

BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA  
FACULTY OF MATHEMATICS AND  
COMPUTER SCIENCE  
SPECIALIZATION COMPUTER SCIENCE IN  
ROMANIAN

## DIPLOMA THESIS

# Automating the Question Generation Process in Online Quiz Applications

Supervisor  
Lector Dr. Ioan Gabriel MIRCEA

*Author*  
*Diana Elena PÎRVU*

2021



---

## ABSTRACT

---

The subject of this thesis is the domain of Natural Language Processing(NLP). This thesis tackles the problem of Question Generation(QG) from a given input text, which is still an area under research.

The following thesis is structured in three chapters. The first chapter gives an overview of the approaches used so far, from rule-based systems to Deep Learning models. The second chapter aims to explain the solution used in this thesis. The last chapter is focused on the software implementation, providing details about the development methodology used.

The main element of originality of this thesis consists in designing and creating a web application aimed for students and teachers that offers the functionality of generating a set of questions from a given input text, as to my knowledge there is no such application publicly available yet.

This work is the result of my own activity. I have not received nor given unauthorized help.

Pîrvu Diana Elena

Cluj-Napoca

2021-06-16

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Background of Question Generation</b>	<b>2</b>
1.1 Question Generation . . . . .	2
1.2 Natural Language Processing . . . . .	3
1.3 Deep Learning . . . . .	3
1.3.1 Transfer Learning . . . . .	5
1.4 Literature Review . . . . .	5
1.4.1 Rule-based systems . . . . .	5
1.4.2 Deep Learning . . . . .	6
1.4.3 Reinforcement Learning . . . . .	7
1.4.4 Conclusions . . . . .	7
<b>2 Methodology</b>	<b>9</b>
2.1 Natural Language Processing . . . . .	10
2.2 T5 . . . . .	10
2.3 Training . . . . .	11
2.3.1 SQuAD . . . . .	11
2.3.2 Data Pre-processing . . . . .	11
2.3.3 PyTorch Lightning . . . . .	12
2.3.4 Optimization . . . . .	13
2.4 API Documentation . . . . .	14
2.4.1 Flask . . . . .	14
2.4.2 Endpoints . . . . .	14
2.5 Testing . . . . .	15

---

2.5.1	ROUGE-L . . . . .	15
2.5.2	METEOR . . . . .	15
2.5.3	Results . . . . .	16
<b>3</b>	<b>Software Design and Implementation</b>	<b>17</b>
3.1	Software Design . . . . .	17
3.1.1	Use Cases . . . . .	17
3.1.2	Class Diagrams . . . . .	18
3.1.3	Sequence Diagrams . . . . .	18
3.1.4	Design Patterns . . . . .	20
3.2	Software Implementation . . . . .	22
3.2.1	Backend . . . . .	22
3.2.2	Node.js . . . . .	23
3.2.3	Persistence . . . . .	24
3.2.4	Frontend . . . . .	25
3.2.5	Ionic React . . . . .	26
3.2.6	Testing . . . . .	26
3.3	User Manual . . . . .	27
3.3.1	Log In . . . . .	27
3.3.2	Sign Up . . . . .	27
3.3.3	Home Page . . . . .	28
3.3.4	Creating a classroom . . . . .	29
3.3.5	Quizes . . . . .	30
3.3.6	Creating a quiz . . . . .	30
3.3.7	Attempts and evaluations . . . . .	31
3.3.8	Summary . . . . .	32
	<b>Conclusions and Future Work</b>	<b>33</b>
	<b>Bibliography</b>	<b>34</b>

---

# List of Figures

1.1	A neural network with 3 hidden layers. Source:[12]	4
2.1	The main steps that have been taken in this approach	9
2.2	Google's T5 can be used for various Text-to-Text problems	10
2.3	Comparison between PyTorch and PyTorch Lightning	13
2.4	An example of a valid request	14
3.1	Diagram of use cases	18
3.2	Class diagram	18
3.3	Create Quiz use case	19
3.4	Generate Quiz use case	19
3.5	Attempt Quiz use case	20
3.6	Evaluate Attempt use case	20
3.7	Factory pattern in the backend part of the application	22
3.8	Components diagram	23
3.9	The most popular web frameworks in 2020, according to StackOverflow	25
3.10	Authentication screen	27
3.11	New user registration screen	28
3.12	Home screen of a professor account	28
3.13	Home screen for a student account	29
3.14	Screen for creating a new classroom	29
3.15	Classroom screen	30
3.16	Screen for creating a quiz manually	31
3.17	Screen for generating a quiz from text	31
3.18	Application screen with the attempts of a quiz	32
3.19	Attempt evaluation screen	32

# Introduction

Question Generation(QG) aims to generate natural language questions[Yu 20] based on given contents, where the generated questions need to be able to be answered by the contents. It is a problem where many researchers have presented their work and better results are yet to be achieved.

Having an application that could extract questions from a given text might be useful in the area of text comprehension. Comprehension requires skills like vocabulary, background knowledge and reasoning, and answering questions is not only a legit way of evaluating the comprehension skills, but also a good way of learning.

The aim of this thesis is to present an application for online evaluations that makes use of modern techniques of Natural Language Processing(NLP) in order to automate the process of generating a set of questions from a given input text. The problem of QG is approached making use of Google's T5 model, that was trained on the task of generating questions, using the SQuAD dataset.

The thesis is structured as follows. The first chapter gives an overview of the topic of Question Generation and the approaches that have been used so far. The second chapter outlines the main theoretical concepts of NLP(keyword extraction, ), Transfer Learning(TL) and optimization used in this project. The last chapter presents the web application that has been implemented using modern responsive frameworks, in order to make use of the presented QG model. Both the software design and the implementation are detailed in this chapter, that contains the diagrams of the application and an user manual.

# Chapter 1

## Background of Question Generation

This chapter aims to give the reader an overview of the topic of Question Generation(QG). The first sections focus on the technical aspects that have been used to approach the subject, while the last section gives a review of the most relevant works that have been done so far in this domain.

### 1.1 Question Generation

Question Generation(QG) is a task complementary with Question Answering(QA) where the aim is to generate questions from an input text that contains the answer or from where the answer can be inferred. It is a Natural Language Processing problem where many researchers have presented their work. Higher accuracy is yet to be achieved. [6]

There are several types of questions that can be generated:

- Factoid questions: questions that start with "What", "Who", "Where", "How much", "Which" or "When".
- Open cloze, or "fill-in-the-blank" questions.
- "True or false" questions.
- Single or multiple choice questions.

Generating questions from a given text may be useful to assess the reading comprehension and to help the reader understand the main points of the text by answering



questions about the given text. Having a system that automates the Question Generation process makes it easier to create an evaluation quiz for students. Also, being a complementary task to Question Answering(QA), Question Generation(QG) may be helpful in improving chat-bots or personal assistants(e.g Siri, Alexa, Cortana etc). Available studies revealed that humans are not very skilled in asking good questions, therefore the assistance of a QG system might be useful in meeting their inquiry needs. [Hus10].

## 1.2 Natural Language Processing

Natural language processing(NLP) [9] aims to build programs that understand and respond to voice or text data- and respond with text or speech of their own—in a way close to how humans do. A lot of NLP tasks transform human text in ways that help the computer process it [9]. Some of these tasks include:

- Part-of-Speech tagging, which aims to determine the part of speech of a particular word or piece of text based on its use and context
- Word sense disambiguation, which is the selection of the meaning of a word based on syntactical analysis
- Sentiment analysis, which identifies and extracts emotions from the text
- Natural language generation, which is the task of turning structured data into human language

One of the tasks of NLP that will be emphasized in this thesis is natural language generation. There are many use cases of this area, such as machine translation, sentiment analysis, chatbots and text summarization.

## 1.3 Deep Learning

Deep learning(DL) is a subset of machine learning in which multi-layered neural networks—modeled to work like the human brain- learn from large amounts of data. [13]

The essential DL models are the feedforward networks or multilayer perceptrons (MLPs). The purpose is to approximate a function  $f^*$ . A feedforward network takes an input  $x$  and an output  $y$ , and finds a mapping  $y = f(x, \theta)$  and learns the parameters  $\theta$  that give the best approximation. [GBC16] The model is associated with a directed acyclic graph describing how the functions are composed together.

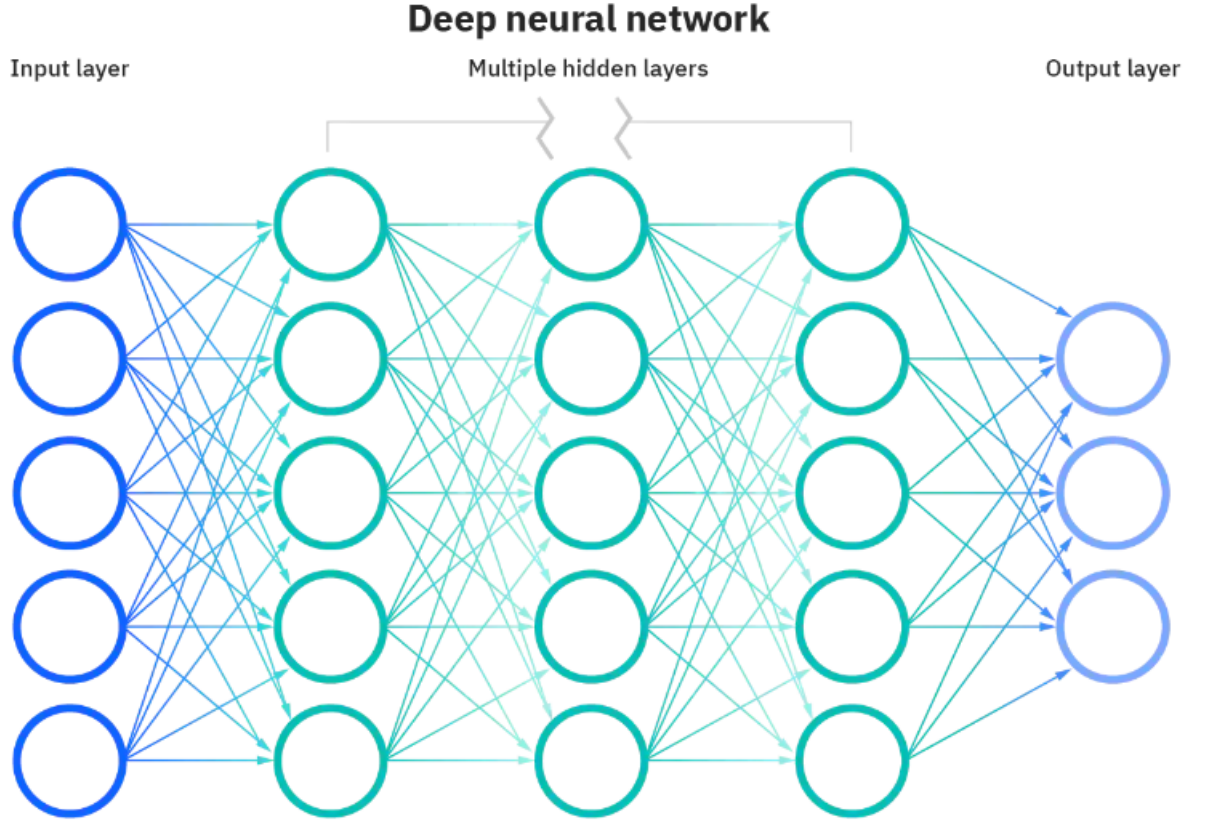


Figure 1.1: A neural network with 3 hidden layers. Source:[12]

Over time, multiple variations of networks appeared, in order to solve specific types of problems. Some renowned examples include the recurrent neural networks (RNNs), which are a type of MLP that have been extended to include feedback connections, and the convolutional neural networks (CNNs), which are especially used in object recognition and image processing.

The large need of computing power and data come as inconveniences of DL, and an alternative that has emerged to be more popular is Transfer learning.

### 1.3.1 Transfer Learning

In Transfer learning(TL) a model is first pre-trained on a data-rich task before being fine-tuned on a downstream task. [Col20] The pre-trained network serves as transferred knowledge to be applied in another domain. But there are numerous options that can be used, including feature transfer and fine-tuning (which depend upon the similarity of the problems at hand), in addition to freezing certain layers of the network and retraining others.[8]

## 1.4 Literature Review

After analyzing many papers that have aimed to cover this subject, a preliminary conclusion is that the more recently published papers tend to be more relevant as they tackle the problem of Question Generation by using Reinforcement Learning(RL) or Deep Learning(DL) methods such as Long Short Term Memory(LSTM), attention mechanism and Gate Recurrent Unit(GRU), unlike the older approaches, that used heuristics and manually-designed rules. The newer approaches also tend to generate more natural questions, getting closer to human level performance.

### 1.4.1 Rule-based systems

These systems are usually implemented using Tregex, a tree query language, and Tsurgeon, a tree manipulation language built on top of Tregex [14]. One of the strategies, mentioned in Michael Heilman’s work in 2009, was to overgenerate questions and then rank them. The system’s performance was evaluated by 15 native English-speakers, who indicated whether each question exhibited any deficiency(such as not being grammatically correct, being vague, missing answer, having an obvious answer, using the wrong WH- word or having formatting errors). [Hei09]

In a later work of Heilman, published in 2011, the focus is on generating two types of questions: concept completion questions and verification questions.[Hei11] Concept completion questions elicit a particular information that completes a given partial concept or proposition (e.g., Who served as the first Secretary of State under George Washington?). Verification questions invite a yes-no answer that verifies given information (e.g., Was Thomas Jefferson secretary of state?). The system follows an ‘overgenerate

and rank’ approach, using Stanford Parser’s API and the lexicalized version of the default grammar (*englishFactored.ser.gz*) that is provided with the parser.[Hei11]

Another paper, published by [Hus10], considered a form of Text-To-Question generation task where the input text are sentences. Only the factoid questions were considered (questions of any of these types: "What...?", "Where...?", "When...?", "Who...?" and "How many/much...?"). The dataset provided by TREC 2007 Question Answering Track was used. The algorithm was evaluated by measuring the Recall and the Precision. Oak system was used in order to tokenize the sentences and the questions. The algorithm works by extracting the elementary sentences from the input, classifying them based on subject, verb, object and generating questions using a predefined set of rules.

### 1.4.2 Deep Learning

The rise of Deep Learning in the last decade lead to many researchers aiming to solve the problem of Question Generation with more scalable methods than the previous ones.

Du et al. [Xin17] tackled the problem using a conditional neural language model with a global attention mechanism and the *Glove 840B 300 d* pre-trained embeddings, built with Torch7 on OpenNMT.

Yao et al. [YZL<sup>+</sup>18] modeled a probabilistic distribution over the potential questions using a latent variable. This allowed the generation of diverse questions by drawing samples from the learned distribution and reconstructing the words sequence via a decoder neural network. Adversarial training was applied. The model was deployed in GAN (Generative Adversarial Network) framework which consists of a generative model G to generate more readable and diverse questions. SeqGAN models the text generation as a sequential decision making process, utilizing policy gradient methods.

Liu et al. [LZN<sup>+</sup>19] proposed a Clue Guided Copy Network for Question Generation (CGC-QG), which was a sequence to sequence (seq-to-seq) generative model with copying mechanism. In CGC-QG, a multi-task labeling strategy was designed to identify whether a question word should be copied from the input passage or be generated instead, guiding the model to learn the accurate boundaries between copying and generating.

Xiao et al. [XZL<sup>+</sup>20] tried to approach the problem of exposure bias using an enhanced multi-flow seq-to-seq pre-training and fine-tuning framework named ERNIE-GEN which bridges the discrepancy between training and inference with an infilling generation mechanism and a noise-aware generation method.

In a paper published by Lopez et al. [Lui20] the researchers used transformer-based finetuning techniques and only one pretrained language model without any additional mechanisms in order to create a simple system that achieves state-of-the-art results. They trained the question generation model on version 1.1 of the Stanford Question Answering Dataset (SQuAD)[RZLL16]. For the base pretrained model, they used HuggingFace’s implementation of the GPT-2 with 124 million parameters(smallest of the 4 available GPT-2 model sizes). After testing, it was observed that 20 samples from the generated question set were non-questions, generated in ”failure mode”: either the last 3 words kept repeating or the generated question was cut prematurely.

### 1.4.3 Reinforcement Learning

The problem of QG using Reinforcement Learning was tackled by Yu et al. [Yu 20], trying to address the limitations of previous approaches that don’t solve the exposure bias problem or fail to exploit the information given in the answer or in the context. The proposed model consists of a graph-to-sequence(Graph2Seq) generator with a Bidirectional Gated Graph Neural Network based encoder, and combining RL and cross-entropy losses to generate valid text. It also uses a Deep Alignment Network to incorporate the answer information in the text.

### 1.4.4 Conclusions

In the previous subsections, we presented some notable approaches to the problem of question generation. It can be noticed that the results of the researchers improved with time, starting from the rule-based systems, and evolving to deep learning and reinforcement learning. The rule-based systems have the disadvantage that the programmer has to write manually all the rules for the system, which would make the system more difficult to extend. The deep learning and reinforcement learning approaches don’t face this problem, since the model learns the rules. The reinforcement learning approaches to the problem of question generation are relatively new, while the

deep learning methods have been used more often, especially in combined with transfer learning in the more recent papers.

# Chapter 2

## Methodology

This chapter introduces the approach that has been used to generate a set of questions from a given input text. The following sections present the theoretical aspects of the implemented QG model. As shown in the figure 2.1, a given input text is first pre-processed (we remove the special characters and split it into phrases), then we extract the main keywords (which are always nouns or proper nouns) and organize them into a map in which the key is the keyword and the value is its context (the chunk of text in which it is used). Each entry is then fed to the QG model, that returns at least one question based on the context for which the answer is the keyword.

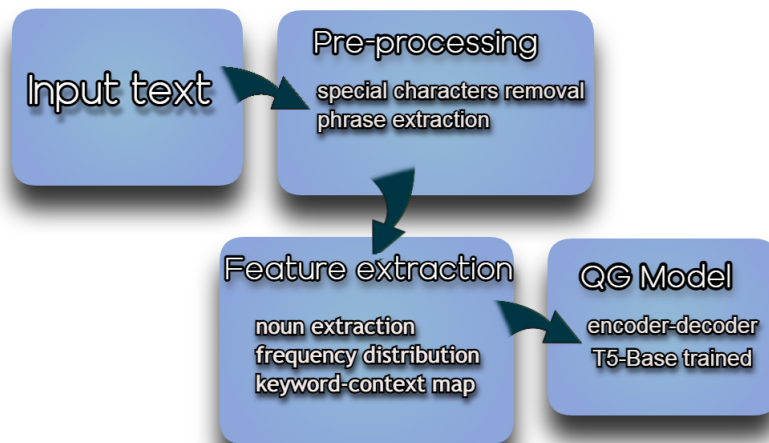


Figure 2.1: The main steps that have been taken in this approach

## 2.1 Natural Language Processing

This section aims to present the preliminary steps that have been taken in order to create a map of keywords-to-context.

### Input Pre-processing

The first step is to clean the received input text, by removing special characters using regular expressions. Also, tokenization is used in order to divide the text into sentences. The sentences that are shorter than 20 characters are excluded, as they are irrelevant for the task.

### Keyword Extraction

The next step is to extract the keywords from the text. First of all, for each phrase, we are choosing the nouns and proper nouns, since only these can constitute keywords. Out of these, we will find the most relevant ones, that will constitute the keys of the map. The nouns are extracted in an unsupervised manner, using a multipartite graph, as proposed by [Flo18], and sorted after the frequency distribution.

## 2.2 T5

T5 [Lui20] is an encoder-decoder model that has been pre-trained on a variety of natural language processing problems, reducing each one of them to a text-to-text problem. T5 can then be trained on a specific task, as it was in this thesis, where it was trained to generate question.

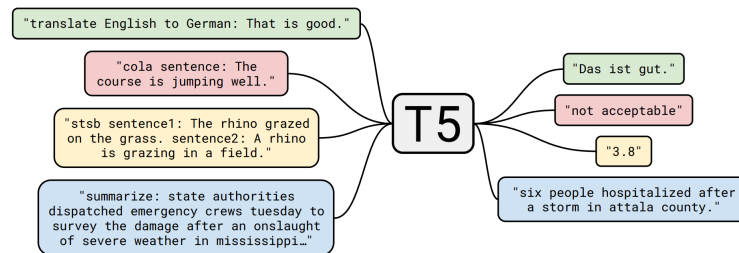


Figure 2.2: Google's T5 can be used for various Text-to-Text problems

As mentioned earlier, the basic input for the model is a keyword(which will be the answer), and the chunk of text were that keyword appears and from which a question can be formulated.



The model has 12 normal layers, 12 decoder layers and uses Rectified Linear Unit(ReLU) as feed forward function.

$$ReLU(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

The Dropout rate is 0.1 and the Normalization layers use a variance of  $\epsilon = 1e - 6$ .

## 2.3 Training

This section presents the training process that has been carried on. The t5-base model was trained for the question generation task on 50000 entries of the SQuAD dataset(due to hardware limitations), using PyTorch Lightning, during one epoch and optimized using Adam optimizer.

### 2.3.1 SQuAD

Stanford Question Answering Dataset(abbreviated SQuAD) [7] is a data set used for reading comprehension, consisting of questions posed of a variety of articles from Wikipedia, where the answer to every question can be found in the text.

Although initially this data set was proposed for training Question Answering(QA) models, it can be applied on the task of QG as well. SQuAD is already split into training, development and testing sets(80/10/10).

The main drawback of this data set is the lack of variety. While SQuAD-trained models perform well, their performance does not generalize to other types of questions than SQuAD's factoid-based ones, that start with 'What'. Generating questions that start with 'Why' or 'How', for instance, rely more on understanding the context and less on copying chunks from the given text.

### 2.3.2 Data Pre-processing

The SQuAD dataset comes in JSON(Javascript Object Notation) format, each sample being having the following fields

- Context - The fragment of text that contains the answer.

- Question - A question posed based on the context.
- Id - the unique identifier of the sample
- Title - the title of the source Wikipedia article
- Answers - an array that contains the text of the answer and the start position in the text

Since not all of the information is essential for this problem, we extracted only the context, the answer and the question from each sample and saved the resulting dataset in CSV files.

### 2.3.3 PyTorch Lightning

Python is an open source programming language, widely used in Artificial Intelligence tasks due to the variety of helpful libraries that it provides. PyTorch Lightning is an open-source Python library that provides a high-level interface for PyTorch. It is designed to make deep learning experiments easier. Models created in PyTorch extend the class *nn.Module*, and implementing the forward method, the training step and the validation step. The main disadvantage of PyTorch compared to Lightning is that the code is just organized, but not abstracted. Lightning solves this problem by providing the *LightningModule*, which is a better structured base class for models, providing methods for loading the dataset, configuring optimizers, and for the execution of the training step, thus facilitating the coding process [1]. A parallel between PyTorch and PyTorch Lightning is shown in the figure 2.3.

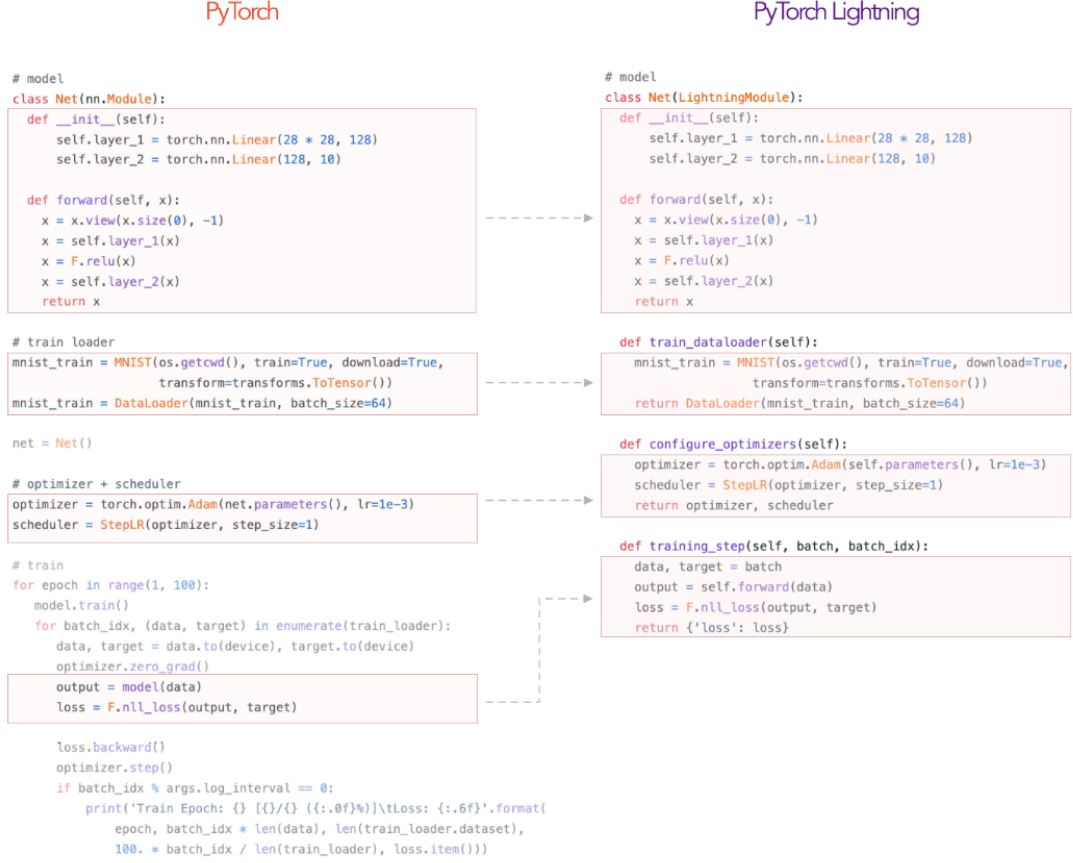


Figure 2.3: Comparison between PyTorch and PyTorch Lightning

The input sequence is fed to the model using *input\_ids*. The target sequence is shifted to the right, prepended by a start-sequence token and fed to the decoder using the *decoder\_input\_ids*. The model also uses attention mask. The attention mask is an optional argument used when batching sequences together. This argument indicates to the model which tokens should be attended to, and which should not. [1]

### 2.3.4 Optimization

The optimization algorithm used in this approach, as mentioned in the beginning of the section, is Adam(Adaptive Moment Estimation), that uses a combination of ideas from other optimizers. In this situation, the learning rate was set to  $\lambda = 3e - 4$  and the smoothing term(to prevent division by zero) was  $\epsilon = 1e - 8$ .

## 2.4 API Documentation

This section presents the REST API that has been implemented in order to make use of the model presented above.

### 2.4.1 Flask

Flask is a lightweight web framework written in Python, that has been used in order to facilitate the communication between the REST API written in Node.js and the question generator script that was written in Python. Thus, in the use case of generating questions automatically, the main server sends a POST request to the Flask server. The result returned by the Python script is sent back to the main server, and then to the client.

### 2.4.2 Endpoints

The implemented API has one end-point, which can be accessed by sending a POST request to *localhost:5000/qgen*, including the input text in the body of the request, as a parameter named *inp*. The result is returned as an array of strings, representing the generated questions.

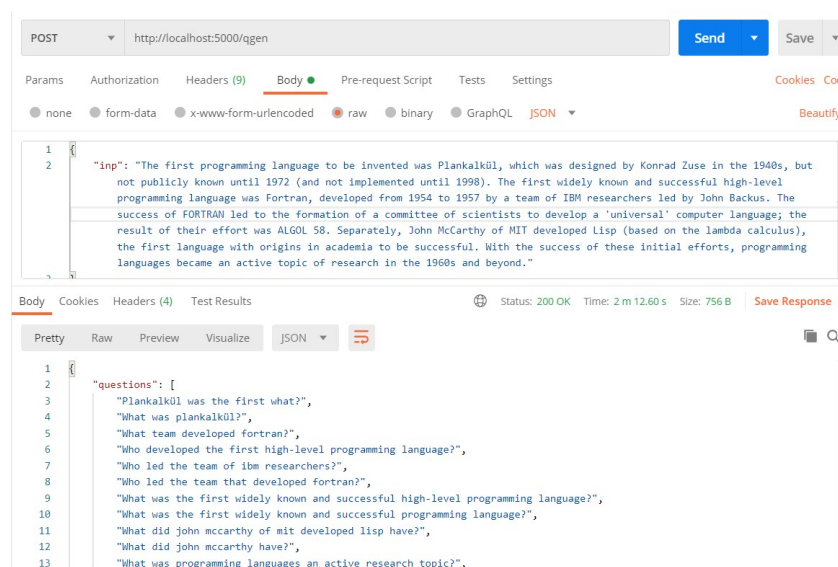


Figure 2.4: An example of a valid request

## 2.5 Testing

As mentioned in the papers presented in the previous chapter, the performance of a language model is measured in various metrics, such as the ROUGE-L and METEOR scores.

### 2.5.1 ROUGE-L

As explained by [Lin04], ROUGE-L estimates the similarity between two texts by using the F-score computed based on the longest common subsequence. Thus, given two texts  $X$  and  $Y$  and  $LCS(X, Y)$  being the length of the longest common subsequence, ROUGE-L equals to 1 when the two texts are equal and to 0 when  $LCS(X, Y) = 0$ .

Let  $R_{LCS}$  be the recall and  $P_{LCS}$  the precision. If  $m$  is the length of the first text, and  $n$  is the length of the second text, then the recall and the precision are calculated as:

$$R_{LCS} = \frac{LCS(X, Y)}{m}$$

$$P_{LCS} = \frac{LCS(X, Y)}{n}$$

Using the formulae above, the F-score is calculated as follows:

$$F_{LCS} = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS}+\beta^2P_{LCS}}, \text{ where } \beta = \frac{R_{LCS}}{P_{LCS}}.$$

### 2.5.2 METEOR

METEOR, as presented by [BL05], is a metric for machine translation evaluation and is evaluating the performance of the language model by computing a weighted F-score and a penalty function for incorrect word order. For this, we look at exact matches, followed by matches after Porter stemming[Por97], and finally using WordNet synonymy. After such an alignment is found, suppose  $m$  is the number of mapped unigrams between the two texts. Then, precision and recall are given as  $m/c$  and  $m/r$ , where  $c$  and  $r$  are candidate and reference lengths, respectively.

$$METEOR = F_{score} * (1 - Penalty)$$

Where  $F_{score}$  and  $Penalty$  are calculated as follows:

$$F_{score} = \frac{10PR}{R+9P}$$

$$Penalty = 0.5\left(\frac{\#chunks}{\#matches}\right)^3$$

### 2.5.3 Results

The METEOR and ROUGE-L scores have been measured on entries from the SQuAD dataset, the average resulting scores being 39.85 and 35.65, respectively, not achieving state-of-the-art performance in terms of METEOR score.

ERNIE-GEN [XZL<sup>+</sup>20] achieved a higher score on ROUGE-L, of 53.77, but a lower score on METEOR(of only 27.57). Liu et al [LZN<sup>+</sup>19] achieved a score of 44.53 on ROUGE-L and 21.24 on METEOR.

# Chapter 3

## Software Design and Implementation

This chapter will present the application proposed, *AutoQuest*, that makes use of the QG model presented in the previous chapter. The first section of this chapter focuses on the process of software design, that is the theoretical part of developing an application. The second section presents the technologies that have been used for the software implementation process. The last section provides a user manual.

### 3.1 Software Design

This section will give an overview to the process of designing an application that provides the user with the functionality of generating a set of questions(a quiz) from a given input text. The application provides use cases for two types of users: student and professor. Generally, a professor is able to create a group(classroom), add other members, post quizzes and evaluate the attempts of other users.

#### 3.1.1 Use Cases

When developing a software, the first step is to understand the problem by gathering the function requirements of the application. This can be done with the help of a use case diagram.

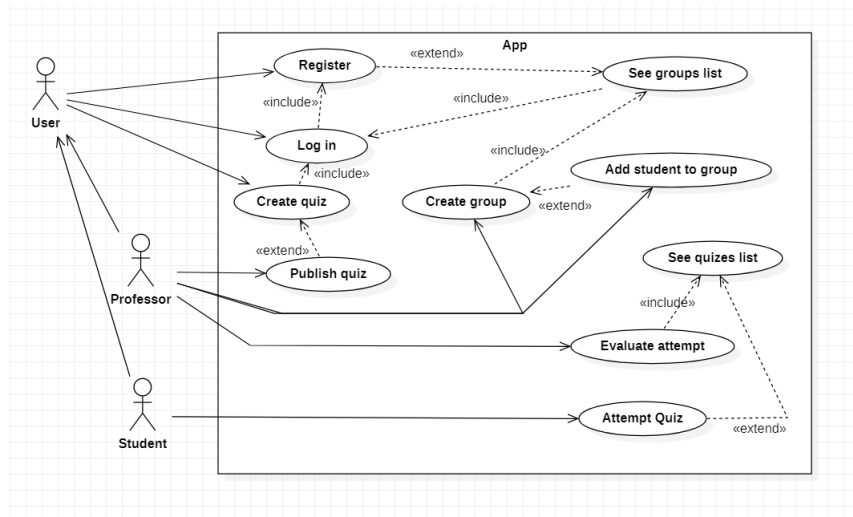


Figure 3.1: Diagram of use cases

### 3.1.2 Class Diagrams

A class diagram is used in order to depict a static, structural view of the application. It shows the objects that compose the system and the relationships between them. It presents the classes, attributes, methods(operations) as well as the relationships between the classes involved in the system.

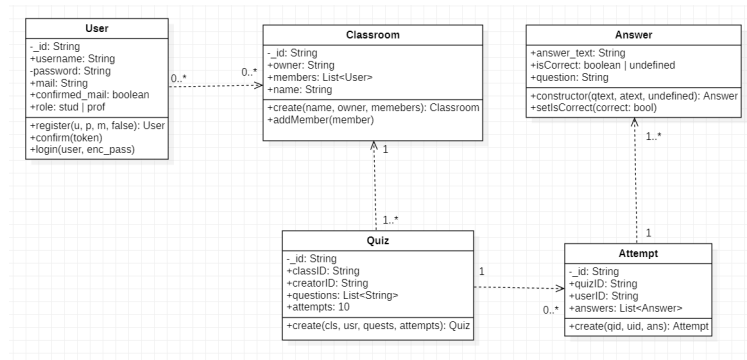


Figure 3.2: Class diagram

StarUML was used to build the class diagram for this project, as well as the sequence diagrams and the use case diagrams.

### 3.1.3 Sequence Diagrams

The sequence diagram illustrates the flow of the messages in the application for each use case. It is helpful in order to visualize several dynamic scenarios. It portrays



the communication between any two lifelines(represented by vertical bars) as a time-ordered sequence of events, such that these lifelines took part at the run time. Below are the sequence diagrams for the main use cases that have been mentioned previously in the use case diagram.

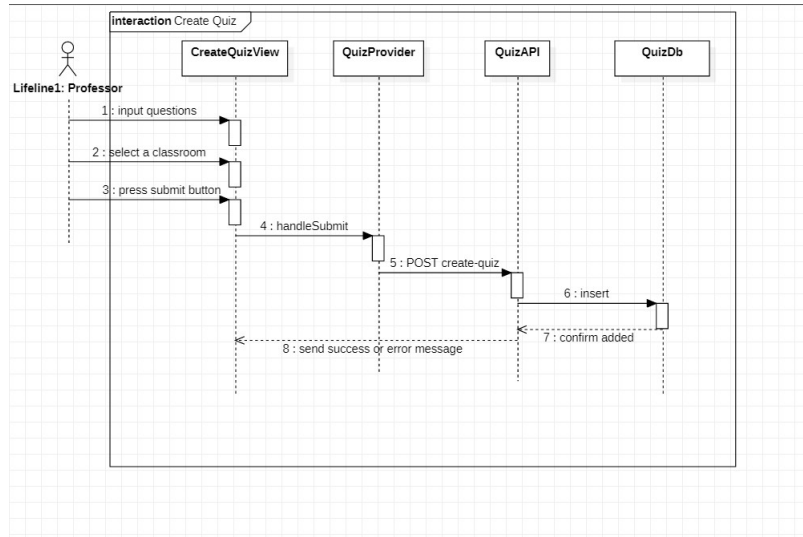


Figure 3.3: Create Quiz use case

The figure 3.16 presents the use case of creating a quiz manually, by writing each question. The save endpoint is used when generating a quiz, as shown in the figure 3.4, after the user selects the relevant questions that have been generated and sends the request to the server.

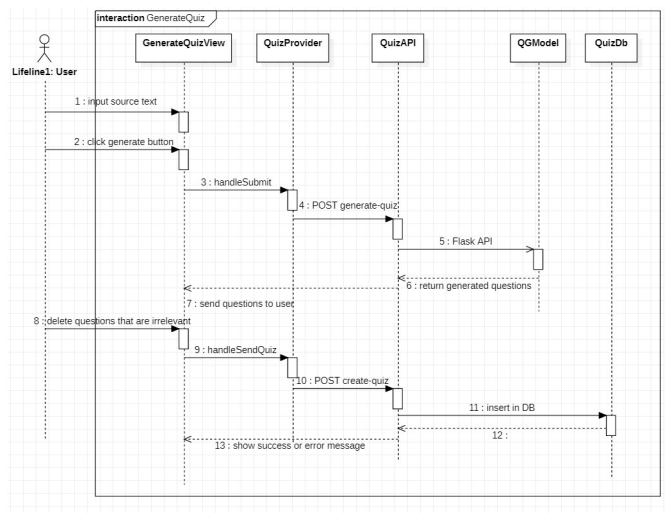


Figure 3.4: Generate Quiz use case

The figure 3.5 shows the flow of attempting a quiz. Each question is displayed individually and the user only navigates forwards through the list of questions.

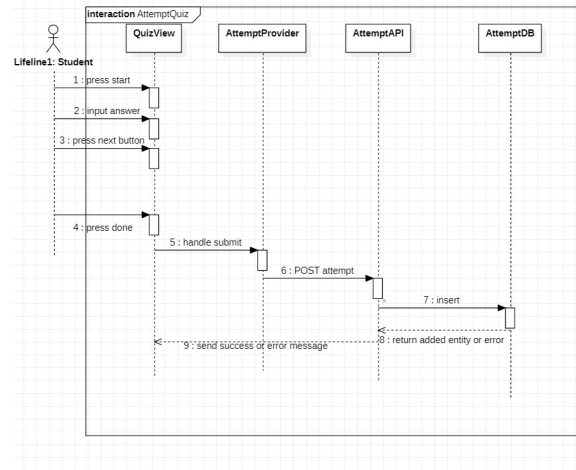


Figure 3.5: Attempt Quiz use case

The use case of evaluating an attempt is shown in figure 3.6. A professor has to log in the account and evaluate all the answers in the attempt.

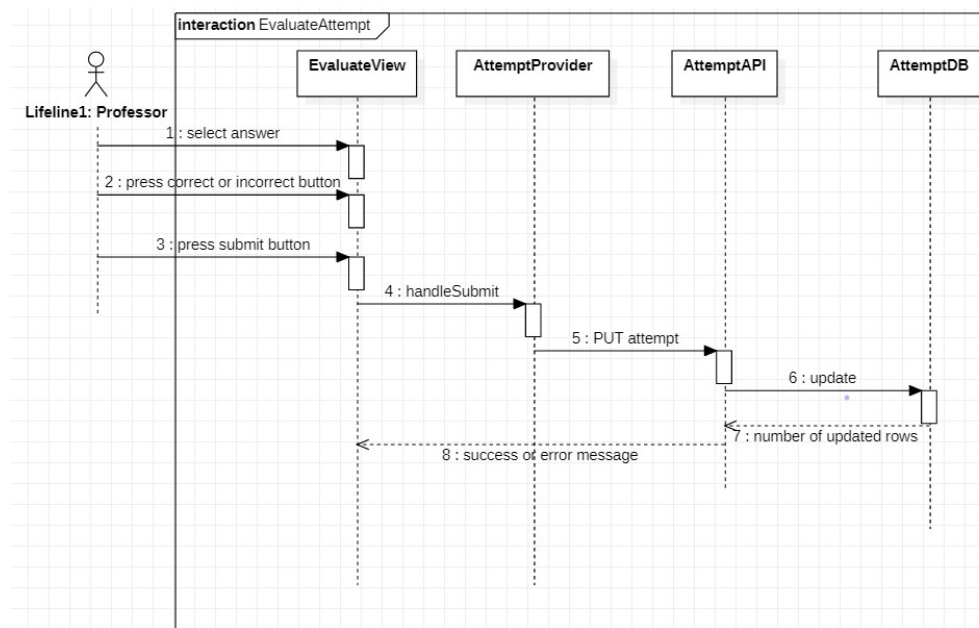


Figure 3.6: Evaluate Attempt use case

### 3.1.4 Design Patterns

In object-oriented programming, design patterns play a key role in writing reusable code and keeping the application organized, thus making it easier to extend the appli-

cation later. Design patterns are used in order to identify and abstract the key aspects of a common structure in order to create a reusable object-oriented design. [Eri94] This section presents the design patterns that have been used in the proposed application.

### Singleton

According to Gamma et al, [Eri94], the purpose of the Singleton pattern is to ensure that there is only one instance of a given class, that can be globally accessed in the application.

Example:

---

```
class SingletonExample{
    private static SingletonExample obj;
    private SingletonExample(){
    public getInstance(){
        if(obj == null)
            obj = SingletonExample();
        return obj;
    }
}
```

---

### Factory

The Factory pattern is used in order to create an instance of an object without exposing the means of creation to the user. It can be coupled with the Singleton pattern, thus having only an instance of the factory object in the application. An example of the Factory pattern applied can be seen in the figure 3.7.

In this application, Singleton and Factory have been used on the backend side. Due to the fact that the persistence classes have many common methods, they all extend a base class and are instantiated using a factory.

### Model View Controller(MVC)

The MVC pattern is used to make a delimitation between the application's concerns. It involves three main parts of the application:

- **Model.** The model represents the classes that carry data.
- **View.** The view represents the part of the application responsible with data visualization and the sum of all the graphic components that the user directly

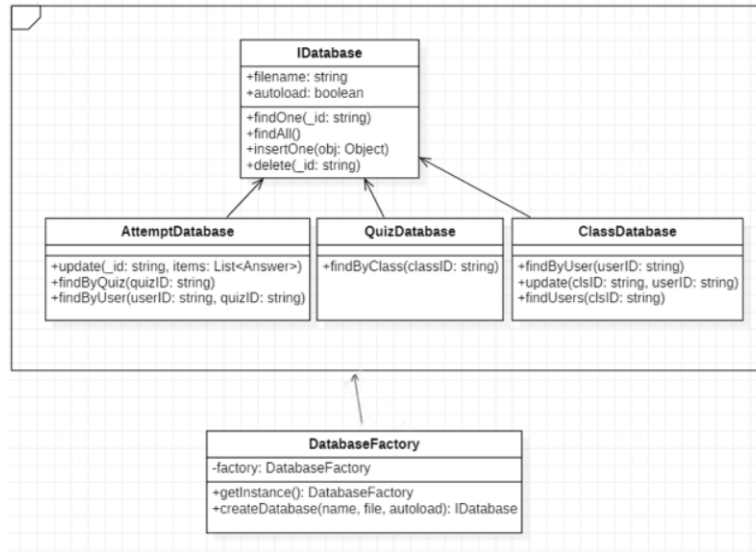


Figure 3.7: Factory pattern in the backend part of the application

interacts with(buttons, input fields, modals, tables, etc).

- **Controller.** The controller keeps the view and the model separate. The controller handles the requests made by the user of the view upon the model.

## 3.2 Software Implementation

The implementation is a significant phase in the process of software development because of the decisions upon development technologies and testing methods. This section aims to present the aspects of the implementation process and the technologies used for frontend, backend and persistence.

The application proposed is a client-server one, that involves a client side which communicates with a server side composed of two servers: a main server, written in Node.js, which handles the majority of the requests from the client, and a secondary server, written in Flask, that runs the question generation model for the input received.

### 3.2.1 Backend

Backend is the term used to refer the parts of an application with which the user doesn't interact directly. It is usually composed of a server which handles the requests made by the user(most commonly through a REST service), the access to a database, file uploads or encryption and decryption of data. Backend technologies usually con-

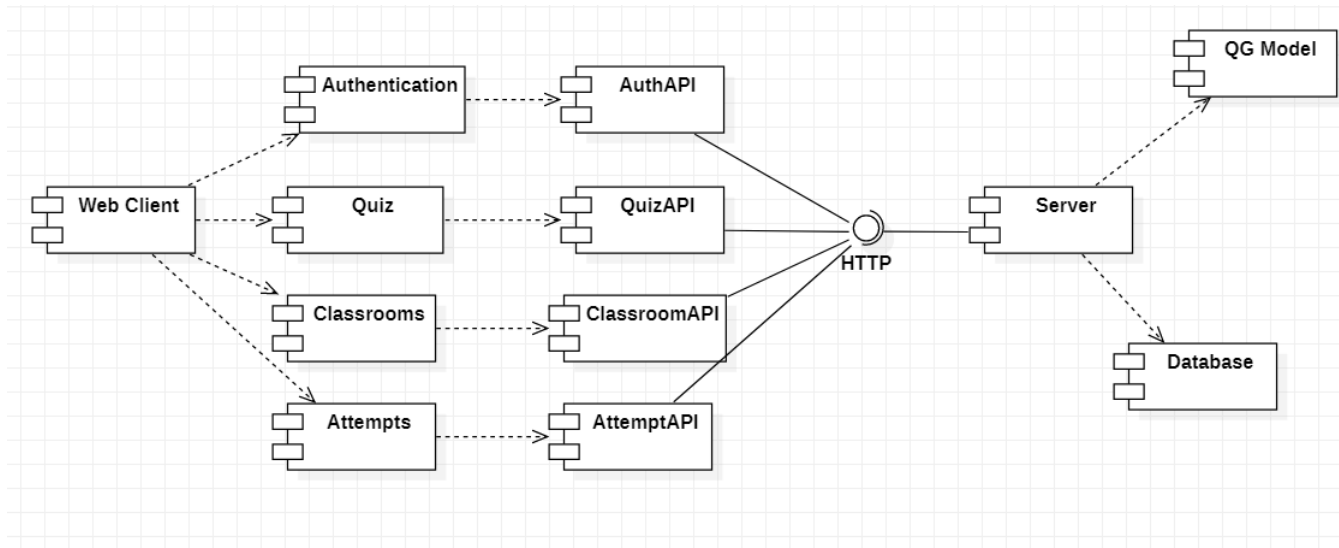


Figure 3.8: Components diagram

sist of a programming language(in this case, Javascript) and a framework(in this case, Node.js). The table below offers a description of the endpoints that have been implemented.

### 3.2.2 Node.js

Node.js is an Javascript runtime, designed to build scalable applications. It is asynchronous and event-based, and it's a powerful technology for backend development. [4] The main advantage of Node.js is that it offers an easier to use concurrency model that doesn't involve OS threads. Since the vast majority of the functions of Node.js don't perform direct I/O operations, there is no need for locks, and thus the problem of deadlocks is implicitly solved. [4]

A survey done by StackOverflow in 2020 shows that Node.js is the most popular web framework among all the types of programmers, including professionals [10], as shown in the figure 3.9.

#### Koa vs Express

Express is a flexible Node.js web framework that provides a performant set of features to create scalable web applications. It allows developers to set up their servers without any major constraints and behaves mostly like a middleware to help manage the client and database connection and routes. [15]

Koa is a Node.js web framework built by the same team that built Express, but is

Endpoint	Type of request	Description	Return type
Authentication			
/auth/login	POST	The body of the request contains the username and the encrypted password	JSON: { token:string, role:string, _id:string }
/auth/register	POST	The body of the request contains the details of the user: username, encrypted password, e-mail address and type of account.	Message. The server sends an e-mail with a link for confirmation.
/auth/confirmation	PUT	The body of the request contains the token that has been sent to the user via e-mail.	Message.
/auth/users	GET	Fetches the list of users.	List of JSONs representing the users {_id:string, username:string, role:string }.
Quizzes and attempts			
/main/quiz	POST	Fetches the quizzes of a classroom. The id of the classroom is in the body of the request.	List of quizzes.
/main/generate-quiz	POST	Sends a request to generate a quiz.	List of strings, representing the questions generated.
/main/create-quiz	POST	Sends a request to create a quiz.	The created quiz.
/main/attempt	POST	Adds a new attempt. The details of the attempt are in the body of the request.	The created attempt
/main/attempt/:id	PUT	Sends a request to update an attempt after it has been evaluated.	The updated attempt.
/main/attempts/:quizID	POST	Fetches the attempts of the quiz with the given id. The user's id is in the body of the request.	List of attempts
Classrooms			
/main/classes	POST	Creates a new classroom.	The created classroom.
/main/classes/:id	GET	Fetches the list of the classrooms where the user with the given id is enrolled.	List of classrooms.
/main/classes/:classID	PUT	Adds a user to a classroom. The user's id is in the body of the request.	The updated classroom.

different to its predecessor in the way that it is a minimalist, lightweight framework aiming to be more expressive and a more robust foundations for applications and web APIs. [15]

### 3.2.3 Persistence

Due to the experimental purpose of this application and for the ease of use, NeDB was used as database. NeDB is a lightweight database, written in Javascript, and designed to be partially compatible with MongoDB's API. [5] Considering the use of Javascript as programming language for the backend of the application, using a JSON-based database simplifies the conversion between the form the data is stored and the form that is used in the code. Also, using a NoSQL database may help prevent a series

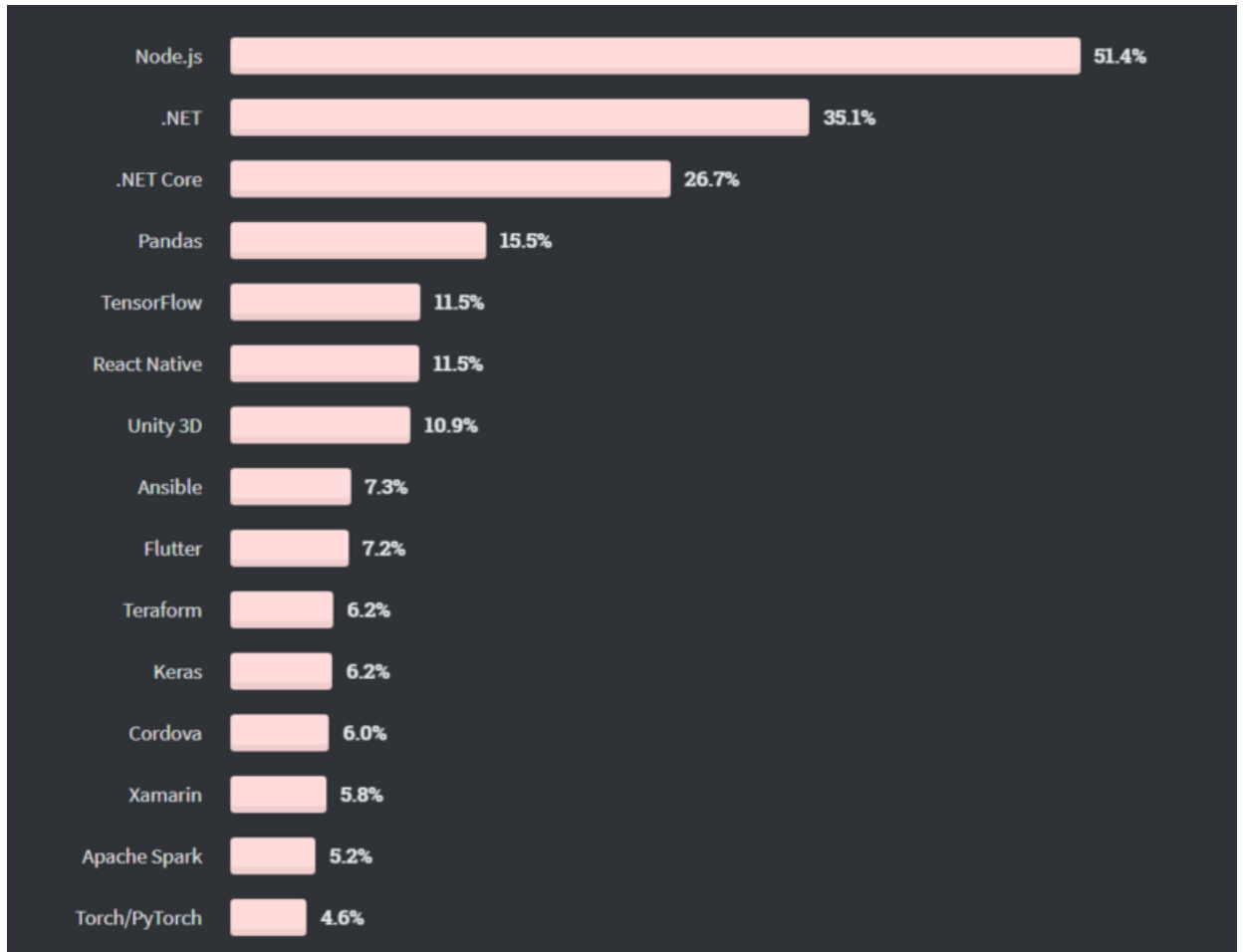


Figure 3.9: The most popular web frameworks in 2020, according to StackOverflow

of vulnerabilities like SQL injection or second-order SQL injection. Future work may include switching to a MongoDB database and using Mongoose as an ORM(Object-relation mapping).

### 3.2.4 Frontend

Complementary to the backend, the frontend side of an application represents the sum of the parts of the application which are visible to the user and with which the user can interact directly. The main tools used in the frontend development of web applications are:

- Hyper Text Markup Language (HTML) [16], which is the standard markup language for documents designed to be displayed in a web browser. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items.

- Cascading Style Sheet(CSS) [17], which is used to control the presentation of a document written in a markup language like HTML.
- Javascript, which is a high-level, event-based programming language that is widely used in order to add dynamic interactions to a website and also maintain communication with the server side.

### 3.2.5 Ionic React

Ionic is an open source framework for building mobile and web applications, with integration of popular frameworks like React, Angular and Vue. The Ionic framework was used to build the front end part of the application, as it provides a library of well designed UI components and the ease of building an application that can run both on web and on mobile from a single codebase. [3]

React is one of the Javascript component-based libraries that can be used in Ionic applications, along with Angular and Vue. React has been proven to be suitable for extensible and scalable applications such as Facebook and Dropbox.

### 3.2.6 Testing

Testing is one of the main activities of software development, as it helps the developers to evaluate the quality of the developed piece of software. Also, testing is helpful in determining whether the application satisfies the requirements that had been gathered in the early steps of the development process, with an emphasis on the security requirements, in order to prevent unauthorized access to restricted data. There is a variety of types of tests that can be carried on, each with specific objectives and strategies.

Considering the rather low complexity of the developed software, the tests that have been carried on are mainly unit tests, tests based on specification(Black Box Testing) and tests based on implementation(White Box Testing). The last two methods mentioned previously were combined with Equivalence Class Partitioning(ECP) for Black Box Testing and Cyclomatic Complexity(CC) as coverage criteria for White Box Testing. The tests were implemented in Javascript, and aimed at covering all the use cases and scenarios in which the classes on the backend side of the application can be



involved.

## 3.3 User Manual

The aim of this section is to familiarize the user with the application in order to facilitate the first interactions with the main functionalities of the application.

### 3.3.1 Log In

The application offers the basic authentication functionality. The user is required to enter the username and password in order to access his account. The application login screen is presented in figure 3.10.

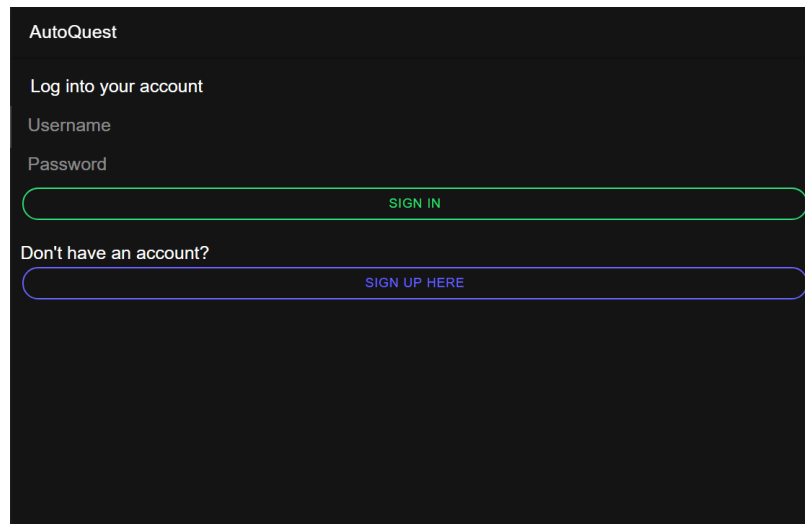
The image shows a dark-themed login interface for 'AutoQuest'. At the top, the text 'AutoQuest' is displayed. Below it is the instruction 'Log into your account'. There are two input fields: 'Username' and 'Password'. A red rounded rectangular button labeled 'SIGN IN' is positioned below the password field. Below the 'SIGN IN' button is the text 'Don't have an account?' followed by a blue rounded rectangular button labeled 'SIGN UP HERE'.

Figure 3.10: Authentication screen

### 3.3.2 Sign Up

As shown in figure 3.11, in order to create an account, the user must complete the registration form with an username, a password, a mail address and the type of account that that is wished to be created. The user can be either a student or a professor, and in order to sign up as a teacher, one must provide an official mail address(belonging to a university). If the registration succeeds, the user is notified by mail(on the e-mail address entered) to activate the account. This is done by clicking the link received in the message, which leads to the confirmation page.

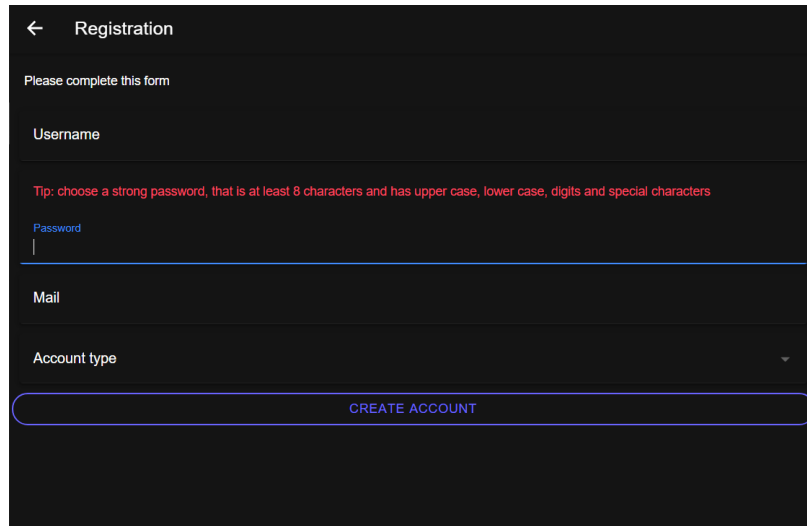
A mobile application registration screen with a dark theme. At the top, there is a back arrow and the title "Registration". Below the title, a message says "Please complete this form". The form contains four fields: "Username", "Password" (with a red tip: "Tip: choose a strong password, that is at least 8 characters and has upper case, lower case, digits and special characters"), "Mail", and "Account type" (a dropdown menu). At the bottom of the form is a large blue button labeled "CREATE ACCOUNT".

Figure 3.11: New user registration screen

### 3.3.3 Home Page

After authentication, the application shows a list of all the classrooms where the current user is enrolled. From the home page, the user can access the functionality of generating a quiz out of a given input text.

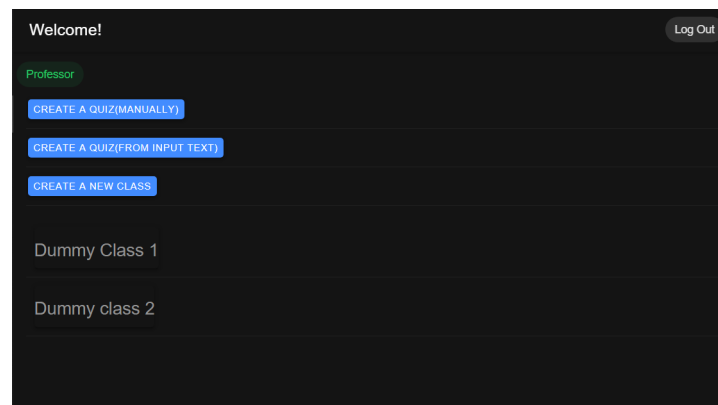
A mobile application home screen for a professor. At the top, it says "Welcome!" with a "Log Out" button on the right. Below this, the user's role "Professor" is displayed in a green box. There are three blue buttons: "CREATE A QUIZ(MANUALLY)", "CREATE A QUIZ(FROM INPUT TEXT)", and "CREATE A NEW CLASS". At the bottom, there is a list of classrooms, showing "Dummy Class 1" and "Dummy class 2".

Figure 3.12: Home screen of a professor account

The difference is that only an user with permission can upload the generated quiz to a classroom, as it can be seen in the figures 3.12 and 3.13. A professor account can also create a quiz manually, entering the questions, and create a new classroom.

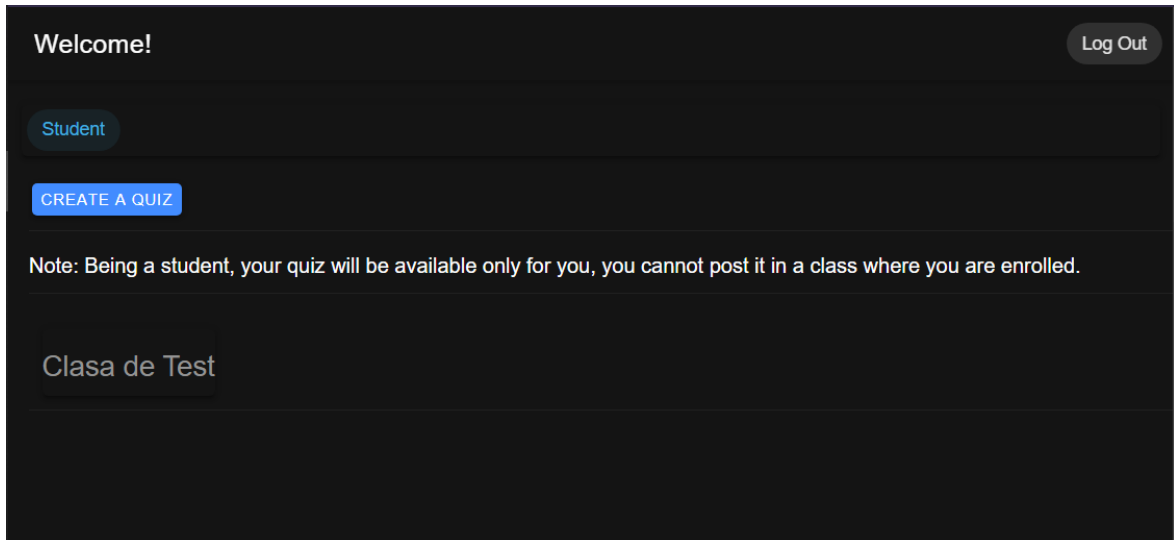


Figure 3.13: Home screen for a student account

### 3.3.4 Creating a classroom

The figure 3.14 shows how classroom can be created by a professor: by filling in the name of the class and selecting the users to be added from the list of given users.

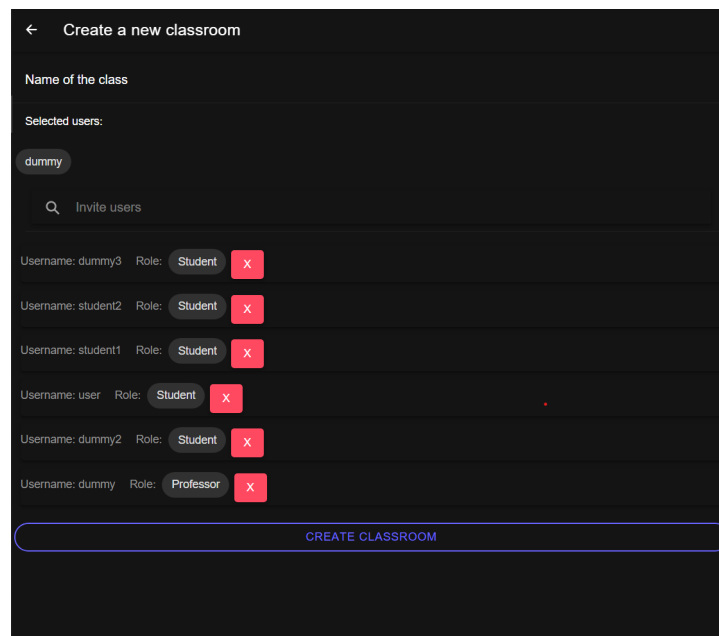


Figure 3.14: Screen for creating a new classroom

### 3.3.5 Quizzes

When clicking on a class, the user can see all the quizzes that have been uploaded there. A professor can also add a new student in the class by writing the username in a field and sending the request.

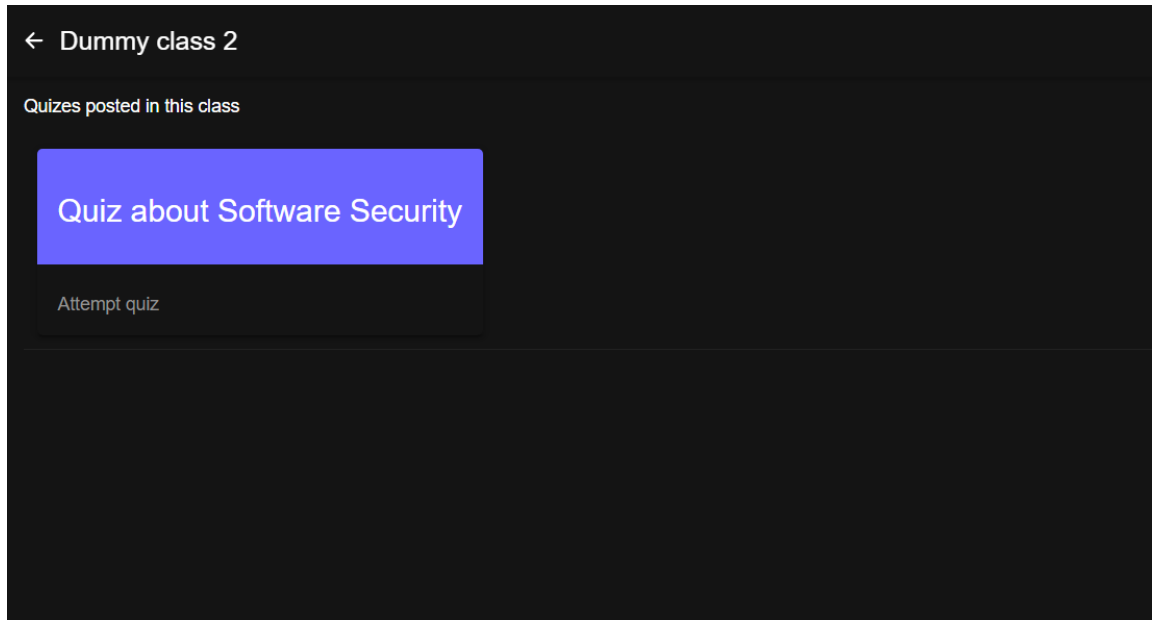


Figure 3.15: Classroom screen

A click on a quiz from the list leads to a page where the user can take the quiz. The attempts for the corresponding quiz can also be seen there. A user can see only his attempts, while the professor can see all the attempts of all the users who took the quiz and can evaluate them.

### 3.3.6 Creating a quiz

One way to create a quiz is by typing the questions manually, and the selecting a classroom in which the quiz will be added after clicking the button.

Another way is by generating a quiz from a given text. The user fills in an input and clicks the button that sends a request to the Flask API. The generated questions are returned to the client, who selects the most relevant questions and then sends the request to the server.

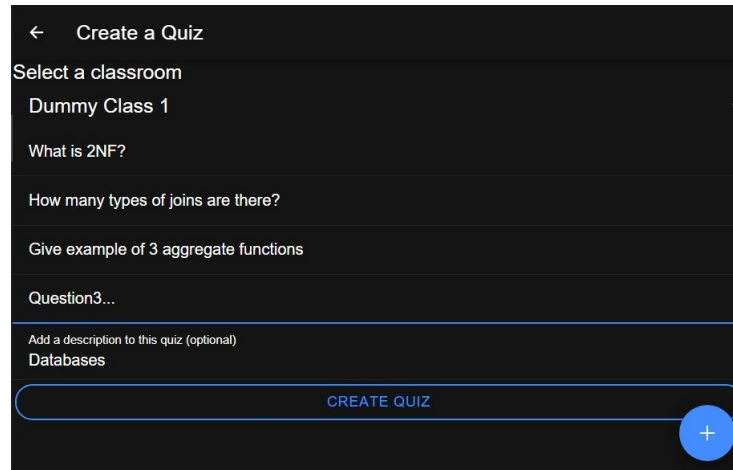


Figure 3.16: Screen for creating a quiz manually

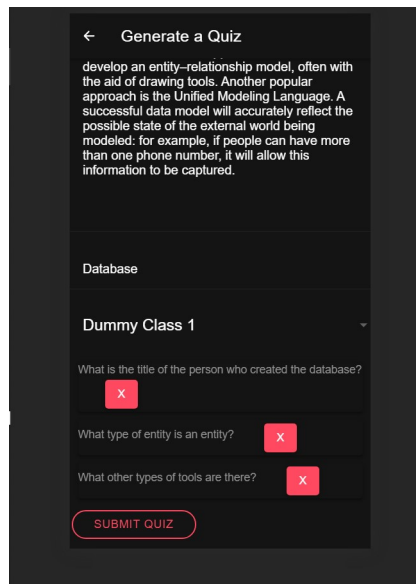


Figure 3.17: Screen for generating a quiz from text

### 3.3.7 Attempts and evaluations

A click on a quiz from the list showed in the classroom takes the user to a page where the attempts can be seen. A normal user can see only his attempts, while a professor can see all the attempts of all the users that took that quiz.

A professor can also evaluate an attempt. By a click on the button "*Evaluate this attempt*", the application changes to another page where the professor can evaluate each answer with true or false and the submit the evaluation.

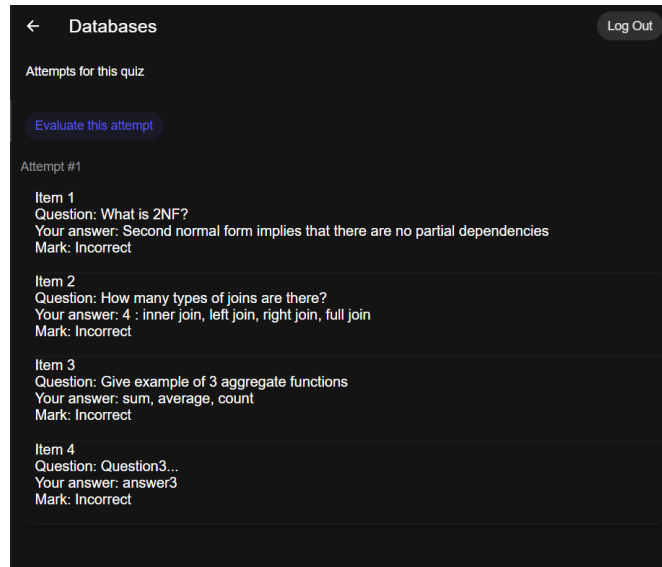


Figure 3.18: Application screen with the attempts of a quiz

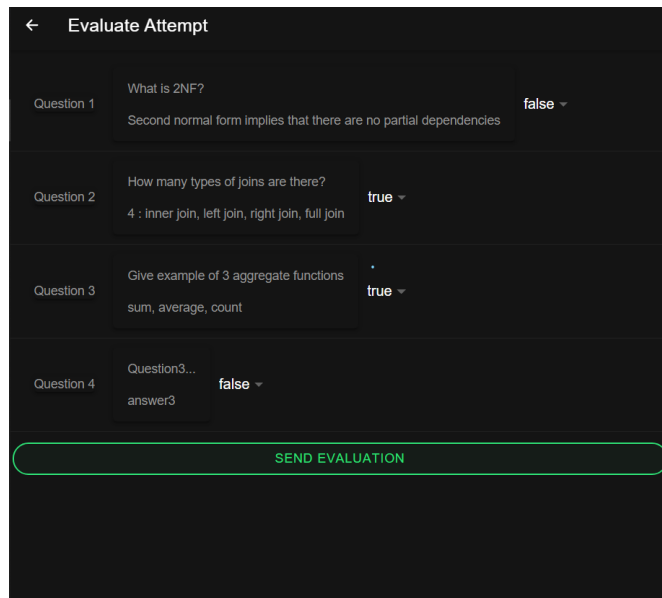


Figure 3.19: Attempt evaluation screen

### 3.3.8 Summary

To sum up, this chapter presented the web application that was designed and implemented in order to make use of the question generation model. The application is implemented using modern frameworks for asynchronous operations and responsive design.

# Conclusions and Future Work

This thesis explored the problem of question generation from natural language by using artificial intelligence algorithms, as well as designing and implementing a web application that could make use of the resulted model.

Future work may include creating a custom data set that would serve as an enhancement of SQuAD, with a wider variety of questions that can be generated. Also, adding the generation of more types of questions(true/false questions, multiple choice questions) would be another improvement idea.

As far as the web application is concerned, further work may include the migration of the database to MongoDB, since NeDB is a rather lightweight database and was used in this project solely for experimental purposes. Another functionalities to be added would be integrating Wikipedia's API in the client side in order to automatically extract the source text for the QG model and adding statistics and graphics for the user activity.

# Bibliography

- [1] \*\*\* Hugging Face - The AI community building the future. <https://huggingface.co/>. Online, accessed 29 March 2021.
- [10] Stack overflow developer survey 2020. <https://insights.stackoverflow.com/survey/2020>.
- [12] \*\*\* Neural Networks. <https://www.ibm.com/cloud/learn/neural-networks>.
- [13] \*\*\* Deep Learning. [https://www.ibm.com/cloud/learn/deep-learning#toc-deep-learn-md\\_Q\\_0f3](https://www.ibm.com/cloud/learn/deep-learning#toc-deep-learn-md_Q_0f3).
- [14] Tregex, tsurgeon and semgrep. <https://nlp.stanford.edu/software/tregex.shtml>.
- [15] Express vs koa. <https://medium.com/@theomalaper.cognez/express-vs-koa-and-hapi-a2c65f949b78>.
- [16] Html: Hypertext markup language. <https://developer.mozilla.org/en-US/docs/Web/HTML>.
- [17] Css: Cascading style sheets. <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- [3] \*\*\* Ionic React Documentation. <https://ionicframework.com/docs/react>. Online, accessed 29 March 2021.
- [4] \*\*\* Node.js. <https://nodejs.org/en/docs/>. Online, accessed 29 March 2021.



- [5] \*\*\* NeDB - The Javascript Database. <https://github.com/louischatriot/nedb/blob/master/README.md>. Online, accessed 29 March 2021.
- [6] \*\*\* Intel - Using Natural Language Processing for Smart Question Generation. <https://software.intel.com/content/www/us/en/develop/articles/using-natural-language-processing-for-smart-question-generation.html>. Online, accessed 29 March 2021.
- [7] \*\*\* SQuAD2.0 - The Stanford Question Answering Dataset. <https://rajpurkar.github.io/SQuAD-explorer/>.
- [8] \*\*\* Transfer learning for deep learning. <https://developer.ibm.com/technologies/artificial-intelligence/articles/transfer-learning-for-deep-learning#>.
- [9] \*\*\* Natural Language Processing. <https://www.ibm.com/cloud/learn/natural-language-processing>.
- [BL05] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. pages 65–72, June 2005.
- [Col20] Colin Raffel and Noam Shazeer and Adam Roberts and Katherine Lee and Sharan Narang and Michael Matena and Yanqi Zhou and Wei Li and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. 2020.
- [Eri94] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [Flo18] Florian Boudin. Unsupervised Keyphrase Extraction with Multipartite Graphs. 2018.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.

- [Hei09] Heilman Michael . Question Generation via Overgenerating Transformations and Ranking. 2009.
- [Hei11] Heilman Michael . Automatic Factual Question Generation from Text. 2011.
- [Hus10] Husam Ali, Yllias Chali, Sadid A. Hasan. Automatic Question Generation from Sentences. 2010.
- [Lin04] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. pages 74–81, July 2004.
- [Lui20] Luis Enrico Lopez and Diane Kathryn Cruz and Jan Christian Blaise Cruz and Charibeth Cheng. Transformer-based end-to-end question generation. 2020.
- [LZN<sup>+</sup>19] Bang Liu, Mingjun Zhao, Di Niu, Kunfeng Lai, Yancheng He, Haojie Wei, and Yu Xu. Learning to generate questions by learning what not to generate. *The World Wide Web Conference on - WWW '19*, 2019.
- [Por97] The Porter Stemming Algorithm. <https://tartarus.org/martin/PorterStemmer/>, 1997.
- [RZLL16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.
- [Xin17] Xinya Du, Junru Shao and Claire Cardie. Learning to ask: Neural question generation for reading comprehension, 2017.
- [XZL<sup>+</sup>20] Dongling Xiao, Han Zhang, Yukun Li, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. Ernie-gen: An enhanced multi-flow pre-training and fine-tuning framework for natural language generation. 2020.
- [Yu 20] Yu Chen and Lingfei Wu and Mohammed J. Zaki. Reinforcement learning based graph-to-sequence model for natural question generation. 2020.
- [YZL<sup>+</sup>18] Kaichun Yao, Libo Zhang, Tiejian Luo, Lili Tao, and Yanjun Wu. Teaching machines to ask questions. pages 4546–4552, 2018.