage overview*

The first thing that the users has to decide is whether they want to do tag validation over their input data. In case they want, they need to click the checkbox and the proper controls to upload a file with tags will appear and the others will become hidden. This tag file has to consist of all the available part-of speech tags for the specific language. After providing this file, it has to be uploaded and it will be stored on the server for later usage. This upload as well as the input data files upload and the preprocessing request, is handled with the use of ASP.NET Generic handler that act as target for the incoming HTTP requests.

As already mentioned, no matter whether tag validation is required or not, the default controls are the ones for uploading the input data files. Due to the fact these files do not have specific extension, we cannot filter them this way. The input data files are actually plain text files, maybe even without extension, that contain the data used for training and testing model. Once the files are selected they are being stored using JavaScript array and in the same time the name, size and the status of each of them is being printed in the table. If the users have forgotten to add files, they can use again the button and add the extra files.

Once all the files are selected, they need to be uploaded. To do this a jQuery Ajax () request is used to send a request to the generic handler. This process might take some time due to size of the files and the speed of the connection with the server. A status bar is shown, but its accuracy is not always very precise. In the generic handler first it is checked whether a folder for the files exists and if it does not it is being created. Also the tag validation is handle there. Once the process is complete a message is returned and the table status fields are being changed. If tag validation is chosen beforehand, the users will be notify if some file had problems with the tags and it will be discarded from the table and deleted from the server folder.

Once this step of the stage is done, the upload controls will hide and the preprocessing one will be shown. These set of controls consist of a button that starts the process and input field where the users are required to enter a number as percentage (Figure 17) that will be used in the division of the data for training and testing. At this stage the table contain only the files passed the validation, if there was such one. By pressing the button the process will start by sending another Ajax request to different generic handler. The software will create a temporary file in which all the data from the input files, after formatting, will be combined. Then based on the number entered as percentage this file will be divided into two parts. Here there are three possibilities for the percentage. The first one is to be “0%”, then all the data will be used to train the model. The next one is to “100%” then all the data will be used for testing a model. And the third case is if the number is between the above two values. Then the application determine the percent and randomly choose this amount of sentence and move them to another file. Let us have an example. If the user choose to have 20% testing data and he has total of 100 sentence. This means that from the temporary file 20 random sentence will be moved in a file for

testing and the rest will be saved in another that will be used for training. After this the process continues by stripping the tags from each sentence from the testing file and they are copied to another file that is marker "without tags". As a final result a message is shown to the user that the process has been successful. Also a link is provided if the users would like to download the output files which from this stage are training data and two files for testing, namely one with tags and one without Figure 17. Then the users have to use the button to proceed to the training stage or they can just leave the application and continue another time.

One button that was only mentioned was the cancel button. This button is visible during the whole stage and its purpose is to restart the stage and also delete all the uploaded files.

5.7 Training part-of-speech model

The training stage uses OpenNLP engine to create part-of-speech language model with the data provided from the previous stage. Figure 18 shows a visual representation of the stage. This however is the case when the user comes from previous stage. Then the process is direct and since all the requirements for this stage are fulfilled the user sees only two controls – the training button and the button to proceed to the testing stage, which is disabled before the training is done. After the training is done the users get a message that informs them whether the process was successful. They are also provided with a download link for the model file if they wish to download it as well as the train button is disabled and the button to proceed to the next stage is enable at this point. Drawback of this process is that once a model is trained with a set of data, another set cannot be added to the same model. The training data has to be merged and a new model has to be created.

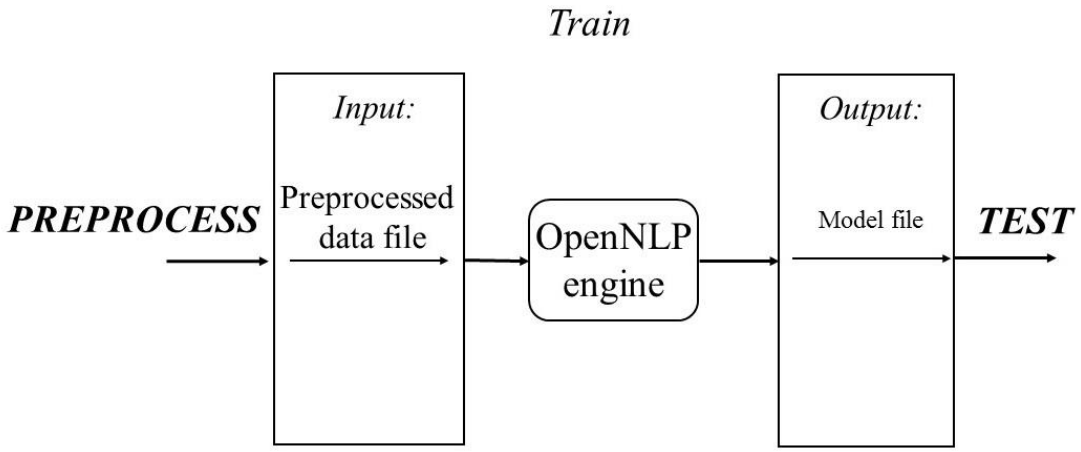

Figure 18. *Training stage overview*

However, the things are a bit different if the users' initial stage for the process flow is this one and they do not fulfil the requirements for the stage. For visual idea of the stage in

this state see Figure 19. In the current case when the users come directly to this stage their session values for this stage are missing. Therefore, the server renders a section that gives them two choices.

The first one is to upload a local file. In this case the interface provides them with a simple file upload control. They select the desired file and upload it to the server. After this action is over the location for this training file is saved in the session and the user is redirected to the same page. In this case the server reads that there is available training file and the section that is being rendered to the user is the one explain in the beginning of this chapter.

The second choice is to select a file that is already on the server. A tree view structure is shown with two controls beneath, namely the train and the proceed (Figure 19). The user expand the tree and observes all the folders available. The tree structure has checkboxes but with the help of JavaScript code, we ensure that only one option can be selected at a time. Moreover, if the user want to download the preprocessed data from the previous stage, they only need to click the name of the folder. After selecting the desired folder, the users have to press the training button and the model will be trained. In order to prevent the page from executing post back (reloading the page) this tree view structure with the controls is enclosed in an update control that executes an asynchronous request when the training button is clicked. This visualize a loading image that informs us for the training process and when it is over the train button is disabled and the button for the testing stage is enabled. Moreover, there is again a message for the status of the process as well as a download link to the model. If the user click the training button without choosing a folder, he is being alerted for that.

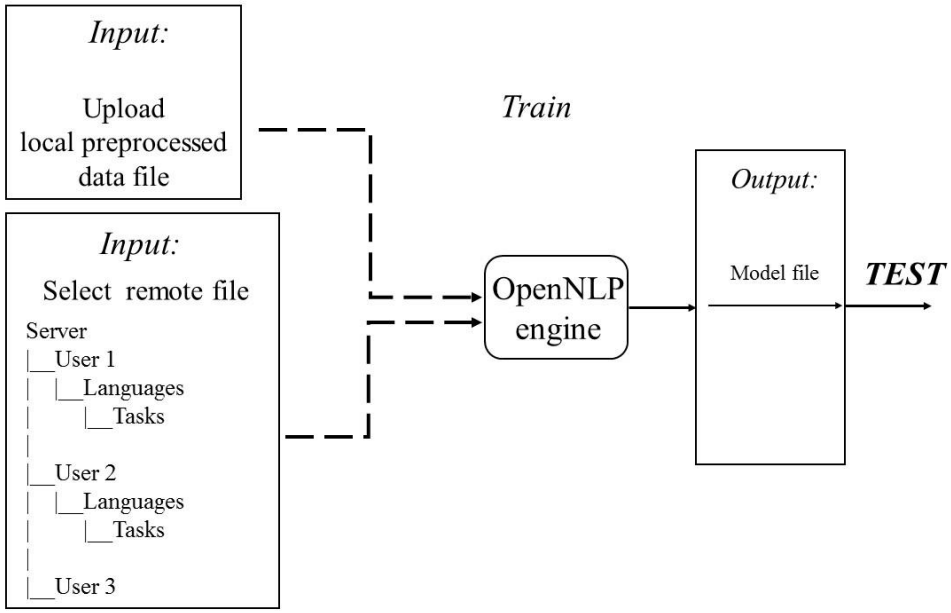

Figure 19. *Training stage as initial stage*

5.8 Testing a model

During the testing stage, the model, created in the previous stage, is being evaluated by its behavior when used over unannotated data. There are three files that are required for this stage as it can be seen in Figure 20. This stage has similar sections like the ones in the training stage. The sections here are three and the one needed is rendered according to the data stored in the session. If all the required files are present, then the user sees just two controls – one for testing and one to process to the next stage (disabled until testing is done). In this scenario the user directly test the model. The evaluation is done word-by-word. The part-of-speech tagger read the testing file without tags and predict an appropriate tag, then this prediction is compared with original tag from the testing file with tags. In the end a ratio is calculated between the correctly tagged words over the whole amount of words. When the process is completed the user is notified about the status of the stage. Also if the process is successful the evaluation result and all the files' locations are being printed on the screen. In the end of this stage there are two output files. The one contains the accuracy result from the testing and all the files' physical locations. The other contains all the predicted tags from the testing file without tags in one column and next to them in another column are all the correct tags. The latter file was created in case the users wants manually to observe where the part-of-speech tagger has made an error. And as in all other stages, there is a download link too.

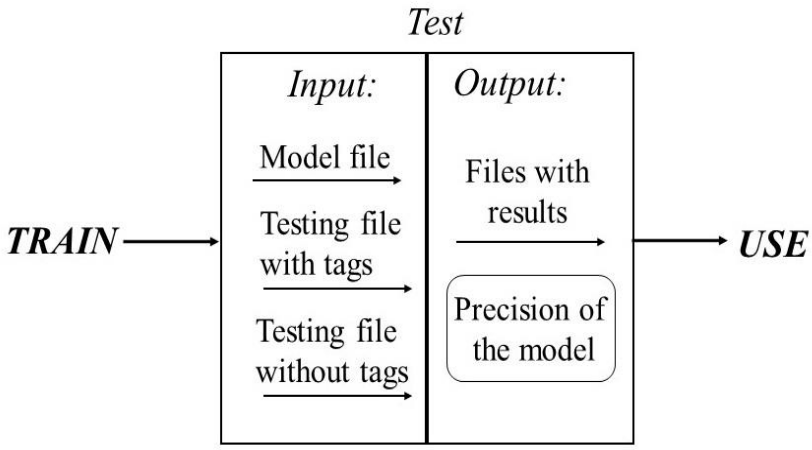

Figure 20. *Testing stage overview*

The other scenarios are if the training, the testing data or all three files is missing. Then there are similar sections like the one in the training stage. The users has to either choose from the local storage or from the files that are already on the server. If they select to upload their own files from the local machine, then the users will be redirected again to first scenario. If they decide to use files from the server, they will have to choose among the folders provided in the two tree view structures: one for language model and one for testing files.

5.9 Using a model

The goal of this stage is the model to be used with various inputs of unannotated data since the results from the testing stage were satisfactory. It is also called a production run stage. It is pretty similar to the testing of the model, when it comes to its functionality, with the only difference it does not calculate accuracy. The user interface is also quite sample. If the user's session has the location of a model file saved then the server will render the section where the user has to upload the file with unannotated data. If this data in the session is missing then the user, similarly to training and testing stage, has first to choose the model file either from the local machine or from the server. Visual representation of the task can be seen in Figure 21.

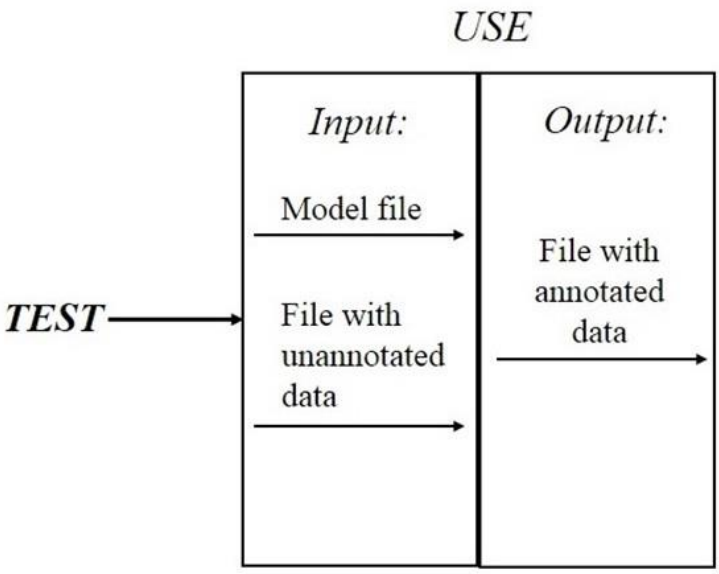

Figure 21. *Use stage overview*

The data file can be anything from random sentences to online articles and books, as long as the information is stored in a single text file. Then this file is also uploaded and the part-of-speech tagger process it. The result from this task is written in a file that contains the data tagged and the user can download it and observe it (Figure 21). At this state of the stage the user has two choices. Either to go to the home page and start with a new process flow or to add new data set that needs to be tagged.

6. EVALUATION

In Chapter 2.3 three different sets of criteria were set. The first one refers to the overall project success, the second set is used to evaluate the design of the application, and the third set presents technical criteria. The last one will not take place here, because it was already explained how it was taken into account during the implementation process. Let us first run-through the project criteria and observe how they were fulfilled during the developing process.

The first criterion concerns nontechnical users. The current web application reduce the memory load by releasing the users from remembering different commands and specifications. The interface replace all the technical details with visual controls, such as buttons, input fields, dropdown buttons, and the process requires mostly clicking and following steps. In addition to that the application handles everything automatically while following the process. For example, after preprocessing the training data and proceeding to the training stage, the application will provide the necessary data that follows and the users can directly train the model without the need to search for this data. Moreover, the application stores the data after every step on the server and provides a way to download it. This helps the users to reuse their data or to use others' data for their tasks.

The next criterion refers to the flexibility of the process flow. The current application provides a way for the users to start from any stage they desire, as well as to leave the application after every stage. The current flexibility in the application acts in a way that each start of a process is a separate instance with its own file structure. This way the users can track what they did during every run. A drawback of the current solution is that for one full process run there might be several folder structures, if the users start and stop during the flow. A good solution to improve the application, when it comes to this criteria, might be the use of some kind of configuration file for every instance which stores data such as completed stages, time, task name and files location. This way instead of starting a new process instance with its own folder structure every time, users can continue working on their old one. This also might reduce the amount of folder structures and make them more organized.

The third criterion involves the application expandability with other engines and their supported tasks. The web application, in its current stage, provides expandability with other engines and tasks as long as they support both folder and process flow structure. Every page in the application process flow correspond to some stage. Let assume that one wants to implement OpenNLP sentence detector task or another engines' task with the same stages. In this case only the logic for the current task has to be implemented. However, if a task requires additional stages, then new pages have to be created with both

design and logic. As far as the folder structure is concerned when adding more stages then a new folder structure can be created separate from the current or merged with it.

The next criterion concerns the portability. In order to observe the advantages and disadvantages of the current solution, let us compare it with the CLI method of accessing the natural languages processing engines. The CLI demands individual installation on every machine, whereas, the current solution require deployment only on a web server. Hence, it can be accessed from multiple machines simultaneously without any additional installations. Another feature is that CLI requires the files to be on the local machine. The current web application not only allows the use of local files, but it also enables the use of files that are on the server. A drawback for this criterion is the accessibility. Since the application is deployed on a remote server, there should be always an active connection in order to access to application and the files.

Indeed the last criterion is the accessibility of the application. If a user is using the engine through the CLI, then it has to be deployed on the current working station. In this situation if the user works on different machines then they all need to have the engine and the required software installed. In contrast is the web application, presented as a solution in this thesis. Its portability provides remote access from various locations and from different machines. However, a drawback of the application is a situation when there appears to be connection issues, such as the server is down, or the server is overloaded. Then the use of the application will be slow or will not be possible at all. The details explained above for the project criteria are summarized in Table 3.

Table 3. Criteria and their achievement

Criteria	**Fulfilment**
No technical knowledge	Everything is being handled automatically from the application during the process flow
Process flow flexibility	Multiple entry and exit points were provided for every stage of the process flow; data is automatically saved;
Expandability	New engines and tasks can be added as far as they support both the folder and system structure
Portability	Getting rid of installation can be done through a web-based application.
Accessibility	The interface is a web application , which allows its accessibility from various locations

In addition to these criteria let us now see how the current web interface design corresponds to Nielsen's 10 Usability Heuristics for User Interface Design. These heuristics are not specific usability rules, they refer more to guidelines and can be taken into account during the user interface design process. The heuristic evaluation helps to identify usability problems in the user interface design.

The first heuristic refers to the visibility of system status. The current application provides different ways of feedback among the different stages. An example that compiles with this heuristic is when uploading files there is a status bar during that process and after the process is done the status of the files in the table is being changed. This way the users are informed about the process status.

The second principle refers to matching the system with the real world. The application interface is simple and it consists only the necessary controls. Its stages are named based on the step for the task and follow the convention of the engine's task. Moreover, the controls are named in similar matter, so they will not mislead the users about their purpose.

The next heuristic concerns the user control and freedom. The application provides multiple entry and exit point, but there are also some restrictions for the process flow. Due to the application file handling the data is passed among the different pages. If wrong data is provided the system will crash with an error and the users will have to restart the current stage. This heuristic cannot be fully covered due to the nature of the process flow. For example, if wrong data is provided for training a model, then the engine will crash. In this case the stage need to be restarted, because an undo action of that page will navigate the user to the previous one.

The forth principle refers to the consistency of the interface. The layout of the application is fairly similar among all the pages. They all contain the working area and the manual information sections. Moreover, all the elements with similar functionality use the same layout. For example, all the controls for uploading files in the different stages have the same visual layout, only the names are different due to their purpose.

The following principle targets the prevention of errors. The web application provides messages for most of the controls if someone tries to skip using them or use them without fulfilling the requirements. Moreover, some of the controls are even disabled to prevent their use. For example, if user tries to upload files, without selecting any, then this button will be disabled until files are selected. However, there is drawback, because incorrect files can be selected. The problem appears, because there is no predefined file format. For example, the files that can be used must contain plain textual data, but they might have extension, such as ".txt" or ".log", or they may lack such one.

The sixth principle refers to the memory load and every control in the application refers to some action. Moreover, all the elements are named accordingly and instructions are

provided to prevent any errors and reduce to amount of information that the users need to remember.

The following heuristic focuses on the flexibility and the efficiency. The current application facilitates and guides the workflow. However, not all stages are efficient and flexible. The only stage that complies with the heuristic is the "use" stage. It allows the users to iterate that stage as many times as need with different data. However, things are different with the other stages. For example, in the preprocessing stage it is possible only ones to upload data. A possible improvement here will be to provide the interface with a way to preprocess data even if the stage is at its end. This can be achieved by adding the data and using the same variables to format and then merge it with the rest.

The next principle refers to minimalistic design. The web application implements only the most needed functionality, without any additional visual elements that will make the process slow and will affect the layout. Moreover, the interface provides simple design, which meets the users' requirements.

The ninth principle refers to error messages that tend to explain an error or unexpected behavior. This heuristic is also not fully covered. During the evaluation, all the errors that the users came in touch with were observe as far as they were part of the web application interface. However, due to the fact the OpenNLP API is used, not all errors provided by the engine were covered. Some of the engine's errors that were met and compiles with this heuristic concern the state of the input data.

The last heuristic concerns the documentation. According to the users' desired every stage has its own section that provides a brief explanation of its work and a short manual what needs to be done. Moreover, the application has a separate page which provides a visual manual about the steps needed in order to execute each stage.

After explaining how the application meet these heuristics, it can be concluded that there are several of them can classified as strengths. These principles are visibility, match with real world, consistency, memory, aesthetics, and help documentation. Although I have classified them as strengths, they should be still considered as improvement areas. The rest of the heuristics can be classified as weaknesses of the interface design that need improvement. Table 4 summarize the details explained above.

Table 4. *Comparison of the application with Nielson's heuristics*

Usability Heuristics	**Web application design**
Visibility of system status	The user interface contains only the necessary controls and in every view there is element that represents the current status.

Match between system and the real world	We don't use icons, but all the buttons were named with an understandable names.
User control and freedom	Users can enter and exit the process flow, but there is no undo and redo functionality
Consistency and standards	The user interface reuses the same visual elements and layouts
Error prevention	All the input fields, checkboxes, radio buttons and upload buttons are provided with error messages if they are skipped or tried to be used without data.
Recognition rather than recall	For all actions there are controls provided, as well as instructions for some of them in order to prevent even errors.
Flexibility and efficiency of use	The process provides a guided work flow, but now all stages provide flexible and efficient use.
Aesthetic and minimalist design	Only the most essential functionality takes place.
Help users recognize, diagnose, and recover from errors	All the errors gives information what went wrong with the application. However, not all errors were covered as far as the OpenNLP engine is concerned.
Help and documentation	Every individual content page provides a manual on how to be used as well as there is separate page that has the whole process described by pictures.

As a result, it can be stated that both the project and design criteria were carefully taken into consideration and were used as guidelines during the development process of the application.

7. SUMMARY

The resulting application, explained in chapter 5, grants access to the part-of-speech tagging task, supported by OpenNLP natural language processing engine, without any complications from the targeted group of non-technical users, in our case people with linguistics background. The system allows them to preprocess their data into the required input data format, create a part-of-speech language model by training it with this data, evaluate the accuracy of model and last to use the tagger for their purposes.

In addition to that, the process flow of the application, presented in chapter 5.4, contains multiple entry and exit points. These points give the users the freedom to enter the process flow and start from any of the stages. The use of these points, however, rely on the fulfilment of the requirements for the specific stage. Moreover, the results from every stage are stored automatically on the server, which gives the freedom of the users to quit the system at any stage they desire and later on to continue from the stage they have paused.

Moreover, the system should be able to expand with new engines and their tasks. Some of them might require additional tools or libraries, like the one mentioned in chapter 5.2, in order to interact with our application. However, this implementation should not be much more complicated than the one that was already done in order to access OpenNLP engine and its part-of-speech task. Also, the application's amount of supported tasks can easily be extend with others provided by OpenNLP engine, in a way similar to the implementation of the current task.

In addition to all, the accessibility and portability criteria are fulfilled due to the specifics of the application, which was developed as a web-based by using Microsoft ASP.NET framework and it was deployed on a remote Windows server machine. Therefore, it can be access by multiple users at any moment as long as they have access to Internet and to the server. The application does not require any specific installations, except for an Internet browser, but most operating systems nowadays have at least one that is preinstall within the package.

Currently the application is deployed on one of the remote servers own by Lionbridge and it is being used by its employees. This application relief the users' workload when they have to perform part-of-speech tagging tasks due to several reasons. Firstly, as already stated the users are linguists and this application gives them the freedom to perform the task without the need of any scripts or commands, which may be an obstacle for them. Moreover, it handles the preparation and management of the files, which until now was manual and time consuming process. And last, the users are not entitle to be in the office and to use company workstations in order to perform their tasks. The application has portable interface that can be accessed even from outside the corporate network.

Future work

As any other software application some improvements can be done in future in order for system to be more efficient and usable. These upgrades can be divided into two categories. The first one concerns the user interface of the application. A good improvement can be the integration of responsive design features. This will allow the user to use the application through a wider range of devices such as tablet or phones. Moreover, some of the elements, like help sections or error messages, can be moved to modular windows. This feature may increase the size of the working area in the application as well as focus the users' attention to the content presented in these modular windows. The user interface for some stages can be also modified to provide reusability features of the stage. For example the testing stage can provide retesting feature with multiple data, not only the one used during the process flow. This improvement will remove the need to start new process flow in order to test with different data.

The second category focuses on the server side efficiency of the web application. A possible improvement might be providing an algorithm to read various types of files, such as Word or Excel, that are not supported at the moment. This way the application will allow the users to use a wider variety of data for their tasks. Another addition to the system can be improving the file and folder structure, by allowing the users to interact with it. However, this feature might lead to the implementation of some sort of control system that will monitor these interactions. And last but not least, a database can be implemented to work side by side with the folder structure, which can provide some features like sorting and viewing the data in various ways.

REFERENCES

[1] Apache OpenNLP Developer Documentation, The Apache Software Foundation, website. Available (10.03.2015): http://opennlp.apache.org/documentation/1.5.3/manual/opennlp.html

[2] Apache OpenNLP Tools 1.5.3 API, The Apache Software Foundation, website. Available (10.03.2015): http://opennlp.apache.org/documentation/1.5.3/apidocs/opennlp-tools/

[3] OpenNLP Proposal, The Apache Software Foundation, website, Available (10.03.2015): http://wiki.apache.org/incubator/OpenNLPProposal

[4] G. L. Saveski, Accessing natural language processing engines and tasks , Tampere University of Technology, 2014

[5] The Linux Information Project, GUI Definition, website. Available (8.09.2015): http://www.linfo.org/gui.html

[6] L. Marquez i Villodre, Part-of-speech Tagging : A Machine Learning Approach Based on Decision Trees, Universitat Politecnica de Catalunya, 1999, pp. 16–20.

[7] Maximum Entropy Framework, SourceForge, website. Available (20.09.2015): http://maxent.sourceforge.net/about.html

[8] Professional Translation and Localization Services, Lionbridge, website. Available (10.11.2015): http://www.lionbridge.com/

[9] Microsoft, ASP.NET Overview, website. Available (10.11.2015): https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx

[10] C# Reference, website. Available (10.11.2015): https://msdn.microsoft.com/en-us/library/618ayhy6.aspx

[11] Microsoft Windows Server 2012, website. Available (10.11.2015): https://www.microsoft.com/en-us/server-cloud/products/windows-server-2012-r2/

[12] NLTK 3.0 documentation, NLTK Project, website. Available (10.11.2015): http://www.nltk.org/

[13] The Stanford Natural Language Processing Group, Stanford NLP Group, website. Available (10.11.2015): http://nlp.stanford.edu/

[14] A Tutorial on Techniques and Applications for Natural Language Processing, Philip J. Hayes, Jaime G. Carbonell, Carnegie-Mellon University 1983, pp 6-10

[15] S. B. Kotsiantis, Supervised Machine Learning: A Review of Classification Techniques, University of Peloponnese, 2007, pp. 2–4.

[16] T. Naseem, B. Snyder, J. Eisenstein, R. Barzilay, Multilingual Part-of-Speech Tagging: Two Unsupervised Approaches, Journal of Artificial Intelligence Research 36, 2009, pp.1–3.

[17] Bootstrap, website. Available (16.04.2016): http://getbootstrap.com/css/

[18] E.D. Liddy, Natural Language Processing. Encyclopedia of Library and Information Science, 2nd Ed., Marcel Decker Inc., 2001, pp. 1-10.

[19] A. Ratnaparkhi, Maximum entropy models for natural language ambiguity resolution, University of Pennsylvania, 1998, pp. 6–51.

[20] M. Krol, I. Tarnopolsky, User Interfaces, The Gale Group Inc., 2002, website. Available (26.04.2016) : http://www.encyclopedia.com/topic/User_Interface.aspx

[21] V. Beal, API - application program interface, website. Available (27.04.2016): http://www.webopedia.com/TERM/A/API.html

[22] V. Beal, The Differences between Thick & Thin Client Hardware, 2006, website. Available (accessed on 27.04.2016): http://www.webopedia.com/DidYouKnow/Hardware_Software/thin_client.asp

[23] V. Beal, The Differences between Thick, Thin & Smart Clients, 2006, website. Available (accessed on 27.04.2016): http://www.webopedia.com/DidYouKnow/Hardware_Software/thin_client_applications.asp

[24] W3C Recommendation, Hypertext Transfer Protocol -- HTTP/1.1, 1996, Website. Available (accessed on 27.04.2016): https://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-01#Connection

[25] SharpNLP - open source natural language processing tools, Microsoft, website. Available (27.09.2016): https://sharpnlp.codeplex.com/

[26] W3C Recommendation, HTML & CSS, website. Available (28.04.2016) : https://www.w3.org/standards/webdesign/htmlcss

[27] Mozilla Developer Network and individual contributors, JavaScript, website. Available (28.04.2016): https://developer.mozilla.org/en-US/docs/Web/JavaScript

[28] The jQuery Foundation, jQuery, website. Available (28.04.2016): https://jquery.com/

[29] Microsoft, Internet Information Services (IIS), website. Available (29.04.2016): http://www.iis.net/overview

[30] St. Walter, K. Hoffman, N. Dudek, ASP.NET 4 Unleashed, Pearson Education, Inc., 2011, pp 237-268.

[31] University of Pennsylvania, Alphabetical list of part-of-speech tags used in the Penn Treebank Project, website. Available (03.05.2016): https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

[32] J. Spolsky, User Interface Design for Programmers, Apress, 2001, pp 5-8.

[33] D. Dragilev, What is User Interface Design? , website. Available (03.05.2016): http://www.freshtilledsoil.com/what-is-user-interface-design/

[34] J. Nielsen, Parallel & Iterative Design + Competitive Testing = High Usability, Nielsen Norman Group, website. Available (03.05.2016): https://www.nngroup.com/articles/parallel-and-iterative-design/

[35] J. Nielsen, Improving System Usability Through Parallel Design, Nielsen Norman Group, 1996, Available (03.05.2016): https://www.nngroup.com/articles/parallel-design/

[36] J. Nielsen, 10 Usability Heuristics for User Interface Design, Nielsen Norman Group, 1995, Available (03.05.2016): https://www.nngroup.com/articles/ten-usability-heuristics/

[37] Cleverism, Top Programming Languages used in Web Development, website. Available(05.05.2016): https://www.cleverism.com/programming-languages-web-development/

[38] Refsnes Data, The world's largest web developer site, website. Available (05.05.2016): http://www.w3schools.com/default.asp

[39] The PHP Group, PHP, website. Available (05.05.2016): https://secure.php.net/

[40] Oracle, Oracle Technology Network for Java Developers, website. Available (05.05.2016): http://www.oracle.com/technetwork/java/index.html

[41] Microsoft, Microsoft Application Architecture Guide, 2nd Edition, Microsoft Press, 2009 pp 55-67.

[42] Microsoft, Microsoft Application Architecture Guide, 2nd Edition, Microsoft Press, 2009 pp 277-297.

[43] J. Frijters, IKVM.NET Home Page, website. Available (07.05.2016): https://www.ikvm.net/

[44] St. J. Murdoch, Hardened Stateless Session Cookies, University of Cambridge, 2008, Available (20.05.2016): http://sec.cs.ucl.ac.uk/users/smurdoch/papers/protocols08cookies.pdf

[45] I. Dacosta, S. Chakradeo, M. Ahamad, P. Traynor, One-Time Cookies: Preventing Session Hijacking Attacks with Stateless Authentication Tokens, Georgia Institute of Technology. College of Computing,2012, pp 1-10

[46] J. G. Ochin, Cross Browser Incompatibility: Reasons and Solutions, International Journal of Software Engineering & Applications, 2011, Available (21.05.2016): http://www.airccse.org/journal/ijsea/papers/0711ijsea05.pdf

[47] Wordpress, Documented approach for cross browser compatibility, 2013, Available (21.05.2016): https://testingnerds.files.wordpress.com/2013/03/documented-approachforcrossbrowsercompatibility.pdf

[48] M. A. Hearst, Search user interface, Cambridge university press, 2009, website, Available (21.05.2016): http://searchuserinterfaces.com/book/sui_ch2_evaluation.html

[49] R. Jeffries, J. R. Miller, C. Wharton; K. M. Uyeda, User Interface Evaluation in the Real World: A Comparison of Four Techniques, Hewlett-Packard Company, 1991, Available (22.05.2016): http://www.hpl.hp.com/techreports/91/HPL-91-03.pdf