

**BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA**  
**FACULTY OF MATHEMATICS AND COMPUTER SCIENCE**  
**SPECIALIZATION COMPUTER SCIENCE, ENGLISH**

**DIPLOMA THESIS**  
**Rest Web Application for Generating**  
**Questions Based on a Text**

**Supervisor**

**lect. dr. Guran Adriana**

**Author**

**Gal Oscar**

**2019**

**UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA**  
**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**  
**SPECIALIZAREA INFORMATICĂ, ENGLEZA**

**LUCRARE DE LICENȚĂ**

**Aplicație Rest Pentru Generarea de Întrebări**  
**Bazate pe un Text**

**Conducător științific**

**lect. dr. Guran Adriana**

**Absolvent**

**Gal Oscar**

**2019**

# Contents

<b>Introduction .....</b>	<b>3</b>
<b>Fundamentals .....</b>	<b>5</b>
<b>1.1   Regex Expressions .....</b>	<b>5</b>
1.1.1    Basic Regular Expression Patterns .....	5
1.1.2    Disjunction and grouping .....	7
1.1.3    More Operators .....	8
<b>1.2   Natural language processing and natural language understanding.....</b>	<b>9</b>
1.2.1    Tokenization .....	9
1.2.2    Classifiers .....	11
1.2.3    Part-of-speech tagging .....	12
<b>Questions .....</b>	<b>15</b>
<b>2.1   Question grammar rules.....</b>	<b>15</b>
2.1.1 Yes/No Questions .....	15
2.1.2 “Wh” Questions .....	16
2.1.3 Tag Questions .....	17
2.1.4 Negative Questions .....	18
<b>2.2   Question Formation .....</b>	<b>18</b>
<b>Case Study: StudHelper.....</b>	<b>21</b>
<b>3.1 Software Development.....</b>	<b>21</b>
3.1.1 Introduction.....	21
3.1.4 Rest applications.....	23
3.1.3 Analysis and Design .....	24
3.1.4 User Manual.....	27
<b>3.2 Future Improvements .....</b>	<b>33</b>
<b>Conclusion and future work.....</b>	<b>34</b>
<b>Bibliography .....</b>	<b>35</b>

# List of Figures

FIGURE 1.1: EXAMPLE OF REGEX EXPRESSION AND THE MATCHING CASES FOR EACH [1] .....	6
FIGURE 1.2: ALIASES FOR COMMON SETS OF CHARACTERS [1] .....	8
FIGURE 1.3: COUNTING REGULAR EXPRESSIONS [1] .....	8
FIGURE 1.4: CHARACTERS THAT HAVE TO BE BACKSLASHED [1] .....	9
FIGURE 1.5: THE TOKENIZATION PROCESS [3] .....	10
FIGURE 1.6: EXAMPLE OF COMPLEX TOKENIZATION. [3] .....	10
FIGURE 1.7: SUPERVISED CLASSIFICATION [4] .....	11
FIGURE 1.8: EXAMPLE OF POS TAGGING OF A SENTENCE [3] .....	13
FIGURE 1.9: DEPENDENCIES OF THE SENTENCE IN POS TAGGING. [3] .....	13
FIGURE 3.1: DIAGRAM OF THE MCV DESIGN PATTERN .....	25
FIGURE 3.2: DJANGO REST ARCHITECTURE DESIGN [18] .....	25
FIGURE 3.3: DATABASE DIAGRAM OF OUR API .....	26
FIGURE 3.4: THE BASIC ARCHITECTURE OF AN ANGULAR APPLICATION .....	27
FIGURE 3.1: INDEX PAGE OF THE APPLICATION .....	28
FIGURE 3.2: THANK YOU FOR REGISTRATION PAGE .....	28
FIGURE 3.3: RECEIVED EMAIL FROM THE SERVER .....	29
FIGURE 3.4: PASSWORD RESET FORM .....	29
FIGURE 3.5: PASSWORD RESET EMAIL .....	30
FIGURE 3.6: PASSWORD CHANGE FORM .....	30
FIGURE 3.7: HOMEPAGE .....	31
FIGURE 3.8 ADDING A NEW TEXT .....	31
FIGURE 3.9: FILE PAGE WITH THE QUESTIONS PART 1 .....	31
FIGURE 3.10: FILE PAGE WITH THE QUESTIONS PART 2 (FORM NOT COMPLETED) .....	32
FIGURE 3.11: FILE PAGE WITH THE QUESTIONS PART 2 (AFTER CHECKING ANSWER) .....	32

# Introduction

How would someone tell if you read this text or not? They could ask you to summarize the text or to compare it to other papers from the same genre or the same topic. They could also ask you to discuss the weaknesses and strength of the paper.

Of course, if you are a highly skilled and motivated reader, then you will not be checked if you retained necessary information from the text. However, this is not the case with all the readers. For instance, an elementary school teacher might ask his or her students basic questions because they are still dealing with understanding complicated texts.

Nowadays information is available everywhere on the internet (e.g., Wikipedia, Medium, and so on). However, studying new subjects is a challenge that we have to face on a daily basis. The purpose of the present thesis is to highlight the importance of a tool developed for students in which they can quickly learn by exercising the knowledge that they have just read by answering questions.

We aim to create a system for generating questions that take as input a text, for instance, a web page or an encyclopedia article or even a text from a course book. After the input was received, it is processed, and by that, we get to parse it, understand it and extract every helpful information from the received text. The final step is the generation of useful questions to see how much from the topic was retained by the reader.

Generating such questions can be a time-consuming and effortful process. In this research, we work toward automating that process. In particular, we are going to focus on generating questions from individual texts.

The thesis is structured as follows:

Chapter 1 presents an overview of the fundamental theory required for succeeding in implementing the generator. Throughout this chapter, we briefly explain regex expressions (an essential step into achieving the desired results) and the natural language understanding and natural language processing tools used in the algorithm. The end of the chapter focuses on explaining the theory behind the formations of questions and the notations used for the part of speech tagging in the English language.

The second part focuses on the approach used in achieving the wanted generator. In this section, we presented the grammar behind the formation of questions in English and also the steps in achieving the desired question generator. The steps of creating the algorithm are accompanied with some suggestion on how to choose the perfect sentences for the creation of questions. This suggestion uses a lot of Natural Language Understanding and Processing tools that are presented in the first chapter.

Chapter 3 mainly focuses on the development of the rest-full based application in which the algorithm that we implemented with the help of the current thesis is seen in action. The final application is supposed to help people learning by answering questions based on a text received as input. In the rest-full based program that we implemented the user can create an account and associate to his or her profile, texts for practicing what they retained from the topic and if any mistakes are made, those mistakes are highlighted for further study by the user.

We are going to focus on non-fictional texts that convey factual information rather than opinion. While personal essays and narratives texts would be an exciting topic, we leave these problems to further work, and that is because this thesis is focused on generating questions from teachers' materials (lectures or course supports).

# Chapter 1

## Fundamentals

This section will give an introduction into regex expression, natural language processing, natural language understanding and the grammar behind questions in English.

### 1.1 Regex Expressions

Regular expressions are an algebraic notation for characterizing a set of strings, put in other words, regex is a sequence of characters that define a search pattern. These expressions are beneficial for searching in a text when we are looking for instance through a corpus a given model.

When searching through a text, the search can be designed to return every match on a line, if there are more than one, or it can be set to return only the first apparition of the given pattern. For the following example, we generally underline the exact part that matches the model and only the first match. The backslash character doesn't usually delimit a regular expression, but in the current thesis, they will be shown with a backslash for better readability.

Depending on the regular expression parser, some of them may recognize only subsets of the pattern, or to treat some exceptions slightly differently. [1]

#### 1.1.1 Basic Regular Expression Patterns

The most straightforward kind of regular expression is a sequence of simple characters. For instance, if in a text you are looking for the word "apple" the corresponding regular expression would be `\apple\`.

A regular expression is case sensitive so if you would look for the word "Apple" the above rule would not hold. We can solve that issue by using square brackets [ and ]. The string between the square brackets is called disjunction of characters to match. An example that could match "Apple" and also "apple" would be `\[Aa]pple\`.

Another rule that should be remembered is the sign dash (-). It is used when a range of characters is targeted. For instance, the pattern `\[2-5]` specifies any one of the characters 2, 3, 4, or 5. The pattern `\[a-f]` specifies one of the characters a, b, c, d, e or f.

The square brackets can also specify what a single character or a set of characters cannot be, and it is done using the caret ^. If the caret is placed after the open square brackets than it negates the resulting pattern. For example, if you are looking for the first small letter in a sentence than the regex that would to that be `\[^A-Z]`.

Optional elements also have a special rule and it is given by the question mark. For instance, if you are looking for an optional s for the word teacher then the following pattern can be used `\[teacher?s]`.

The set of operators that allows us to specify rules like "one or many" for a given character is based on the sign asterisk or \*. The symbol actually means "zero or more occurrences of the immediately previous character or regular expression". For example, if you want to find a word that was one or more s in it is `\aa*`, meaning that the founded word should contain one a followed by zero or more as. More complex patterns can be repeated, for the situation, if you want to find a natural number that has at least one character then the pattern could be used: `\[1-9][0-9]*`.

One important character also called a wildcard expression is the period sign `\.` that can match any single character. `\ca.` would match the word "cap" but also the word "can".

Anchors are also essentials rules that you should keep in mind while working with regex expressions. The most common anchors are the caret "^" and the dollar sign "\$". The caret matches the start of the line. The pattern `\^Anna` matches the word Anna if and only if the word is placed at the beginning of the line. The same goes for the dollar sign, but this time the symbol matches the character/characters placed at the end of the line.

All of the above rules are also shown in figure 1.1.

Regex Expression	Match	Example Pattern Match
<code>\Inc\</code>	Any word that is exactly form with those chars	Apple <u>Inc.</u> is an American company
<code>\[Aa]pple\</code>	Apple or apple	<u>Apple</u> Inc. is an American company
<code>\[0-9]</code>	Any digit character	My phone number is ( <u>5</u> 55232)
<code>\[0-9]*</code>	Any digit character zero or more times	My phone number is ( <u>555232</u> )
<code>\^[a-z]</code>	Any character that is upper	<u>A</u> pple Inc. is an American company
<code>\woodchuck?s\</code>	woodchuck or woodchucks	<u>woodchucks</u>
<code>\^Apple\</code>	Apple if and only if it is at the beginning of the row	<u>Apple</u> Inc. is an American company
<code>\.\$</code>	any character at the end of the line	Apple Inc. is an American company <u>.</u>

FIGURE 1.1: EXAMPLE OF REGEX EXPRESSION AND THE MATCHING CASES FOR EACH [1]



### 1.1.2 Disjunction and grouping

Suppose we need to search for texts in a recipe article; perhaps we are interested in the words salt and pepper. In a case like this, we need to create a rule like "salt or pepper" and this is done using the disjunction operator also called the pipe symbol `|`. The model `\salt|pepper\` matches either the string cat or the string dog.

Disjunction is an essential operator. Many times, when working with regular expressions we need to find words that might end with some sets of chars or to start with that set of chars, or we need to find specific words. If we are looking for words ending for instance in "ie" and also words that end in "vb", the following regex should do the trick `\[.*ie|.*vb]\`.

The disjunction operator has low precedence so that the complete subexpression on the left or the entire subexpression on the right is matched.

To make the disjunction character to work only to a specific pattern, we need to add the parenthesis operator ( and ). By closing group characters in parenthesis, we make them act like a single character for neighboring operators like pipe and asterisks. So the model `\part(y|ies)\` would specify that we meant the disjunction only apply to the suffixes y or ies.

Operators may sometimes take precedence over another, requiring us occasionally to use parentheses to specify what we mean, this is just like the case with basic arithmetic where multiplication and division take precedence over addition and subtraction. The following table gives us the order of the regex expressions operator precedence, from higher priority to lowest precedence.

Parenthesis	( )
Counters	+ ? * { }
Sequences and anchors	The ^my end\$
Disjunction	

TABLE 1.1: PRIORITY FOR REGEX EXPRESSIONS OPERATIONS

Regular expressions tend to word greedily, meaning that patterns always match the longest string as they can. Of course, there are ways to enforce a non-greedy matching and that is done using the `*?` operator which matches as little text as possible.

### 1.1.3 More Operators

In figure 1.2 there are presented some aliases for typical ranges, which can save you a lot of typing. Besides what we presented until now, we can also specify numbers as counters, and we do that by enclosing them in curly brackets. For instance, the regex `\{5\}` means "exactly 5 occurrences of the previous pattern or character".

Regex Expression	Expansion	Match	First match
<code>\s</code>	<code>[\r\t\n\f]</code>	whitespace (space or tabs)	
<code>\S</code>	<code>[^\s]</code>	non-whitespace	<u>A</u> нна has 5 apples
<code>\d</code>	<code>[0-9]</code>	any digit	Anna has <u>5</u> apples
<code>\D</code>	<code>[^0-9]</code>	any non-digit	<u>A</u> нна has 5 apples
<code>\w</code>	<code>[a-zA-Z0-9_]</code>	any alphanumeric/underscore	<u>D</u> iana
<code>\W</code>	<code>[^\w]</code>	a non-alphanumeric	Anna has 5 apples <u>!</u>

FIGURE 1.2: ALIASES FOR COMMON SETS OF CHARACTERS [1]

Ranges of numbers are also a permitted syntax in a regex. So `\{a, b\}` define from a to b occurrences of the previous model or characters; and `\{a, \}` means at least an appearances or the preceding expression. In figure 1.3 are presented the regexes for counting and summarizing.

Regex Expression	Match
<code>{a}</code>	a occurences of the previous char or expression
<code>{a,b}</code>	from a to b occurences of the previous char or expression
<code>{a, }</code>	at least a occurences of the previous char or expression
<code>{ ,b}</code>	at most b occurences of the previos char or expression
<code>?</code>	exactly zero or one occurrence of the previous char or expression
<code>+</code>	one or more occurences of the previous expression
<code>*</code>	zero or more occurences of the previous expression

FIGURE 1.3: COUNTING REGULAR EXPRESSIONS [1]

Finally, if you want to search in a text for a question mark or an asterisk, how would you do it? You cannot say `\\.\\` because this would translate into "any character from the text". Certain special characters, like the ones mention above have to have a so-called escape character before them; and in our case, it is the backslash (`\`). Figure 1.4 exemplifies the usage of the backslash character and also some other rules that need to be kept in mind.

Regex Expression	Match	First pattern matched
\.	a period "."	Dr. Gray, I assume
\*	an asterisk "*"	Thom*son
\?	a question mark	Do you think that aliens exists ?
\t	a tab	
\n	a new line	

FIGURE 1.4: CHARACTERS THAT HAVE TO BE BACKSLASHED [1]

## 1.2 Natural language processing and natural language understanding

Natural language processing (shorthand NLP) is a subfield of artificial intelligence and computer science. It focuses on how to program a computer to process and analyze large amounts of natural language data. NLU is the shorthand for natural language understanding and it is a subtopic of computer science and NLP, that deals with machine reading comprehension. [2]

### 1.2.1 Tokenization

Before we start explaining the NLP tools that we used to achieve the question generator, we first need to establish some grand rules. We have to define what we consider a word. Let's take as an example the following sentence: "Apple is looking at buying U.K. startup for \$1 billion." How many words are in there? There are 11 words if we don't count the punctuation marks as words, and 12 if we count them. Depending on the case, the algorithm should or shouldn't count punctuation as words. In our case, we will have to take in consideration also the punctuation.

Tokenization in computer science is the process of converting a sequence of characters (in our case our input text) into a series of tokens (strings with an assigned, and thus identified meaning). The resulting tokens are then passed to another form of processing. The tokenization process can be considered a sub-task of parsing the input. Let's take the previous sentence example and see precisely for the tokens are formed, illustrated in figure 1.5.

Apple is looking at buying U.K. startup for \$1 billion.											
0	1	2	3	4	5	6	7	8	9	10	11
Apple	is	looking	at	buying	U.K.	startup	for	\$	1	billion	.

FIGURE 1.5: THE TOKENIZATION PROCESS [3]

The tokenization process is composed of the following steps. First, the raw text is split on whitespace characters, similar to the `text.split(' ')` function. The tokenizer then processes the text from left to right, performing two checks on each substring:

1. Exception checks. For example, "don't" doesn't contain white spaces but actually, there are two words inside that token, which are "does" and "not" and our tokenizer will split that word into "do" and "n't", while "U.K." should always remain one token.

2. Prefix, suffix or infix check. At this point, the tokenizer will try to split by special characters like commas, periods, hyphens or quotes. We will consider prefix: character or characters at the beginning, e.g. \$, (, ". Suffix: character or characters at the end, e.g. km, ), ", !. Infixes are the character or characters in between, e.g. -, --, /, ...

The tokenizer that we are going to use is the one from spaCy [3]. Because it's tokenizer can split, complex and nested tokens like combinations of abbreviations and many punctuation marks. A complicated example of the previously explained steps appears in the figure 1.6.

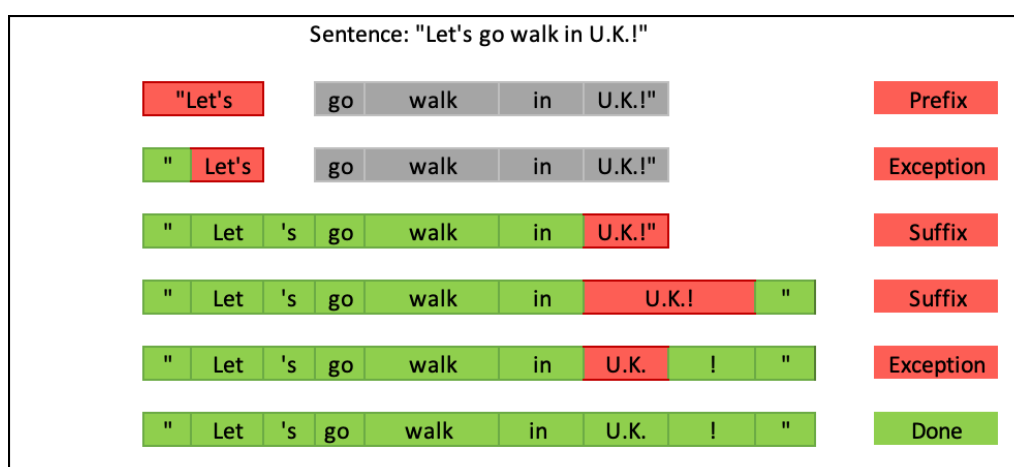


FIGURE 1.6: EXAMPLE OF COMPLEX TOKENIZATION. [3]

## 1.2.2 Classifiers

Classifiers, when talking about Natural Language Processing, labels tokens with category labels or class labels. They actually try to predict the class of given data points. Typically, labels are represented with strings (such as “politics” or “sports”).

For creating the desired question generator, we use the NLTK classifiers. As it is mentioned in NLTK documentation [4], classification is the process of choosing the correct class label for a given input. The framework defines several classifier classes:

- a) ConditionalExponentialClassifier
- b) DecisionTreeClassifier
- c) MaxentClassifier
- d) NativeBayesClassifier
- e) WekaClassifier

A classifier is called supervised if it is build based on a training corpora. The framework used by supervised classification is shown in Figure 1.7.

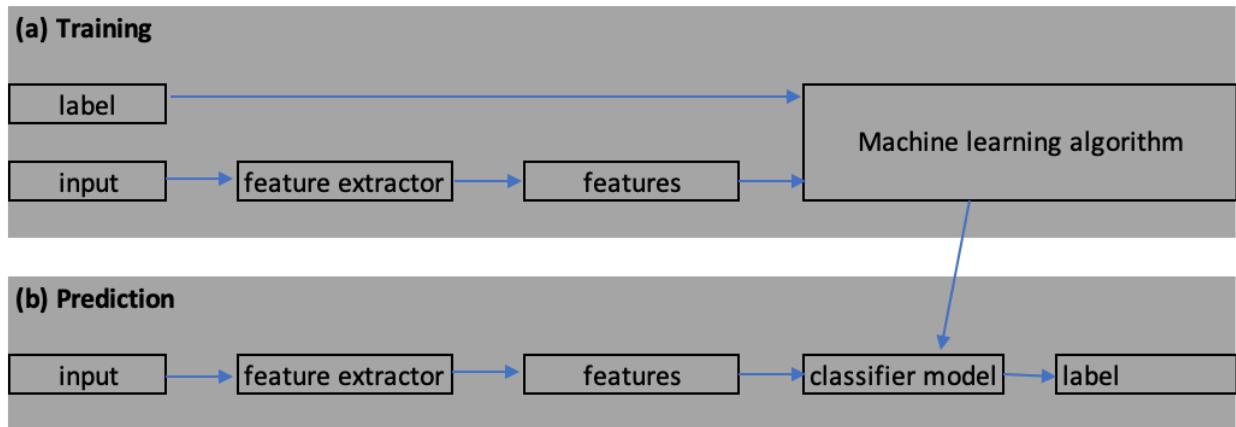


FIGURE 1.7: SUPERVISED CLASSIFICATION [4]

In the above picture, during training ((a) Training), a feature extractor is used to convert each input to a feature set. For instance, the feature extractor for gender identification classifier would construct a dictionary that contains the last two characters of the word in order to figure out if the word is feminine or masculine, and that dictionary will be the features set. Pairs of features

sets and labels are then “fed” to the machine learning algorithm to generate a model (we will not go further talking about machine learning algorithms).

During prediction (stage (b)), the same feature extractor is used to convert unseen inputs to feature sets. This feature sets are then “fed” into the model, generated at the previous step, which generates predicted labels. [4]

### 1.2.3 Part-of-speech tagging

A Part-of-speech Tagger (short POS Tagger) is a piece of software that takes as input some text in some language (in our case is English) and assigns parts of speech to each word (tokens), such as nouns, verb, pronoun, preposition, adverb, conjunction, participle and article.

The original POS Tagger was originally written by Kristina Toutanova. Since that time, Dan Klein, Christopher Manning, William Morgan, Anna Rafferty, Michel Gally, and John Bauer have improved its speed, performance, usability, and support for other languages, as it is specified in the documentation of the Stanford documentation [5].

Part of speech is a key feature in Natural Language Processing, because with the help of it we reveal a lot of useful information about the word and its neighbors. As it is presented in [2], knowing whether a word is a noun or a verb tells us about likely neighboring words (nouns are preceded by determiners and adjectives, verbs by nouns) and syntactic structure word (nouns are generally part of noun phase), making the POS tagger a key aspect of parsing. Parts of speech are also useful features for labeling named entities like people or organizations in information extraction.

As a POS tagger we are going to use the one from SpaCy, because it’s the best open source python library for Natural Language Processing. The framework uses terms like **head** and **child** to describe the words **connected by a single arc** in the dependency tree. The term **dep** is used for the arc label, which describe the type of syntactic relation that connects the child to the head. You can see an example of a tagged sentence in figure 1.8, where text is the original word text, lemma is the base form of the word, POS is the simple part-of-speech tag, tag is the detailed part-of-speech tag and dep is the syntactic dependency, i.e the relation between tokens. [3]

Sentence: Apple is looking at buying U.K. startup for \$1 billion					
Text	Lemma	Pos	Tag	Dep	Shape
Apple	apple	PROPN	NNP	nsubj	Xxxxx
is	be	VERB	VBZ	aux	xx
looking	look	VERB	VBG	ROOT	xxxx
at	at	ADP	IN	prep	xx
buying	buy	VERB	VBG	pcomp	xxxx
U.K.	u.k.	PROPN	NNP	compound	X.X.
startup	startup	NOUN	NN	dobj	xxxx
for	for	ADP	IN	prep	xxx
\$	\$	SYM	\$	quantmod	\$
1	1	NUM	CD	compound	d
billion	billion	NUM	CD	probj	xxxx

FIGURE 1.8: EXAMPLE OF POS TAGGING OF A SENTENCE [3]

Also using SpaCy build-in visualizer, we can see how each word is interconnected with the other words. This can be seen in Figure 1.9

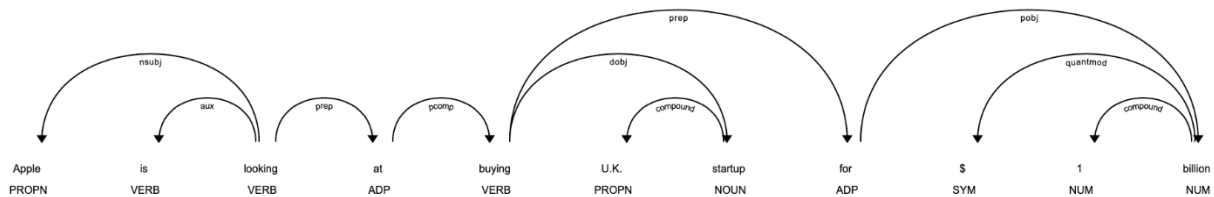


FIGURE 1.9: DEPENDENCIES OF THE SENTENCE IN POS TAGGING. [3]

Last but not least, in this section we will present the tagset that is going to be used in this paper. An important tagset for English is the 45-tag Penn Treebank tagset shown in table 1.2, which was being used in labeling many corpora.

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
<b>CC</b>	coordinating conjunction	<i>and, but, or</i>	<b>NNS</b>	noun, plural	llamas	<b>PDT</b>	predeterminer	all, both
<b>CD</b>	cardinal number	One, two	<b>NNP</b>	proper noun, singular	IBM	<b>POS</b>	possessive ending	's
<b>DT</b>	determiner	A, the	<b>NNPS</b>	proper noun, plural	Carolinas	<b>PRP</b>	personal pronoun	I, you, he
<b>IN</b>	preposition/subordinating conjunction	of, by, in	<b>PRPS</b>	possessive pronouns	yours, one's	<b>VBD</b>	verb past tense	ate
<b>JJ</b>	adjective	yellow	<b>RB</b>	adverb	quickly	<b>VBG</b>	verb gerund	eating
<b>EX</b>	Existential 'there'	there	<b>RBR</b>	comparative adverb	faster	<b>VCN</b>	verb past participle	eaten
<b>FW</b>	foreign word	holla	<b>RBS</b>	superlative adverb	fastest	<b>VBP</b>	verb non-3sg present	eat
<b>JJR</b>	comparative adj	bigger	<b>RP</b>	particle	up, off	<b>VBZ</b>	verb 3sg person	eats
<b>JJS</b>	superlative adj	wildest	<b>SYM</b>	symbols	+, -, ,	<b>WDT</b>	wh-determ.	which, that
<b>LS</b>	list item marker	1, 2, One	<b>TO</b>	"to"	to	<b>WP</b>	wh-pronoun.	what, who
<b>MD</b>	modal	can, should	<b>UH</b>	interjection	ah, oops	<b>WPS</b>	wh-possesive	whose
<b>NN</b>	single or mass noun	llama	<b>VB</b>	verb base form	eat	<b>WRB</b>	wh-adverb	how, where

TABLE 1.2: PENN TREEBANK PART-OF-SPEECH TAGS (EXCLUDING PUNCTUATION) [3]



# Chapter 2

## Questions

This chapter will mainly focus on the question generator. The first part presents an introduction to forming question in English, after that the algorithm that generates question will be presented.

### 2.1 Question grammar rules

Questions are the heart of understanding what we read. Asking questions is about interacting with the text to develop a profound understanding. In order to be capable to create a question generator, first we have to explain the types of question in English and also how they are formed.

As it is presented in the article [6], in English we have 4 essential types of questions. Every type will be presented and explained in their own sub chapter.

#### 2.1.1 Yes/No Questions

The first type of question are the ones with a yes or no answer. Let's take some examples and see how these type are formed:

(1) Sentence: It is a sunny weather today.

Question: Is it a sunny weather today?

(2) Sentence: You can play the piano like a master.

Question: Can you play the piano like a master?

(3) Sentence: Mihai plays the piano.

Question: Does Mihai play the piano?

(4) Sentence: Mihai played the piano yesterday.

Question: Did Mihai played the piano yesterday?

## CHAPTER 2 – QUESTIONS

(5) Sentence: Mihai and Angela play the piano.

Question: Do Mihai and Angela play the piano?

There is a difference between questions 1-2 and 3-5, for the first cases (1-2) the sentence has an auxiliary verb “to be” so the question is simply made by inverting the verb with the pronoun. For the second case (3-5) things are a little different, meaning that when the question is formed you have to put “do” or “does” at the beginning, but it also has to be at the correct tense (for example if the sentence is in the past tense then you have to put “did”).

### 2.1.2 “Wh” Questions

The second types of question are the five “W” questions. The “Five Ws” are presented in the table 1.3. Here things start to become a little trickier, because besides the five “W” words there are some other words like “how” that can be integrated in this category of questions. Also, this type of question is also known as “the wh questions”.

Words	Usage
<b>What? Which?</b>	to ask about things
<b>Where?</b>	to ask about places
<b>Who?</b>	to ask about people
<b>When?</b>	to ask about time
<b>Why?</b>	to ask about reason
<b>How?</b>	to ask about the way things happened or are done
<b>How many? How often? How much?</b>	to ask about number or amounts

FIGURE 1.3: THE WORDS “WH” AND THE USAGE OF THEM

For the above mentioned words we will present some examples and ways of forming those questions. The reason why we will only present a few of them, is because we only took into consideration the question which will help us the most in generating the question generator for students.

## CHAPTER 2 – QUESTIONS

### a) WHO

Sentence: Anna is playing football in the central park.

Question: Who is playing football in the central park?

The above question is formed as follows: we put the “wh” word, in our case “who”, into the subject position. We use the word “who” because the subject in this case is a person.

### b) WHAT

Sentence: Every year, earthquakes causes thousands of deaths.

Question: Every year, what causes thousands of deaths?

The question is formed by replacing the subject of the sentence with the word “what”, but this can be done if and only if the subject is an object or a thing. No other changes are required in order to obtain this type of questions.

### c) WHEN

Sentence: We can register for graduation in July.

Question: When can we register for graduation?

This type of question is a little more difficult to form. We will divide the process of creating this type in 3 steps. The first step is to identify the date component from the sentence (for our case “in July”), and replace it with the “W” word; the resulting sentence will be: “We can register for graduation when”. The second step is to move the inserted word in front of the sentence; the sentence will become: “When we can register for graduation”. The final step is to move the operator in front of the subject; the final result will be: “When can we register for graduation?”

### d) WHERE

This word is mainly used in the same manner as the point (c) from above. [6]

## 2.1.3 Tag Questions

Tag questions are actually simple sentences, which have a question tag at the end. They are mainly used to check if the reader understood something correctly or not. Example of tag questions:

- (1) The train leaves at 9 a.m., doesn’t it?
- (2) You will bring the cake, won’t you?
- (3) They went to the cinema, didn’t they?
- (4) You play the guitar, don’t you?

## CHAPTER 2 – QUESTIONS

In this type of question there are 2 cases, if the verb is an auxiliary one, or if it isn't. For the case where the verb is an auxiliary one, if the sentence is in a positive form, then it is followed by a negative question tag, and if it is a negative question then it is followed by a positive tag; the tag being created with the verb from the sentence.

If the verb isn't an auxiliary one, then the question is formed by forming the question tag with "do/does" at the correct tense. Examples for this case can be seen in 3-4.

### 2.1.4 Negative Questions

Negative questions are in some way similar with the tag questions, because they are both used in order to confirm something you believe is true. Some examples of this type of questions can be:

- (1) Didn't you hear the news?
- (2) Hasn't he called back yet?

These types of questions can be formed in 2 ways. The first way implies that the verb that is used should be a verb contraction, and the formula is [negative verb contractions] + [subject] + [main verb] + [other information]. The second way is used in more formal settings, you might use "not" instead of a contraction, having the formula: [auxiliary verb] + [subject] + not + [main verb] + [other information]. Examples of the two methods are shown below.

- (3) Wouldn't you like another cup of coffee?
- (4) Has she not handed in her assignment?

## 2.2 Question Formation

In this sub-chapter we will present the algorithm for generating the question. The algorithm is one combining the technique for the question generator from [7], [8] and [9]. The steps to achieve the factual question generator are:

1. Tokenization: Tokenize the input data received from the user using a tokenizer such as the one from SpaCy

## CHAPTER 2 – QUESTIONS

2. POS-tagging: Assign part-of-speech to every token from the input using a POS tagger such as the one from Spacy or NLTK
3. True-casing determination: Assign the tokens their true capitalization case styling using a true-caser (also present in SpaCy framework or one such as Stanford CoreNLP's TrueCaseAnnotator). This step is especially useful for textbooks that include important vocabulary terms in a caps-lock style, but it could also be skipped if, for instance you are trying to achieve a question generator for newspaper articles.
4. Subject/phrase determination: Build a list of subjects and corresponding phrases from the tokens and the part-of-speech tags using a multiclass classifier. A multiclass classifier or trivial determiner can then be used for question generation.
5. Synonym replacement. Convert some work into their synonym words. This step aims to prevents students or whoever is using the question generator from matching the words from the question to the ones in the input text, in order to answer the questions.
6. Additional information extraction: Associate additional information with each subject/phrase match that could be useful for question generation using regular expressions.
7. Question formation: Attempt to form a question of each question type or each subject/ phrase match using basic patterns of each question type as a guide. The types of questions and how they are formed is presented in the previous sub-chapter.
8. False answer formation: Form similar, but false, answers for each question based on the question type using patterns of good false answer for each question as a guide. For instance, true/false questions can be negated to determine the correct and incorrect answers, while word embedding or a system scoring similarity between words and POS tags of other subjects can be applied for fill-in-the-blank type question
9. Shuffle and return questions

After implementing all the steps from above you should have a question generator that is capable to create questions from a text input.

## CHAPTER 2 – QUESTIONS

In order to achieve a better question generator, we will have to use the classifier that we spoke about (in chapter 1). When we receive the text as an input the first step will be to use the classifier to classify the text into a specific type. After classifying the text into a category, we have to mark the sentences that will be used to create questions. In order to figure out which sentence is in correlation with the subject of the text, a textual entailment classifier should be used. Textual entailment takes a text (the sentence in our case) and a hypothesis (the category class of the text) and checks if the two are in a relation. If they are, then we will mark the question for generating questions out of it.

# Chapter 3

## Case Study: StudHelper

In this chapter we will go through the software development of the application, the problem that I ran into when designing the application and future improvements.

### 3.1 Software Development

As it is specified in the title of this thesis paper, the final application should be a REST-full based application, in which a user can give as input text (scientific texts only in this version of the application) and the program should return a set of relevant question from the text. This sub-chapter will have 4 main parts. In the first one (“Introduction”) fundamentals about web development technologies are explained, the second part (“Rest applications”) presents in details the Rest architecture for servers, the third part (“User manual”) mainly focuses on the frontend of the application and the technologies behind also explaining how the application is supposed to work, the final section (“Analysis and Design”) presents an overview of how the application StudHelper was developed containing also UML diagrams of the application.

#### 3.1.1 Introduction

Let’s start from the beginning; in order to have a fully functioning rest web site there are 2 components required: one is the API (the server side or the backend part of the program) and the client application (the frontend part). In order to understand everything, we will go through each part and briefly explain how those components work.

In computer programming, an API (application programming interface) is a set of subroutine definitions, communication protocols, and tools for building software. The first time when somebody used the term API seems to be in an article created by Ira W. Cotton, Data Structures and techniques for remote computer graphics, published in 1968. The concept of API design evolved with speed in the last couple of years. [10]

## CHAPTER 3 – CASE STUDY: STUDHELPER

There are a lot of API designs, but we are going to use the REST one, which is presented in the next sub chapter. The communication protocol for the application is in our case HTTP. The acronym stands for Hypertext Transfer Protocol and it's a stateless, application-layer protocol for communicating between distributed systems also being the foundation of the modern web. Http allows the communication between clients and server. The communication usually takes place over TCP (Transmission Control Protocol) / IP (Internet Protocol), but any reliable information transport can be used. The default port for TCP/IP is 80, but also other ports can be used. The communication takes place via requests and responses pair. The client initiates a request on some URL and the server will return a response message. [11]

At the heart of web communication is the request message, which is sent via Uniform Resource Locators (URL for short). This is an example of a URL:

*`http://www.mydomain.com:80/path/to/resource?a=b&x=7`*

Where “http” is the protocol, “www.mydomain.com” is the host, “:80” is the protocol, “/path/to/resource” is the resource path and everything after question mark are query parameters.

When a request is made it is also required to attach to the request a verb. Http has formalized on a few that capture the essentials that are universally applicable for any kind of application:

- GET: fetch an existing resource. The URL contains the path for the resource that has to be return to the user
- POST: create a new resource. This type of request usually carries a payload that specifies the data for the new resource
- PUT: update a resource. Also, in the URL is specified the path for the resource and it also requires a payload in order to change the old values with new ones
- DELETE: delete an existing resource.

The above verbs are not all the verbs in the HTTP communication protocol, but those are the most used ones.

Now that we set the grounds for the backend part of the application, we should get on the same level on the frontend part.

In the past websites had a completely different look than today, on the web pages from those days there were only static texts with a bit of formatting and maybe some animations, and that was all thanks to HTML (Hypertext Markup Language) and CSS (Cascade Styling Sheet).



## CHAPTER 3 – CASE STUDY: STUDHELPER

From then until now front-end development has changed radically, and that was the result of an exponential growth of the JavaScript language. [12]

Some of the core technologies used for frontend development are: HTML, JavaScript and CSS. In this thesis we present the AngularJS, used to be the “golden child” of the JavaScript frameworks, being introduced by Google in 2012.

With the help of modern frameworks, the web site can now be brought to life. The best thing about this kind of frameworks is that, you can write reusable code (components that you can use later), you don’t have to repeat code implementation, and also you can make calls asynchronously to the backed. Also, with the help of modern front-end frameworks there are lots of features that come with the framework; for instance, in Angular there is the *FormControl* class which will give you a huge power when working with forms and validation data. Also, Angular is mobile and desktop ready meaning you have one framework for multiple platforms.

### 3.1.4 Rest applications

We mainly focus on the REST design of an API. REST stands for Representational State Transfer, and it relies on a stateless client-server, cacheable communication protocol – and in virtually all cases, the HTTP protocol is used. As a programming approach, REST is a lightweight alternative to Web Services and RPC. Much like Web Services, a Rest service is:

- Platform independent (you don’t care if the server is a UNIX, the client is a Mac, or Windows, or anything else)
- Language independent (C# can be connected with Java and so on)
- Standard based (runs on top of HTTP)
- Can easily be used in the presence of firewalls

The REST architectural style describes six constrains. These constrains, applied to the architecture, were originally communicated by Roy Fielding in his doctoral [13] dissertation, and defines the basis of the REST-full style.

Those 6 constrains are:

- Uniform interface

## CHAPTER 3 – CASE STUDY: STUDHELPER

- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on demand (optional)

Rest offers no build in security features, encryption, session management, SQL injection or so on; but these features can be added on top of HTTP, for instance for encryption, it can be used on top on HTTPS (secure sockets). A disadvantage when working with API design like this, is cookies. The “st” stands for State Transfer, and indeed, in a good REST design operation are self-contained, and each request carries (transfer) with it all the information (state) that the server needs in order to complete the received request.

As is presented in [14] : “ ... REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems. I described the software engineering principles guiding REST and the interaction constraints chosen to retain those principles, while contrasting them to the constraints of other architectural styles.”

### 3.1.3 Analysis and Design

The web-application is rather simplistic for mainly two reasons: first this is a website that should encourage the user to read and understand the text, we don’t want to bother the user with pop-ups or commercials or anything like that; the second reason is that in this application we only want to show how the question generator algorithm works and to prove that it will helps people study faster.

The structure of our application was primarily designed in a Model-View-Controller manner. When developing the web application, we used AngularJS to design and develop the controllers and the views of the web application and the Django REST framework to design the models. In the figure 3.2 is presented the diagrams that illustrate the MVC design pattern that we used [15].

## CHAPTER 3 – CASE STUDY: STUDHELPER

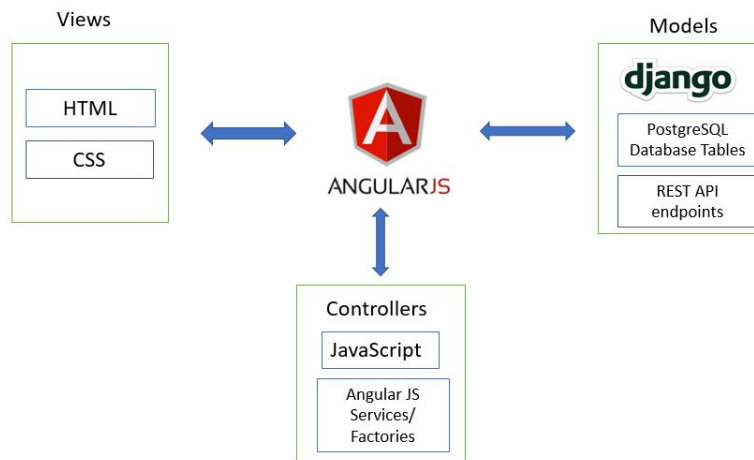


FIGURE 3.1: DIAGRAM OF THE MCV DESIGN PATTERN

The backend is written in Python. As a web framework we used Django [16]/Django REST framework [17] which is a powerful and flexible toolkit for building Web APIs. The reason why we also used Django REST is because we have a client application which we will talk about later and we needed the REST technology in our backend application, and simple Django doesn't offer us those capabilities because it follows a model-template-view architectural pattern.

The design of the Django REST framework architecture can be seen on figure 3.1

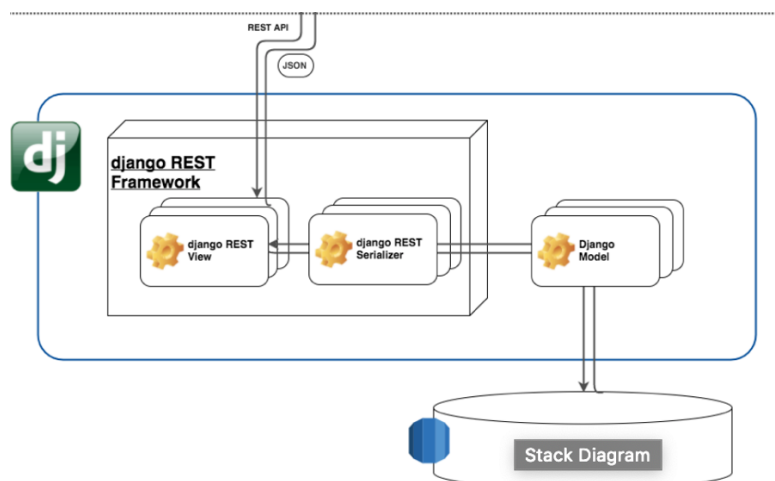
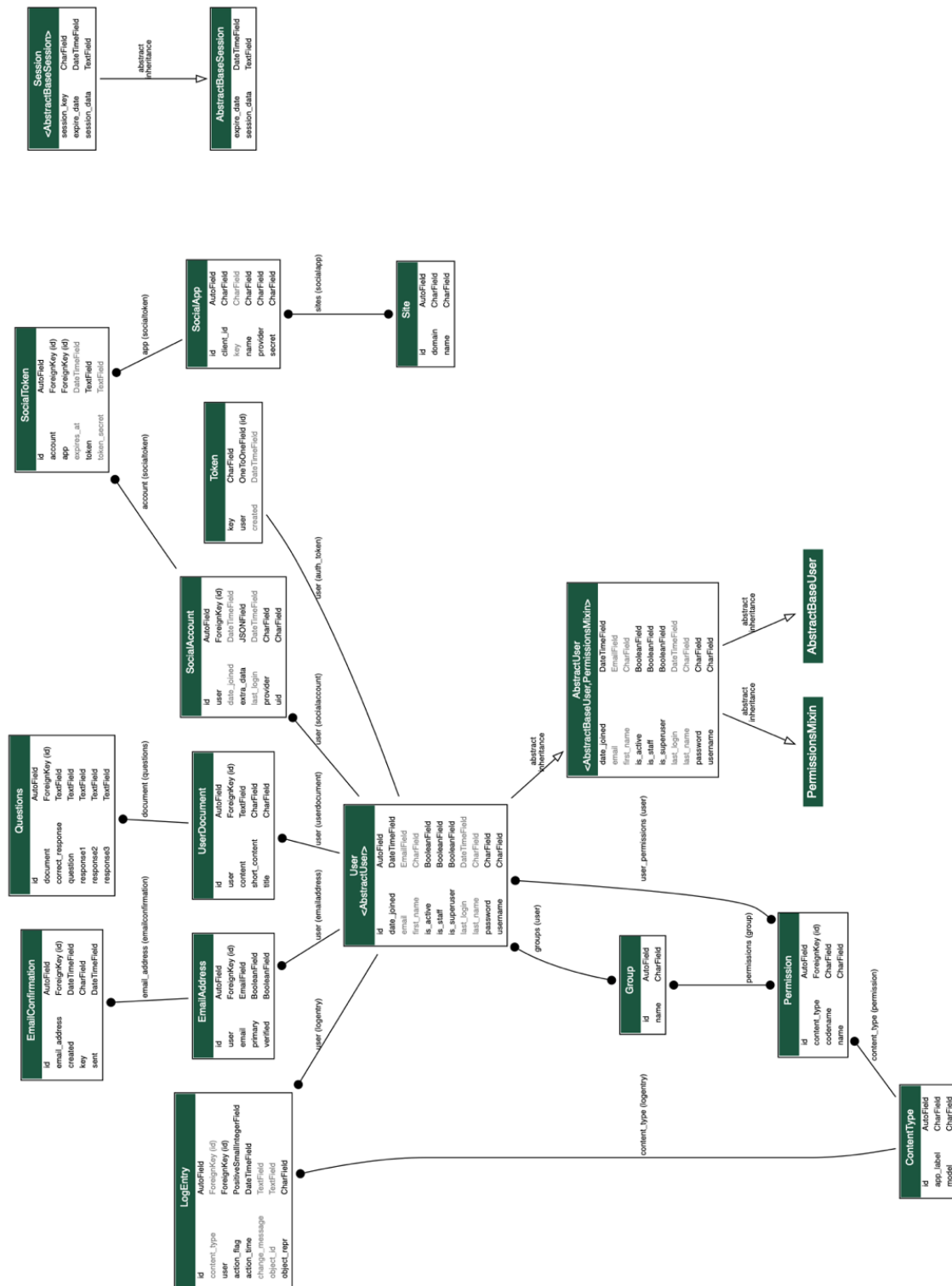


FIGURE 3.2: DJANGO REST ARCHITECTURE DESIGN [18]

## 26

FIGURE 3.3: DATABASE DIAGRAM OF OUR API



## CHAPTER 3 – CASE STUDY: STUDHELPER

If we are talking about the frontend application, it is done in AngularJS. Angular is a framework for building client applications in HTML, TypeScript and CSS; it is written in TypeScript. The basic building blocks of an Angular application are NgModules, which provide a compilation context for components [19]. In the figure 3.4 is a diagram that shows how these basic pieces are related.

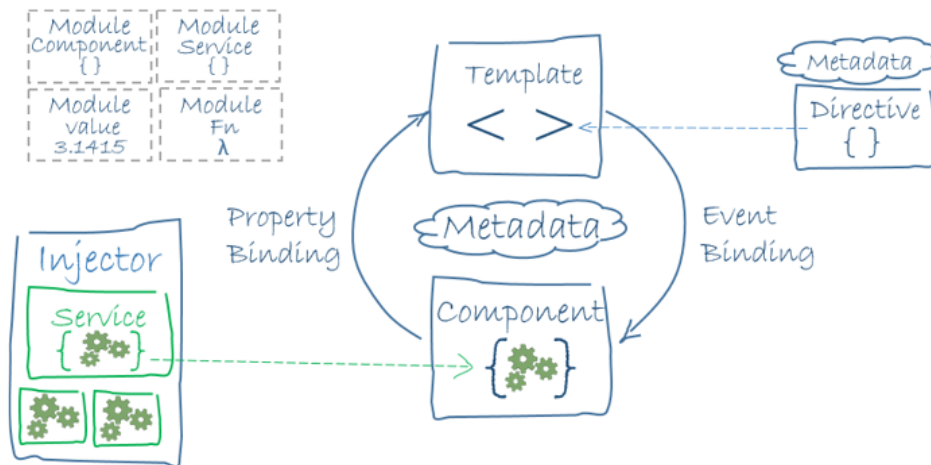


FIGURE 3.4: THE BASIC ARCHITECTURE OF AN ANGULAR APPLICATION

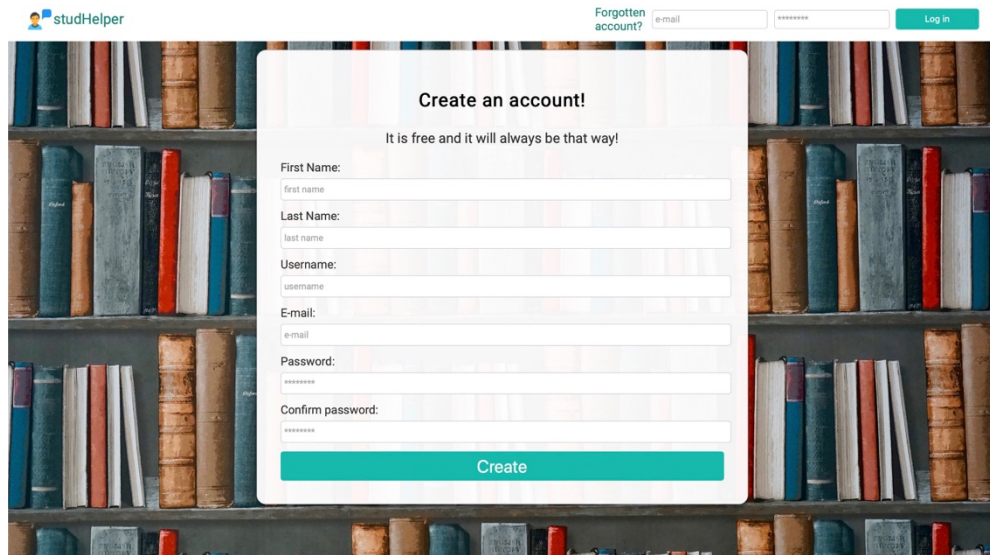
### 3.1.4 User Manual

In the previous sub-chapter, we already presented an introduction into what front-end frameworks can do and the principles behind them. In this part we will mainly focus on the implementation of the application that will use the question generator algorithm.

Although the application is very minimalist in terms of UI/UX and design, the web application was carefully designed around the user needs and around the principle that everything should be simple to achieve and simple to find.

In the current chapter the application will be presented based on the capabilities and features. The first feature is the authentication. The moment when the user will visit the web site, he or she will be welcomed with a register form, or if the user is a returning one with the login form. Also, feature forget password is an available one. In the figure down below the first page of the application is presented.

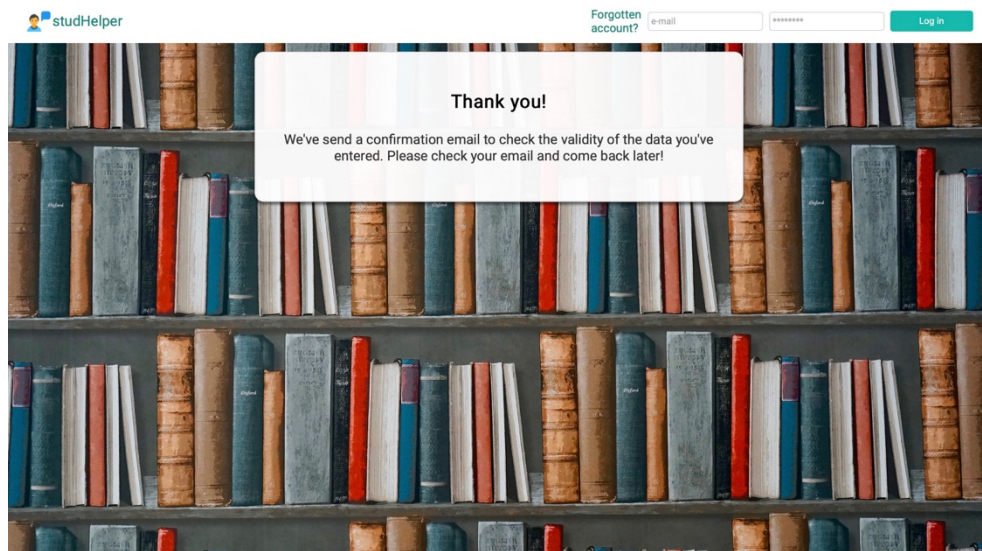
## CHAPTER 3 – CASE STUDY: STUDHELPER



The screenshot shows the 'Create an account!' form on the studHelper website. The form is centered on a background of bookshelves. At the top left is the 'studHelper' logo. At the top right are links for 'Forgotten account?' and 'Log in', with an 'e-mail' input field. The form itself has a title 'Create an account!' and a subtitle 'It is free and it will always be that way!'. It contains input fields for 'First Name:', 'Last Name:', 'Username:', 'E-mail:', 'Password:', and 'Confirm password:'. A green 'Create' button is at the bottom of the form.

FIGURE 3.1: INDEX PAGE OF THE APPLICATION

Moving forward, let's assume that the user completed the register form, then he or she will be redirected to a static page in which they are informed that, they have to check their email for an account validation, and this page is presented in figure 3.2, and the confirmation email is presented in figure 3.3



The screenshot shows the 'Thank you!' message on the studHelper website. The message is centered on a background of bookshelves. At the top left is the 'studHelper' logo. At the top right are links for 'Forgotten account?' and 'Log in', with an 'e-mail' input field. The message box has a title 'Thank you!' and a subtitle 'We've send a confirmation email to check the validity of the data you've entered. Please check your email and come back later!'. The background is a full-width image of bookshelves filled with books.

FIGURE 3.2: THANK YOU FOR REGISTRATION PAGE

## CHAPTER 3 – CASE STUDY: STUDHELPER

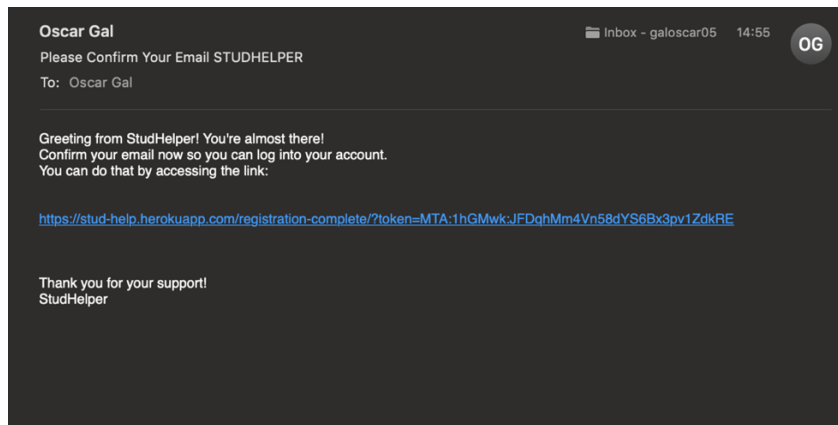


FIGURE 3.3: RECEIVED EMAIL FROM THE SERVER

After the registration was completed the user will be redirected to the homepage. While we are still explaining the authentication methods in the application, if you looked closed at the header in the figure 3.1 you could notice the log in form. The user completes the form and hit the log in button and if the credentials are correct the user will be redirected to the homepage.

Password reset is captured in figure 3.4, which is the forget password form that the user has to complete. In figure 3.5 the forget account email that the user will receive after completing the form previously presented. After the clicking the link in the email the user will be led to the password reset page, presented in figure 3.6.

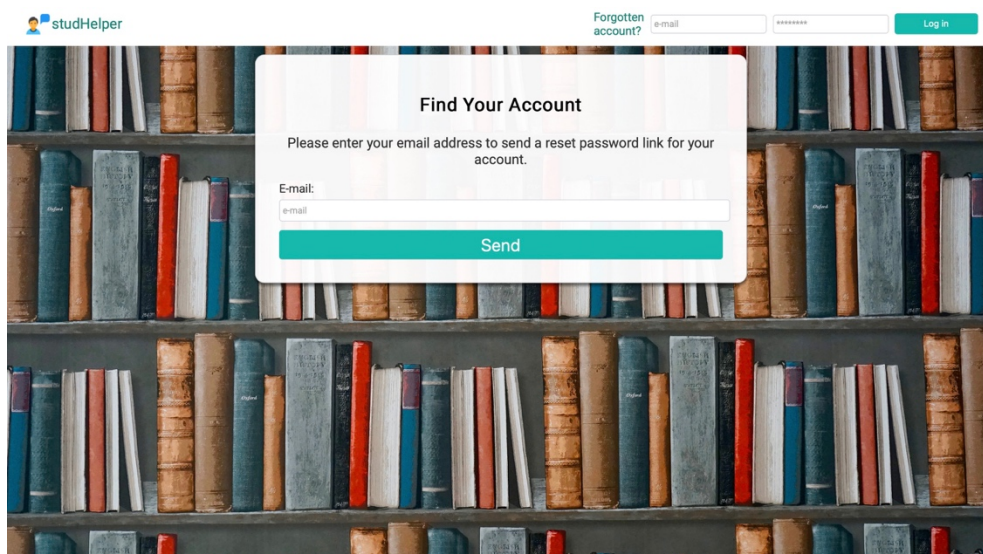


FIGURE 3.4: PASSWORD RESET FORM



## CHAPTER 3 – CASE STUDY: STUDHELPER

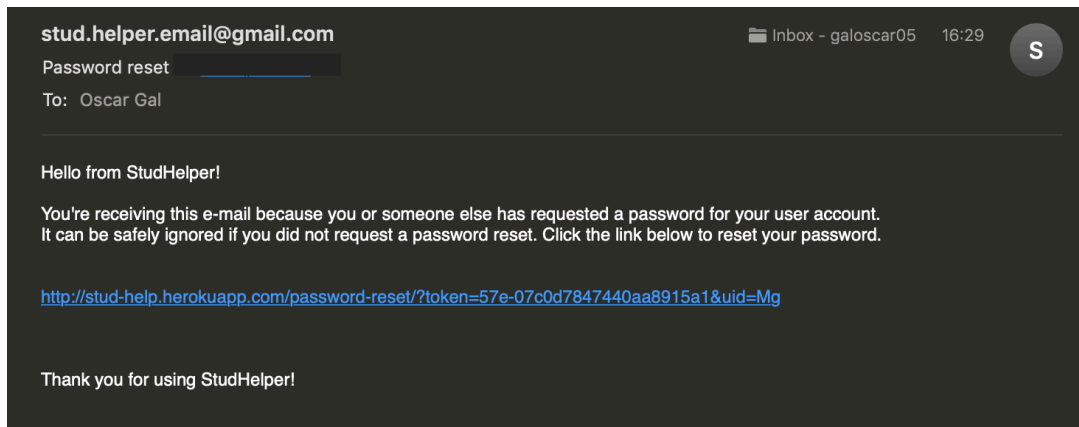


FIGURE 3.5: PASSWORD RESET EMAIL

A screenshot of a web application interface. The background is a bookshelf. In the center, there is a white modal box titled 'You are almost done' with the subtitle 'Reset your password'. Inside the modal, there are two input fields: 'Password:' and 'Confirm Password:', both with masked text. Below these fields is a green 'Submit' button. At the top of the page, there is a navigation bar with the 'studHelper' logo, a 'Forgotten account?' link, an email input field, a password input field, and a 'Log in' button.

FIGURE 3.6: PASSWORD CHANGE FORM

After the user successfully register/log in/reset his or her password, he or she has access to the homepage of the application which is presented in figure 3.7. On the homepage there are 3 actions possible. The first action is logging out by pressing the button “Sign Out” in the right corner. The second action is the uploading a new text into the account by pressing the document with a plus sign on it – the page for adding a new document is presented in figure 3.8. The third and final action is to enter on the “file details page” by clicking a document already added on the account; on this page the user can see the text and at the bottom of the page is presented the questions section. This page is presented in figures 3.9, 3.10 and 3.11.



## CHAPTER 3 – CASE STUDY: STUDHELPER

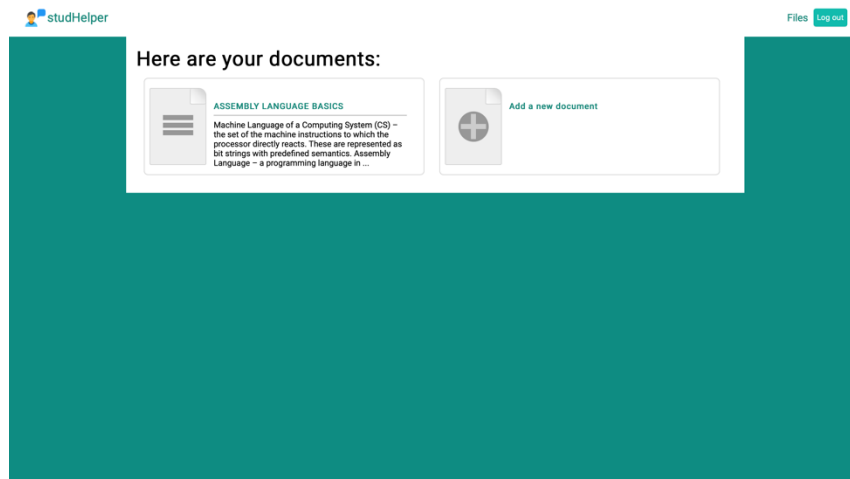


FIGURE 3.7: HOMEPAGE

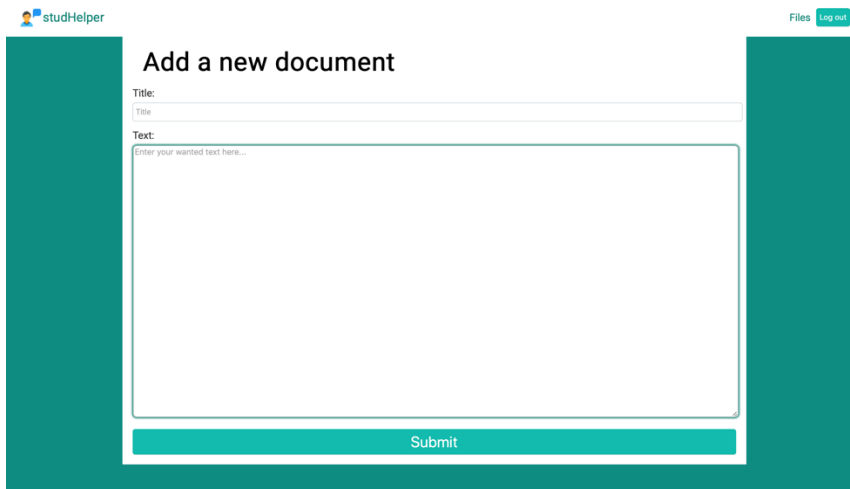


FIGURE 3.8 ADDING A NEW TEXT

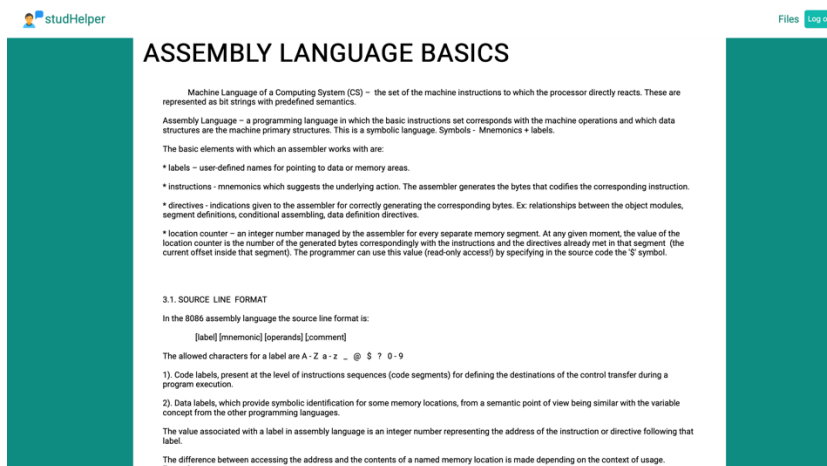


FIGURE 3.9: FILE PAGE WITH THE QUESTIONS PART 1

## CHAPTER 3 – CASE STUDY: STUDHELPER

The screenshot shows the StudHelper interface with a teal sidebar on the left and a teal sidebar on the right containing 'Files' and 'Log out' links. The main content area is titled 'Questions:' and contains three questions, each with a list of options:

- Question 1:** What provides symbolic identification for some memory locations?
  - Data labels
  - Code labels
  - Labels
- Question 2:** Which are the 2 mnemonics categories?
  - instructions names and directives names
  - Directives guide
  - Registers, constants
- Question 3:** What is the offset
  - a direct addressing operand
  - executable program structure
  - actual physical address

At the bottom of the questions is a teal button labeled 'Check answers!'.

FIGURE 3.10: FILE PAGE WITH THE QUESTIONS PART 2 (FORM NOT COMPLETED)

The screenshot shows the same StudHelper interface as Figure 3.10, but with feedback bars indicating the results of the answers:

- Question 1:** What provides symbolic identification for some memory locations?
  - Data labels (green bar)
  - Code labels (teal bar)
  - Labels (teal bar)
- Question 2:** Which are the 2 mnemonics categories?
  - instructions names and directives names (green bar)
  - Directives guide (red bar)
  - Registers, constants (teal bar)
- Question 3:** When is computed the offset of a direct addressing operand?
  - at assembly time (green bar)
  - at linking time (teal bar)
  - at loading time (red bar)

At the bottom of the questions is a teal button labeled 'Check answers!'.

FIGURE 3.11: FILE PAGE WITH THE QUESTIONS PART 2 (AFTER CHECKING ANSWER)

### 3.2 Future Improvements

The application was developed to work on specific cases (lectures from university), and this question generator can be extended to work for different cases, like articles from different sources, or even research papers from other fields like medicine or law.

Another huge update for the algorithm would be to add a neuronal network that will make the algorithm smart and capable of learning from mistakes. If we are talking about feature improvements on the question generator, there could also take place a lot of updates, for instance a first good step would be to make this algorithm work on personal essays.

On the application side, multiple updates can be added, for instance a flag that would mark wrong questions so they are never repeated, or even an input field for the user in which he or she could write the correct form of the question. Also, another huge update for the application could be adding computations in cloud for a faster question generation.

## **Conclusion and future work**

In this thesis, we presented the fundamentals of regex expression and natural language processing tools. We also showed how questions are formed in English and the steps to implement the algorithm that will generate questions. Experiments were conducted on various text inputs and data sets to make sure that the algorithm will be a useful one.

We believe that with the evolution in the field of artificial intelligence, a question generator could one day replace the way that people learn. Also by achieving the perfect algorithm one day the creation of tests and exams could also be made with the help of a question generator like this one, and this achievement would change the educational system in a better way.

# Bibliography

- [1] J. H. M. Daniel Jurafsky, Speech and Language Processing.
- [2] G. I. Schünz, Advanced natural language processing for improved prosody in text-to-speech synthesis, 2014.
- [3] "spaCy Documentation," spaCy, [Online]. Available: <https://spacy.io/usage>.
- [4] E. K. a. E. L. Steven Bird, "Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit," [Online]. Available: <https://www.nltk.org/book/>.
- [5] "Stanford Log-linear Part-Of-Speech Tagger," [Online]. Available: <https://nlp.stanford.edu/software/tagger.shtml>.
- [6] Kitlum, "The 5-step Guide to Forming Questions in English Grammar," [Online]. Available: <https://www.fluentu.com/blog/english/questions-in-english-grammar/>.
- [7] M. Heilman, Automatic Factual Question Generation from Text.
- [8] A. Kretch, "Question Generation: Using NLP to Solve “Inverse” Task," [Online]. Available: <https://medium.com/@aleckretch/question-generation-using-nlp-to-solve-inverse-task-a92ff033bcf1>.
- [9] G. B. Y. Z. Xuchen Yao, Semantics-based Question Generation and Implementation.
- [10] K. Lane, API Industry Guide On Api Design, 3Scale, 2015.
- [11] M. Rouse, "TechTarget," [Online]. Available: <https://searchmicroservices.techtarget.com/definition/RESTful-API>.
- [12] I. Bodrov-Krukowski, "Angular Introduction: What It Is, and Why You Should Use It," [Online]. Available: <https://www.sitepoint.com/angular-introduction/>.
- [13] R. Fielding, Representational State Transfer (REST).
- [14] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures.
- [15] Design, <http://students.cs.ucl.ac.uk/2016/group19/design/>.
- [16] "Django Documentation," Django, [Online]. Available: <https://docs.djangoproject.com/en/2.2/>.
- [17] MkDocs, "Django Rest framework documentation," [Online]. Available: <https://www.django-rest-framework.org/api-guide/requests/>.
- [18] "Hacker News," Snippod Inc., [Online]. Available: <https://github.com/shalomeir/snippod-starter-demo-app>.
- [19] "Angular Documentation," Google, [Online]. Available: <https://angular.io/docs>.
- [20] D. M. Elkstein, "Rest Priciples," [Online]. Available: <http://rest.elkstein.org>. [Accessed 1 June 2019].
- [21] S. B. Avraham, "What is REST — A Simple Explanation for Beginners," [Online]. Available: <https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-1-introduction-b4a072f8740f>.
- [22] P. Podila, "HTTP: The Protocol Every Web Developer Must Know," [Online]. Available: <https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177>.