



**Ain Shams University**  
**Faculty of Computer & Information Sciences**  
**Computer Science Department**

# Exam Generator

**July 2023**



**Ain Shams University**  
**Faculty of Computer & Information Sciences**  
**Computer Science Department**

# Exam Generator

**By:**

Aisha Soliman [Computer Science]  
Mohammed Ashraf [Computer Science]  
Mostafa Ashraf [Computer Science]  
Mostafa Ayman [Computer Science]  
Mostafa Labib [Computer Science]  
Mostafa Hesham [Computer Science]

**Under Supervision of:**

Sally S. Ismail  
Dr,  
Computer Science Department,  
Faculty of Computer and Information Sciences,  
Ain Shams University.

Asmaa Bahai Eldin Kassem  
AL,  
Computer Science Department,  
Faculty of Computer and Information Sciences,  
Ain Shams University.

# Abstract

Question Generation (QG) is a branch of Natural Language Processing (NLP). It aims to generate natural language questions based on given contents. It's a combination of natural language understanding (NLU) and natural language generation (NLG) tasks. The purpose of this paper is to develop an end-to-end question generation system which can be used in real-world applications for educational purposes. Crafting exam-style questions is an essential component of education, it serves diverse objectives where both the students and educators can use it to facilitate the education process. We propose a pipelined system composed of two modules, an Answer Extraction module, and a Question Generation module. The Answer Extraction module is an encoder-decoder model that extracts question-worthy key-phrases from contexts. The Question Generation module consists of a set of specialized models finetuned on different types of questions that are generated in an end-to-end manner. The types of questions that can be generated by our system are: WH, MCQ, complete, and true/false. Evaluation of the models in our pipeline shows that our proposed system can generate educationally meaningful question-answer pairs with only context paragraphs as input, demonstrating the practicality of using automatic question generation.

# Table of Contents

Abstract	1
List of Figures	4
List of Abbreviations	5
1- Introduction	1
1.1 Motivation	1
1.2 Problem Definition	1
1.3 Objective	1
1.4 Time Plan	2
1.5 Document Organization	2
1.5.1 Chapter 2	2
1.5.2 Chapter 3	2
1.5.3 Chapter 4	2
1.5.4 Chapter 5	3
1.5.6 Chapter 6	3
2- Background	4
2.1 Project Field & Scientific Background	4
2.1.1 Artificial Intelligence	4
2.1.2 Natural Language Processing	4
2.1.3 Question Answering and Question Generation	4
2.2 Survey	4
2.2.1 Rule-based Approach	4
2.2.2 Neural Network Based Approach	4
2.3 Similar Systems	5
3- Analysis and Design	9
3.1 System Overview	9
3.1.1 System Architecture	9
3.1.2 System Users	23
3.2 System Analysis & Design	24
3.2.1 Use Case Diagram	24
3.2.2 Class Diagram	24
3.2.3 Sequence Diagram	27
3.2.4 Database Diagram	28
4- Implementation and Testing	29
4.1 Functions	29
4.1.1 Login	29

4.1.2 Register	29
4.1.3 Edit User Data	29
4.1.4 Generate Questions	29
4.2 Techniques and algorithms	29
4.2.1 User Data Storge	29
4.2.2 Generate Questions	29
4.3 New Technologies and Architecture	30
4.3.1 Project Architecture	30
4.3.2 New Technologies	31
4.4 UI Design and Wireframes	33
4.5 Testing Procedures and Levels	36
5- User Manual & Deployment Guide	37
5.1 User Manual	37
5.2 Deployment	48
5.2.1 IIS Setup	48
5.2.2 Python Environment	48
5.2.3 Frontend Angular Code Building	49
5.2.4 Backend .NET (C#) Code Building	49
6- Conclusion and Future Work	50
6.1 Conclusion	50
6.2 Future Work	50
References	51

# List of Figures

Figure 1 System Architecture Diagram. ....	9
Figure 2 A graphical representation of sense2vec. ....	10
Figure 3 T5 Pretraining Objective Example .....	13
Figure 4 System Use Case Diagram .....	24
Figure 5 System Class Diagram Part 1 .....	24
Figure 6 System Class Diagram Part 2 .....	24
Figure 7 System Class Diagram Part 3 .....	24
Figure 8 System Class Diagram Part 4 .....	25
Figure 9 System Class Diagram Part 5 .....	25
Figure 10 System Class Diagram Part 6 .....	25
Figure 11 System Class Diagram Part 7 .....	25
Figure 12 System Class Diagram Part 8 .....	26
Figure 13 System Class Diagram Part 9 .....	26
Figure 14 System Class Diagram Part 10 .....	26
Figure 15 System Class Diagram Part 11 .....	27
Figure 16 System Sequence Diagram .....	27
Figure 17 Sytem Database Diagram .....	28
Figure 18 Clean Architecture .....	30

# List of Abbreviations

Abbreviation	What the abbreviation stands for
AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
QA	Question Answering
QG	Question Generation
RNN	Recurrent Neural Network
SQuAD	Stanford Question Answering Dataset
UI	User Interface

# 1- Introduction

## 1.1 Motivation

Exercises on teaching material are not always available. This puts a lot of pressure on both educators and students. For educators, this pressure comes from coming up with quizzes from teaching material. For students, this pressure comes from summarizing studying material and finding questions on it. These tasks are time-consuming therefore automating them would save time and money.

In the age of the internet, information is widely available, but self-learning is still a hard endeavor due to the lack of exercise and practice. This can be solved by providing a way of summarizing articles into question-and-answer pairs. This helps learners to better understand the subject and gauge their understanding.

## 1.2 Problem Definition

This project aims to summarize text into question-and-answer pairs. For a chosen sentence it generates one or more questions, and each question is answered from the given text. More formally, given a set of sentences  $\mathcal{S}$  generate a set of questions  $\mathcal{Q}$  based on  $\mathcal{S}$ . For each  $q \in \mathcal{Q}$  generate an answer  $a \in \mathcal{A}$  where  $\mathcal{A}$  is the set of answers to the generated questions  $\mathcal{Q}$ .

## 1.3 Objective

Develop a web application that helps students practice and teachers write exams with the following features:

1. Summarize text into question-and-answer pairs.
2. Generate different types of questions.
3. Create a user-friendly interface in the form of a web application.



## 1.4 Time Plan

<b>Project Activities</b>	<b>Start Date</b>	<b>End Date</b>
Literature Review & Survey (Planning & Analysis)	1 October	15 November
Project Design, UI Design (Design)	15 November	31 December
Question generation model(s) (Implementation)	1 January	15 February
Front-end and back-end development (Implementation)	15 February	31 March
Integration testing models and app (Testing & Integration)	1 April	31 May

## 1.5 Document Organization

### 1.5.1 Chapter 2

In this chapter three main points were discussed: the project field and the scientific background, a survey of the field, and similar existing systems. The first point discussed the field of the project and provided important definitions related to it. The second point focused on summarizing the history of methods used in the project field. The third point aimed to summarize many recent papers that aimed to solve similar problems to the ones in our project.

### 1.5.2 Chapter 3

In this chapter the analysis and design of the system were tackled. In the first section an overview of the system including the system architecture and users were discussed. The system architecture included listing the different layers in our architecture and describing each component in said layer. The first section also included a description of the intended users of our application and their characteristics. In the second section different diagrams for the system were drawn as part of the system design to be a reference for the implementation later in the software development life cycle.

### 1.5.3 Chapter 4

In this chapter four main topics were discussed: a detailed description of the functionalities of the project, techniques and algorithms used, new technologies and web application architecture, user interface (UI) design and wireframes, and testing procedures and levels used. In the first section the functionalities of our project like editing user data and generating questions and answers were described in detail. In the second section the algorithms used in user data storage and generating questions were discussed. The third section discussed the technologies used in the project and the web application architecture. The fourth section discussed UI design and wireframes of the project. The last section discusses the testing procedures and levels used in the project.

## 1.5.4 Chapter 5

This chapter consists of two main points. The first point is a step-by-step guide of how to operate the project with screenshots illustrating each step of the guide. The second point is an installation guide that describes in detail how to install the program. Additionally, it lists all required third party tools that need to be available for the project to run.

## 1.5.6 Chapter 6

In this chapter the project is summarized in the conclusion, mentioning in brief what the aim of the project is, the system architecture, and a summary of results. Future work that can be is also discussed in this chapter.

## 2- Background

### 2.1 Project Field & Scientific Background

#### 2.1.1 Artificial Intelligence

Artificial intelligence (AI) is a wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence. Machine learning is a branch of AI and computer science that focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

#### 2.1.2 Natural Language Processing

Natural language processing (NLP) refers to the branch of computer science and more specifically, the branch of AI concerned with giving computers the ability to understand text and spoken words in much the same way human beings can. NLP has many applications, The project is concerned with two applications: question answering, and question generation.

#### 2.1.3 Question Answering and Question Generation

Question answering (QA) is a computer science discipline within the fields of information retrieval and NLP. The goal of QA is to build systems that automatically answer questions posed by humans in a natural language. While the goal of question generation (QG) is to generate a valid and fluent question according to a given passage and the target answer. Most of the older methods rely on heuristic rules for QA and QG. More recently, neural network approaches have been proposed. These approaches include the use of a kind of neural network called recurrent neural network (RNN). Most recently, sequence-to-sequence models and their variations have achieved state-of-the-art performance on many NLP tasks including both QA and QG.

### 2.2 Survey

#### 2.2.1 Rule-based Approach

Before neural network approaches were adopted for NLP tasks, especially QA and QG. The most prevalent approach was based on heuristics, as in rule-based. Researchers defined rules or templates to reach their desired goal. For example, in QG, question templates were used to generate questions. The template would be chosen depending on some rules the researchers would enforce. This led to generating unnatural questions. Additionally, questions had limited forms as they were restricted by defined templates.

#### 2.2.2 Neural Network Based Approach

Adopting neural network approaches helped remedy the problems in rule-based approaches and improve results. Neural network approaches evolved with time. For example, in QG and QA, at first, RNN was used as it can process temporal information (data that comes in sequences, such as a sentence). More recently, models that adopt the transformer architecture are being used for NLP tasks. Transformers were introduced in 2017 by a team at Google Brain. A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighing the significance of each part of the input data. The key difference from

RNNs is that transformers process the entire input all at once. This allows for more parallelization than RNNs and therefore reduces training times. The parallelization allows training on larger datasets. This led to the development of pre-trained systems such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), which were trained with large language datasets, and can be fine-tuned for specific tasks.

## 2.3 Similar Systems

The key idea of this paper ([Klein et al., 2019](#)) is that as question answering and generation are naturally related tasks, so leveraging their connection should be mutually beneficial in terms of performance as well as reducing the amount of labeled data, e.g., for training a QA system. The paper's contributions can be summarized into two main points:

- Leverage question generation by tying together GPT-2 and BERT in an end-to-end trainable fashion facilitating semi-supervised learning.
- Using QA as a surrogate measure for assessing question generation quality.

The analysis shows that the proposed generation & answering collaboration framework relatively improves both tasks and is particularly powerful in the semi-supervised setup. The results show that the model facilitates question generation in the small-data regime. Also, the generated questions are of high quality, showing high semantic similarity w.r.t. ground truth data. Additionally, the large margin between the proposed method and the initialization model clearly shows the semantic diversity of the generated question compared to the ground truth questions.

Generated sentences have high diversity and differ significantly from the ground truth. The last example shows one case of failure of the paper's method. The paper claims this is due to the very fine-grained nature of the question. But a look at the generated question shows that the question diversity leads to generating a question that is unrelated to both the answer and the context. Additionally requiring answer annotations could be considered a disadvantage.

A sequence-to-sequence model takes an answer and its context as input and generates a relevant question as output. However, the paper ([Sun et al., 2018](#)) observes two major issues with these approaches:

- (1) The generated interrogative words (or question words) do not match the answer type.
- (2) The model copies the context words that are far from and irrelevant to the answer, instead of the words that are close and relevant to the answer.

The paper proposes an answer-focused and position-aware neural question generation model. answer-focused means that question word generation is explicitly modeled by incorporating the answer embedding, which can help generate an interrogative word matching the answer type. position-aware means that the relative distance between the context words and the answer is modeled. Hence the model can be aware of the position of the context words when copying them to generate a question.

The answer-focused model correctly predicts the question word, though it copies wrong context words that can be further corrected by the hybrid model. The position-aware model can copy the correct context words after introducing the position embedding to the attention distribution. The hybrid model gets the question word right and copies the correct context more than the baseline models. On the other hand, When the question word is determined by a word (that is

far from the answer) other than the answer itself, both the answer-focused and the hybrid model generate a wrong question word. This is due to the encoding of the answer having little memory of that word.

Asking questions plays a vital role for both the growth of human beings and the improvement of artificial intelligent systems. As a dual task of question answering, question generation based on a text passage and a given answer has attracted much attention in recent years. One of the key applications of question generation is practice exercises and assessments for educational purposes.

In CGC-QG ([Liu et al., 2019](#)), a multi-task labeling strategy designed to identify whether a question word should be copied from the input passage or be generated instead, given the answer chunk “Barack Obama”, the questions “The speech in the White House is given by whom?”, “Who gives a speech on democracy?”, and “Today who gives a speech?” are all valid questions. As can be seen, although these questions share the same answer, they can be asked in different ways based on which word or phrase are chosen to be copied (“White House”, “democracy”, or “today”).

The clue word predictor predicts potential clue words that may appear a target question, based on the specific context of the input passage.

Question Generation (QG) is the task of creating questions about a text in natural language. In this work, they ([Kang et al., 2019](#)) propose Interrogative-Word-Aware Question Generation (IWAQG), a pipelined system composed of two modules: an interrogative word classifier and a QG model. The first module predicts the interrogative word that is provided to the second module to create the question. Our interrogative-word classifier is based on BERT, the authors want to build and add the special token [ANS] to let BERT knows that the answer span has a special meaning and must be used differently to the rest of the passage.

The first token of the input is the special token [CLS], to classify interrogative words, it learns how to represent the context and the answer information.

On top of BERT, a feed-forward network was built. that receives as input the [CLS] token embedding concatenated with a learnable embedding of the entity type of the answer.

Question generation (second model) is a sequence-to-sequence neural network that uses a gated self-attention in the encoder and an attention mechanism with maxout pointer in the decoder.

Automatic question generation from text is an important yet challenging problem especially when there is limited training data (i.e., pairs of sentences and corresponding questions). Standard sequence-to-sequence models for automatic question generation have been shown to perform well for languages like English, for which hundreds of thousands of training instances are available. However, training sets of this size are not available for most languages. Manually curating a dataset of comparable size for a new language will be tedious and expensive. This paper ([Kumar et al., 2019](#)) presents a cross-lingual model for leveraging a large question-answering dataset in a secondary language (such as English) to train models for QG in a primary language (such as Hindi) with a significantly smaller question-answering dataset.

This work proposes a shared encoder-decoder architecture that is trained in two phases. The first is an unsupervised pretraining phase, consisting of denoising autoencoding and back-translation. This pretraining phase only requires sentences in both primary and secondary

languages. This is followed by a supervised question generation training phase that uses sentence-question pairs in both languages to fine-tune the pre-trained weights.

To be effective, automated question generation for education must be able to ask students at various levels of development. A frequently used measure of question difficulty is Bloom's taxonomy, which defines a framework of how to assess types of questions across various levels of understanding, progressing from the simplest to the most complex. Factual questions involve recalling information and fall on the lowest level (Recall). By contrast, cause and effect questions are categorized at level 6 – Analysis – according to the original Bloom's taxonomy or level 2 Understanding – in the commonly used revised model. This work ([Stasaski et al., 2021](#)) archives this by extracting cause and effect and then generating questions from them. The pipeline proposed is simple. First it uses an improved version of Cao et al which is a set of language rules that can be used to extract cause and effect then it uses a question generation model to generate questions on cause and effect. Which is then evaluated by using a question-answering model, cause/effect presence in question and human evaluation.

This paper ([Ren et al., 2021](#)) proposes a novel configurable framework to automatically generate distractive choices for open-domain cloze-style multiple-choice questions. There are two criteria for designing distractors: plausibility and reliability. By plausibility, it means distractors should be semantically related to the key and grammatically consistent with the context given by the stem. By reliability, it means the distractor, when filled into the blank of the stem, results in a logically incorrect or inconsistent statement. The framework consists of two main components which are: Context-dependent candidate set generator: which constructs a small set of candidate distractors from a general-purpose knowledge base, based on contextual information formed by the stem and the key and Learning-to-rank model: takes both reliability checking and plausibility measures into consideration. Experimental results on a new dataset across four domains show that our framework yields distractors outperforming previous methods both by automatic and human evaluation.

Previous NQG models suffer from a problem in that a significant proportion of the generated questions include words in the question target, resulting in the generation of unintended questions. This paper ([Kim et al., 2019](#)) proposes answer-separated seq2seq, which better utilizes the information from both the passage and the target answer. By replacing the target answer in the original passage with a special token, our model learns to identify which interrogative word should be used. The authors also propose a new module termed keyword-net, which helps the model better capture the key information in the target answer and generate an appropriate question. This paper develops a novel architecture named answer-separated seq2seq which treats the passage and the target answer separately for better utilization of the information from both sides. Furthermore, the paper proposes a new module called keyword-net as a part of answer-separated seq2seq, which extracts key information from the target answer kept apart before. The keyword-net makes our NQG model consistently aware of the target answer, supplementing the information deficiency caused by answer separation. This module is inspired by how people keep the target answer in mind when they ask questions. Results show that the algorithm answer-separated seq2seq outperforms all the previous NQG models on both data splits by a great margin.

In this work ([Zhao et al., 2019](#)), the authors study the task of Question Generation (QG) from Knowledge Graph (KG) queries. This task, perse, reflects the ability of intelligent systems to translate abstract and schema-specific KG representations into universally natural language questions. KG queries are known for the expressiveness of their semantic representations

through the combination of different relations between different entities. Complex Question Generation (CQG) is more challenging than simple QG . For CQG, there is a lack of training instances to cover the exponentially growing number of different relation combinations.

The authors study complex question generation from KG, with an emphasis on how to leverage existing simple questions. A strong base neural encoder-decoder model that converts a query graph to a natural language question is first designed, and then the authors propose two extensions that explicitly consider instance-level connections between simple and complex questions, which are empirically shown to be more effective than straightforward data augmentation.

In this work ([Krishna et al., 2019](#)) it converts text generation task which converts an input document into a model generated hierarchy of question-answer (QA) pairs arranged in a top-down tree structure Questions at higher levels of the tree are broad and open-ended while questions at lower levels ask about more specific questions. The work has the following advantages:

- Generate 3 types of questions {General, specific, Yes-No} and a lot of types is exactly what is needed to make exams.
- Has a lot of filters that can be enabled to make sure of generating a good question to improve Top-k sampling.
- Can generate a lot of questions by generating specific questions from generated questions.

The work has the following disadvantages:

- Reliance on a flawed answering system.
- Lack of world knowledge.
- Multiple GENERAL QA per paragraph.

This work consists of five steps:

- Answer span selection.
- Question generation is conditioned on answer spans and specificity labels.
- extractive answering generated questions.
- filtering out bad QA pairs.
- Structuring the remaining pairs into a GENERAL-to-SPECIFIC hierarchy.

## 3- Analysis and Design

### 3.1 System Overview

#### 3.1.1 System Architecture

As shown in Fig. 1, the architecture is divided into 3 layers: User Interface, Application Layer, and Data Access Layer. In the Application Layer, we have 5 modules Answer Extraction, Interrogative Word Classifier, WH QG model, Boolean QG model, and MCQ & Complete model.

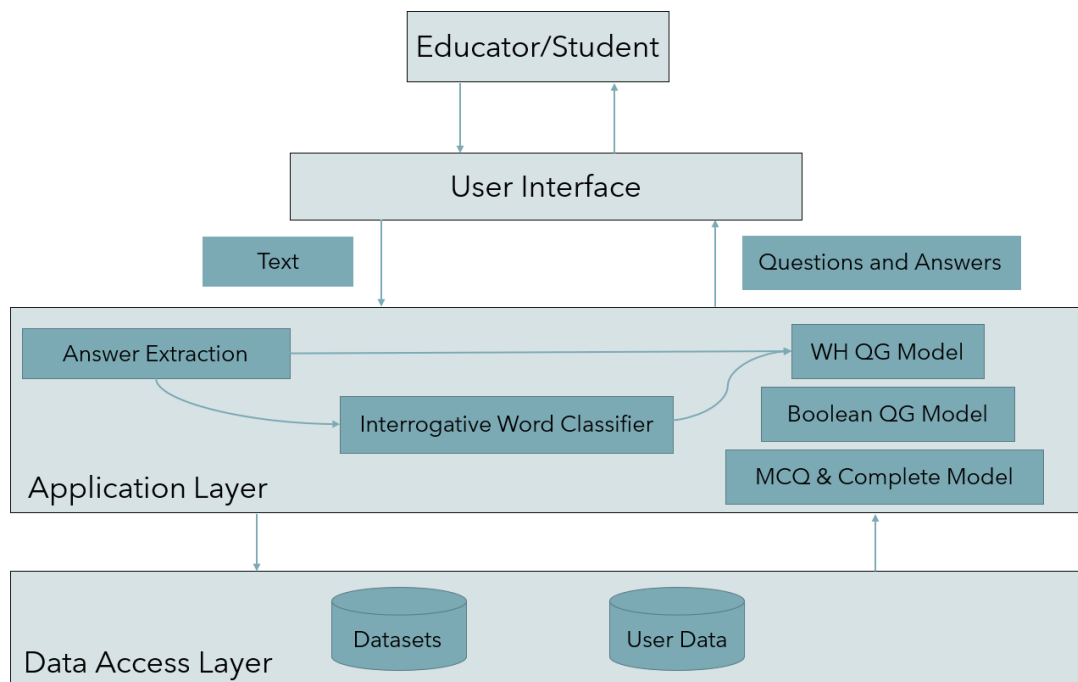


Figure 1 System Architecture Diagram.



# Layers

## A. User Interface Layer

A web application that has the following characteristics:

- Easy to use.
- Lightweight.

## B. Application Layer

The application layer is divided into multiple modules, each of which has a specific objective. The modules are as follows:

### 1. MCQ & Complete (Sense2Vec)

Sense2vec ([Trask et al.](#)) is an advanced neural network model that produces vector representations of words based on large collections of texts. It is an extension of the popular word2vec algorithm. Unlike word2vec, which generates embeddings for individual word tokens, sense2vec creates embeddings for "senses" of words. A sense is a combination of a word and a label that represents the specific context in which the word is used. This label can indicate various aspects such as part-of-speech (POS) tags, polarity, entity names, or dependency tags. This approach eliminates the requirement of training embeddings multiple times and removes the need for a clustering process. As a result, we develop an efficient method where a supervised classifier can utilize the relevant word-sense embedding.

### Architecture

It's built upon the research conducted by Huang et al. (2012) by utilizing supervised natural language processing (NLP) labels instead of unsupervised clusters to identify the specific sense of a word. This approach eliminates the requirement of training embeddings multiple times and removes the need for a clustering process. As a result, we develop an efficient method where a supervised classifier can utilize the relevant word-sense embedding.

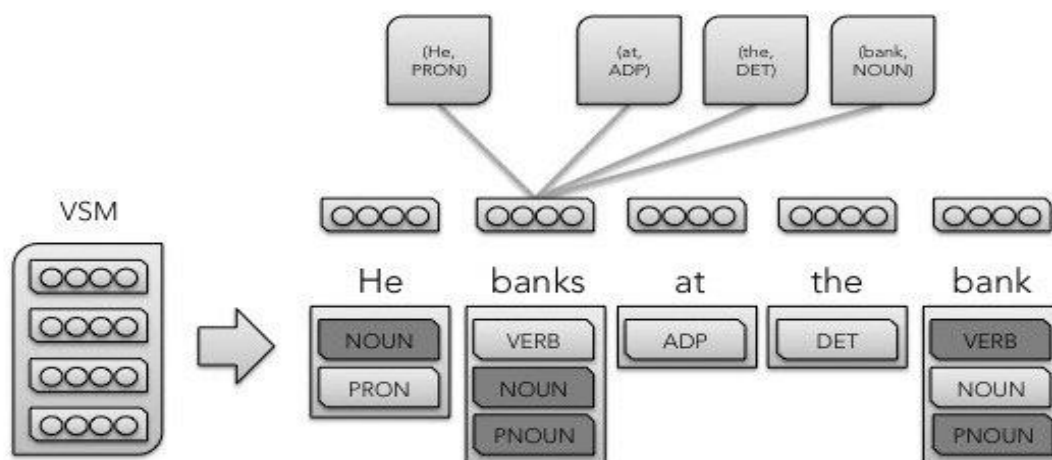


Figure 2 A graphical representation of sense2vec.

Given a labeled corpus (either by hand or by a model) with one or more labels per word, the sense2vec model first counts the number of uses (where a unique word maps set of one or more labels/uses) of each word and generates a random "sense embedding" for each use. A model is then trained using either the CBOW, Skip-gram, or Structured Skip-gram model configurations. Instead of predicting a token given surrounding tokens, this model predicts a word sense given surrounding senses.

## Implementation

Sense2Vec is used as part of a spaCy pipeline to generate questions. Sense2Vec can extract the sense of the word on its own, but its accuracy wasn't good enough for us. The "en\_core\_web\_lg" spaCy model is used for our pipeline. Sense2Vec is pretrained on two datasets of Reddit comments, the first being in 2015, and the second being in 2019. The 2019 dataset is used as it can capture the most recent meanings of words for example the word ghost can mean a supernatural spirit (2015 dataset) or the action ignoring your spouse (2019 dataset). We use spaCy to extract all named entities in the input text. Several filtering techniques are applied to enhance the quality of the distractors. The first technique is making sure that the NER tag of the distractor matches that of the correct answer. The second technique is making sure there are no common words between the distractor and the correct answer. This is done as the dataset is collected from comments on the Reddit website which are prone to spelling mistakes. As such the same word with a different spelling can have two different embeddings where similarity between them is high leading to multiple correct answers or duplicate distractors. The third technique is making sure that cardinals in different formats are not chosen as distractors e.g., if "1" is the answer then "one" is filtered from being chosen as a distractor. The complete questions are the same as MCQ questions but without any distractors and the correct answer is omitted from the text.

# T5 Model

## Architecture

T5 ([Raffel et al.](#)) is a text-to-text transfer transformer model, which means that it is a transformer model that is pre-trained on a massive dataset of text. Additionally, the model is fine-tuned for a wide range of NLP tasks, such as machine translation, question answering, summarization, and text generation. The T5 model architecture is based on the Transformer architecture, which was first introduced in the paper "Attention Is All You Need" ([Vaswani et al.](#)). The Transformer architecture is a neural network architecture that uses self-attention to learn long-range dependencies between words in a sequence.

The T5 model architecture consists of two main components: an encoder and a decoder. The encoder takes the input text and the decoder generates the output text. The encoder and decoder are both made up of a stack of self-attention layers. The encoder starts by embedding the input text into a sequence of vectors. The vectors are then passed through a stack of self-attention layers, which learn the long-range dependencies between the words in the input text. The output of the encoder is a representation of the input text that is stored in a memory. The decoder starts by taking the representation of the input text from the encoder and generating a sequence of tokens. The tokens are then passed through a stack of self-attention layers, which learn the long-range dependencies between the tokens in the output text. The output of the decoder is the generated output text.

## I/O Format

With T5, the authors propose reframing all NLP tasks into a unified text-to-text-format where the input and output are always text strings, in contrast to BERT-style models that can only output either a class label or a span of the input. The text-to-text framework allows the use of the same model, loss function, and hyperparameters on any NLP task, including machine translation, document summarization, question answering, and classification tasks (e.g., sentiment analysis). We can even apply T5 to regression tasks by training it to predict the string representation of a number instead of the number itself.

## Pretraining Dataset

The C4 dataset was used in the unsupervised pretraining objective. C4 dataset is a collection of about 750GB of English-language text sourced from the public Common Crawl web scrape. It includes heuristics to extract only natural language (as opposed to boilerplate and other gibberish) in addition to extensive deduplication. The dataset was explicitly designed to be English only. The authors apply simple heuristic filtering. Any lines that didn't end in a terminal punctuation mark are removed. Additionally, any line with the word JavaScript and any pages that had a curly bracket (since it often appears in code) were removed. The dataset is deduplicated by taking a sliding window of 3 sentence chunks and is deduplicated so that only one of them appears the dataset.

## Pretraining

T5 pretraining was divided into two steps:

### 1. Unsupervised Pretraining

T5 trains with the same objective as that of BERT's which is the Masked Language Model with a little modification to it. Masked Language Models are Bidirectional models, at any time the representation of the word is derived from both left and the right context of it. The subtle difference that T5 employs is to replace multiple consecutive tokens with a single Mask

keyword, unlike BERT that uses Mask token for each word. The model was trained on the C4 corpus with the same objective as a part of the pre-training.



T5's mask language modeling (Raffel et al., 2019)

*Figure 3 T5 Pretraining Objective Example*

As you can see from the above figure, the original text is transformed into input and output pairs by adding perturbations to it. Since the final objective is to train a model that inputs text and outputs text, the targets were designed to produce a sequence. The corruption scheme used was masking a span of text (more than 1 consecutive words). Additionally, the corruption rate used was 15% as it was found to be the best according to the authors' experiments.

## 2. Supervised Pretraining

After unsupervised pretraining, T5 was then finetuned on various tasks such as Language Translation, Summarization, Sentence Similarity, etc. Fine-tuning was done by showing model the I/O text pairs with task-specific prefix-text added to each input. For example — translate English to German: <text>, adding such a prefix enabled the model to tune its weight for a particular task in-hand and would only produce the expected output for that task alone by narrowing its scope of generation. All the tasks essentially share the same objective, training procedure, and decoding process, but differ in the task-specific prefix. The authors also claim that they did not find any single case where the model got confused and output something totally random or expected output of another task.

## 2. Answer Extraction

### Task Description

The task is to train a neural model that when given a paragraph as input, outputs phrases in that paragraph that can be answers to questions. The goal of the model is to extract question-worthy key phrases that when given to a question generation model will result in generating a question that is not only correct syntactically but also semantically.

### Architecture

The answer extraction model utilizes the powerful T5 model to extract key phrases. The T5 model architecture is based on the Transformer architecture, which is a neural network architecture that is well-suited for natural language processing tasks. The T5 model consists of an encoder and a decoder, which are connected by a series of self-attention layers. The encoder takes the input text as input and produces a sequence of hidden states. The decoder then takes these hidden states as input and produces the output text. The self-attention layers in the encoder allow the model to learn the relationship between input words. On the other hand, in the decoder the self-attention layers allow the model to capture the relationship between output text.

The T5 model has several unique features that make it different from other Transformer-based models. First, the T5 model is trained on a massive dataset of text (C4 dataset), which allows it to learn a wider variety of relationships between words. Second, the T5 model can learn the context of a phrase, which helps it to identify phrases that are important even if they do not appear frequently. These features enable it to achieve good performance when it comes to the task of extracting question-worthy key phrases.

### I/O

Since the answer extraction module is a T5 model, both the input and the output are text. The following is an example input given to the model:

extract answers: context: <context>

The input follows the T5 paper format for input which instructs to append a task-specific prefix to the input as that can help the model achieve better results if the task is similar to one of the downstream tasks the T5 model was pretrained on. That prefix being “Extract answers: “.

### Finetuning Datasets

#### Description

We used two reading comprehension datasets in the training of the answer extraction model. The datasets used are as follows:

1. Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.
2. NewsQA is a challenging machine comprehension dataset of over 100,000 human-generated question-answer pairs. Crowdworkers supply questions and answers based on a set of over 10,000 news articles from CNN, with answers consisting of spans of text from the corresponding articles.

As can be seen the datasets are from different sources which introduces variety in the data that should help the model learn and generalize on unseen data examples.

## Preprocessing

Both datasets had the following preprocessing applied:

- Remove duplicate answers from each row.
- Duplicate contexts were combined into a single row and any duplicates were removed.

## Implementation

The Answer Extraction model is a T5 model, specifically a t5-small model. The t5-small model is provided by HuggingFace ([Wolf et al.](#)). The model is trained for 8 epochs with batch size 4. The first epoch being a warmup epoch where the learning rate starts increasing gradually until reaching  $3e-4$ . For the remaining 7 epochs a polynomial decay function of power 1 is applied to decrease the learning rate gradually until it reaches  $3e-5$  at the end of the last epoch. The optimizer used for training is the Adam optimizer and the loss function is the cross-entropy loss function. The size of the encoder was set to 250, and the size of the decoder was set to 70. The task prefix used was “extract answers”.

## Evaluation Metrics

We used the evaluation in ([Subramanian et al.](#)) as their task matched ours. The authors proposed an extension of the SQuAD F1 metric (for a single answer span) to multiple spans within a document, which is called the multi-span F1 score. This metric is calculated as follows. Given the predicted phrase  $\hat{e}_i$  and a gold phrase  $e_j$ , we first construct a pairwise, token-level F1 score matrix of elements  $f_{i,j}$  between the two phrases  $\hat{e}_i$  and  $e_j$ . Max-pooling along the gold-label axis essentially assesses the precision of each prediction, with partial matches accounted for by the pairwise F1 (identical to evaluation of a single answer in SQuAD) in the cells:  $p_i = \max_j (f_{i,j})$ . Analogously, the recall for label  $e_j$  can be computed by max-pooling along the prediction axis:  $r_j = \max_i (f_{i,j})$ . We define the multi-span F1 score using the mean precision  $\bar{p} = \text{avg}(p_i)$  and recall  $\bar{r} = \text{avg}(r_j)$ :  $F1_{MS} = \frac{2\bar{p} \cdot \bar{r}}{\bar{p} + \bar{r}}$ .

### 3. Interrogative Word Classifier

#### Task Description

The task is to predict the interrogative word of the question that is provided to the Question Generation module to create the question. Owing to an increased recall of deciding the interrogative words to be used for the generated questions. Our interrogative-word classifier is based on BERT, a state-of-the-art model in many NLP tasks that can successfully utilize the context to grasp the semantics of the words inside a sentence.

#### Architecture

Our proposed classifier is based on BERT (Bidirectional Encoder Representations from Transformers), which has significantly advanced the state-of-the-art in various NLP tasks including sentence classification, inter-sentence classification, information extraction, and question answering. BERT is a pre-trained deep language model that utilizes a bidirectional transformer architecture. It surpasses human performance in machine comprehension tasks and demonstrates exceptional performance in text classification.

BERT's effectiveness stems from its ability to capture contextual semantic information within vector representations. When trained on large unlabeled texts, BERT can generate representations that encapsulate the surrounding context of each word. These representations can then be fine-tuned

The architecture of BERT consists of two main components: the encoder and the pre-training objectives. Encoder: The encoder is a stack of transformer layers. A transformer layer consists of multiple self-attention mechanisms and feed-forward neural networks. Pre-training Objectives BERT is pre-trained using two unsupervised tasks to learn contextualized word representations.

#### I/O

Given a (passage – answer) of text as input, the BERT-based classifier model aims to predict the appropriate interrogative word among the following seven classes: who, what, when, where, why, how, and which. The model utilizes its understanding of the passage's context to generate a grammatically correct interrogative word as the output.

For example:

Passage:

"The Mona Lisa is a famous portrait painting created by the Italian artist Leonardo da Vinci. It is believed to have been painted between 1503 and 1506. The painting is currently housed in the Louvre Museum in Paris, France. The Mona Lisa is known for her enigmatic smile, which has intrigued art enthusiasts for centuries."

Answer:

"The Mona Lisa was painted between 1503 and 1506."

Output (Interrogative Word): "When"

## **Finetuning Datasets**

### **Description**

We used three datasets, two reading comprehension datasets, and Comprehension dataset constructed using an adversarial model-in-the-loop in the training of the Interrogative Classifier. The datasets used are as follows:

Stanford Question Answering Dataset (SQuAD). SQuAD is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.

NewsQA is a challenging machine comprehension dataset of over 100,000 human-generated question-answer pairs. Crowdworkers supply questions and answers based on a set of over 10,000 news articles from CNN, with answers consisting of spans of text from the corresponding articles.

Adversarial reading comprehension tasks are specifically designed to evaluate the robustness and limitations of machine learning models in understanding. This dataset is crafted to include challenging examples that go beyond the capabilities of standard models. Similar to SQuAD, an adversarial dataset typically consists of passages, questions, and corresponding answer spans or unanswerable questions.

As can be seen the datasets are from different sources which introduces variety in the data that should help the model learn and generalize on unseen data examples.

### **Preprocessing**

Over the three datasets the following preprocessing applied:

Downsampling the datasets (the number of interrogative words is not balanced)

Filtering Passages that no fit in BERT Max-length (512 token)

## **Implementation**

Bert adds a special token [CLS] to classify interrogative words, on top of BERT, a feed-forward network is built that receives as input the [CLS] token embedding concatenated with a learnable embedding of the entity type of the answer.

The interrogative-word classifier is made using the PyTorch implementation of BERT. It was trained for three epochs using cross-entropy loss. The input dimension of the classifier is 773 (768 from BERT base hidden size, output dimension is 8 (SoftMax layer) as the predicted interrogative words are: what, which, where, when, who, why, how, and others. For optimization, Adam optimizer with weight decay and a learning rate of  $5e-5$  was used.

## **Evaluation Metrics**

Through research and evaluation, it was found that the model achieved significantly higher accuracy compared to the base-model (71.7). Furthermore, the accuracy of our model was found to be comparable to that of the IWAQG model (73.8).



## 4. WHQG Model

### Task Description

The goal of this task is to develop a powerful Question Generation (QG) model based on the T5 architecture. This model will take a passage, an answer, and an interrogative word as input and generate high-quality, contextually relevant questions. By incorporating the T5 model, which has shown remarkable performance in natural language processing tasks, we aim to enhance the question generation process and provide accurate, informative, and diverse questions.

### Architecture

The Question Generation model utilizes the powerful T5 model to generate high-quality, relevant questions. The T5 model, built upon the Transformer architecture, consists of an encoder and a decoder interconnected by self-attention layers. These layers enable the model to capture word relationships and generate coherent output. The encoder encodes the input text by learning hidden states and attending to important contextual information. The decoder uses self-attention to generate output text, attending to previously generated words. Leveraging the Transformer's capabilities, T5 excels in capturing context, modeling dependencies, and generating high-quality output, making it popular in natural language processing tasks.

The T5 model distinguishes itself from other Transformer-based models through several distinctive attributes. Firstly, it undergoes training on an extensive text dataset (C4 dataset), facilitating its comprehension of diverse word relationships. This broad training scope enables the model to recognize and assign significance to phrases that may not occur frequently but hold contextual importance. Consequently, the T5 model demonstrates commendable proficiency in extracting key phrases worthy of generating questions.

### I/O

Since the Question Generation module is a T5 model, both the input and the output are text. The following is an example input given to the model:

generate question using question word:

<interrogative word > answer: <answer > context: <context>

According to the T5 paper, it is recommended to include a task-specific prefix in the input to improve the model's performance, especially when the task aligns with the downstream tasks the T5 model was pretrained on. This prefix, "generate question using question word:", is appended to the input and aids in obtaining enhanced outcomes.

### Finetuning Datasets

#### Description

We used one reading comprehension datasets in the training of the question generation model. The dataset:

- Stanford Question Answering Dataset (SQuAD). SQuAD is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding.

## Implementation

The t5-base model was utilized for the QG model. The task prefix used was "generate question using question word". During training, the learning rate was set to  $5e-5$  and the encoder size to 250 tokens. For decoding, a size of 70 tokens was used. The QG model was trained for three epochs, allowing it to learn and improve over multiple iterations.

## Evaluation Metrics

In our qualitative analysis, we conducted a comparison of the generated questions from the baseline model ([Zhao et al., 2018](#)), the IWAQG model, and our model. Across all metrics, our model consistently outperformed the other models. Our model demonstrated higher scores in terms of relevance, information coverage, grammatical correctness, and question diversity. This indicates that our model excelled in generating questions that were more contextually relevant, comprehensive in coverage, syntactically accurate, and exhibited a wider range of question types.

Using the following metrics:

BLUE-1, BLUE-2, BLUE-3, BLUE-4 ([Papineni et al.](#)), METEOR ([Denkowski et al.](#)) and ROUGE-L ([Lin et al.](#)).

## 5. Boolean QG Model

### Task Description

The task is to train a neural model that is capable of generating true/false (boolean) questions that are relevant and be answered by information from the given passage. More formally, given a context and an answer (true/false) the model generates a question that has an answer as either true or false depending on the answer given to it as input.

### Architecture

The boolean question generation model is a T5 model. The T5 model was chosen as it has demonstrated state of the art performance on a variety of natural language processing tasks including but not limited to text summarization, text generation, and question answering. Additionally, the model was pretrained on many downstream tasks such as text translation, and question answering.

The unsupervised pre-training objective used by T5 aligns more closely with a fill-in-the-blank task where the model predicts missing words within a corrupted piece of text. This objective is a generalization of the continuation task, since the “blanks” can appear at the end of the text as well. This helps T5 capture word relations better and generate grammatically and semantically correct outputs. The pretraining of T5 makes it a perfect candidate for our task as it can correctly identify relevant words and phrases when generating a question using the self-attention mechanism of the Transformer architecture.

### I/O

Since the answer extraction module is a T5 model, both the input and the output are text. The following is an example input given to the model:

Generate boolean question: <answer > context: <context>

The following is a description of each part of the input:

- “Generate boolean question:” is the task-specific prefix. The input follows the T5 paper format for input which instructs to append a task-specific prefix to the input as that can help the model achieve better results if the task is similar to one of the downstream tasks the T5 model was pretrained on.
- “<answer>” is a placeholder for either true or false depending on the desired answer of the question to be generated.
- “context:” is a prefix to mark the start of the context and the end of the answer.
- <context> is a placeholder for the context of the question.

## Finetuning Datasets

### Description

Due to the scarcity of research done on this task only one dataset was deemed of high quality to be for training our model. The following is a description of the dataset:

- BoolQ: BoolQ is a question answering dataset for yes/no questions containing 15942 examples. These questions are naturally occurring ---they are generated in unprompted and unconstrained settings. Each example is a triplet of (question, passage, answer), with the title of the page as optional additional context. The text-pair classification setup is similar to existing natural language inference tasks.

### Implementation

The Boolean question generation model uses the HuggingFace TensorFlow implementation of the t5-base model. The model was trained with Adam optimizer with weight decay using the cross-entropy loss function. The model was trained for 15 epochs with a constant learning rate of  $3e-4$  and a batch size of 8. The size of the encoder was set to 500, while for the decoder the size was set to 70. The task prefix used was “generate Boolean question”.

### Evaluation Metrics

As our task is a question generation task albeit more niche, we use BLEU-1 through BLEU-4, METEOR and Rogue-L metrics. Although all the previous metrics were originally developed for machine translation, they were adopted for text summarization and question generation as they still provide good insight into the performance of the models evaluated. Our models outperforms the best model of ([Yuan et al.](#)) in all metrics except for Rogue-L with a 0.4 difference in their favor.

## 6. Sense2Vec

MCQ and complete are related tasks, in complete the task is to extract a key phrase and its sentence omitting the key phrase (answer), for MCQ we add the task of generating distractors related to the answer. MCQ and complete modules share the same steps except that MCQ gives you multiple possible answers in the form of distractors. The distractors in MCQ are generated by the Sense2Vec [10] model. Sense2Vec is an extension of Word2Vec. Unlike Word2Vec, which generates embeddings for individual word tokens, Sense2Vec creates embeddings for "senses" of words. A sense is a combination of a word and a label that represents the specific context in which the word is used. This label can indicate various aspects such as part-of-speech (POS) tags, polarity, entity names, or dependency tags. For example, in Word2Vec you can have the following:

$$v[\text{king}] - v[\text{man}] + v[\text{woman}] \approx v[\text{queen}]$$

where  $v$  denotes vector embedding of word. In Sense2Vec POS tags as well as entity names are used to generate word embeddings. We take advantage of spaCy being able to support adding Sense2Vec as part of a pipeline and use this pipeline to generate MCQ and complete questions.

We use Sense2Vec as part of a spaCy pipeline to generate questions. Sense2Vec can extract the sense of the word on its own, but its accuracy wasn't good enough for us. We use the "en\_core\_web\_lg" spaCy model for our pipeline. Sense2Vec is pretrained on two datasets of Reddit comments, the first being in 2015, and the second being in 2019. We use the 2019 dataset as it can capture the most recent meanings of words for example the word ghost can mean a supernatural spirit (2015 dataset) or the action ignoring your spouse (2019 dataset).

We use spaCy to extract all named entities in the input text. We apply several filtering techniques to enhance the quality of the distractors. The first technique is that we make sure that the NER tag of the distractor matches that of the correct answer. The second technique is making sure there are no common words between the distractor and the correct answer. This is done as the dataset is collected from comments on the Reddit website which are prone to spelling mistakes. As such the same word with a different spelling can have two different embeddings where similarity between them is high leading to multiple correct answers or duplicate distractors. The third technique is making sure that cardinals in different formats are not chosen as distractors e.g., if "1" is the answer then "one" is filtered from being chosen as a distractor. The complete questions are the same as MCQ questions but without any distractors and the correct answer is omitted from the text.

## C. Data Access Layer

This layer is used for the training and testing of the QG models. It is not used in production. It also contains the database which contains the user data. Since the user data is the only stored data, the database consists of only one table to store it, there is no need to divide it and automatically generated tables to manage Hangfire (recurring jobs) and Entity framework migrations. They automatically should not be changed as they are not related to system data.

### 3.1.2 System Users

#### **A. Intended Users:**

The intended users of the system can be divided into two categories as follows:

##### **Learners:**

They will use the application to generate questions to help them study and prepare for exams.

##### **Educators:**

They will use the application to help them make quizzes and exams faster and easier.

#### **B. User Characteristics**

- Basic computer skills.
- Access to the internet.

## 3.2 System Analysis & Design

### 3.2.1 Use Case Diagram

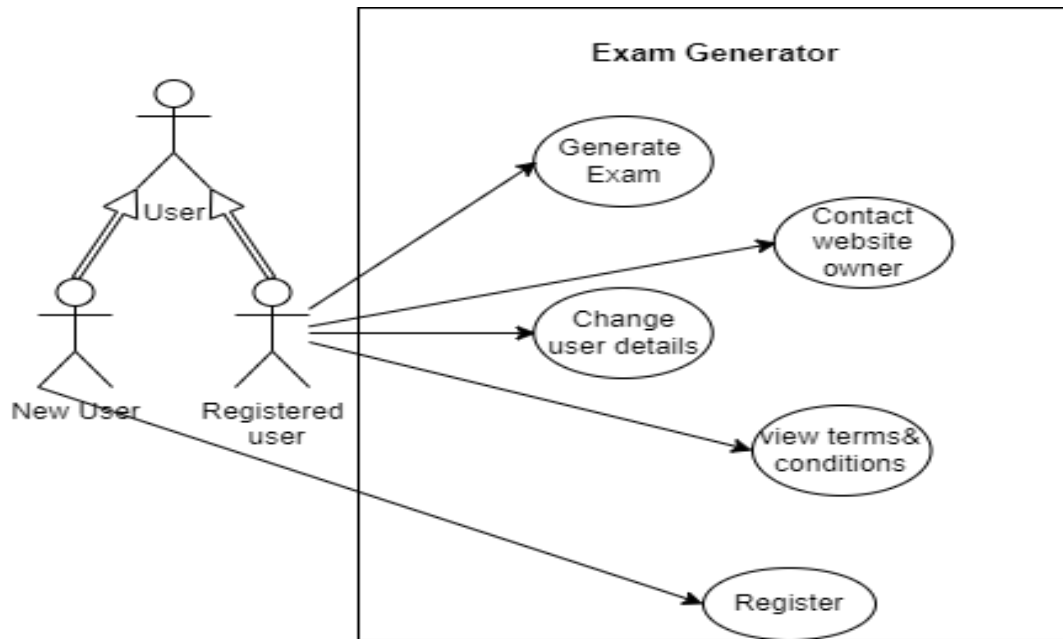


Figure 4 System Use Case Diagram

### 3.2.2 Class Diagram

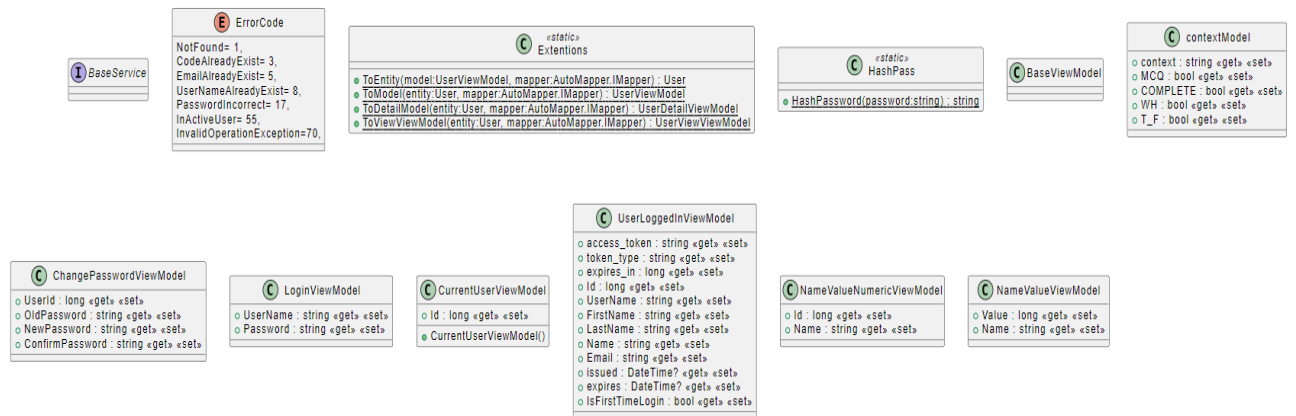


Figure 5 System Class Diagram Part 1

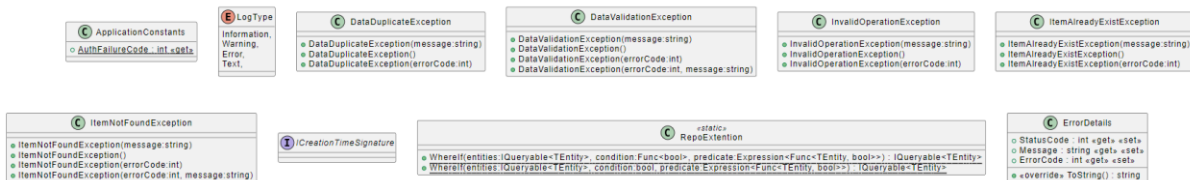


Figure 6 System Class Diagram Part 2



Figure 7 System Class Diagram Part 3





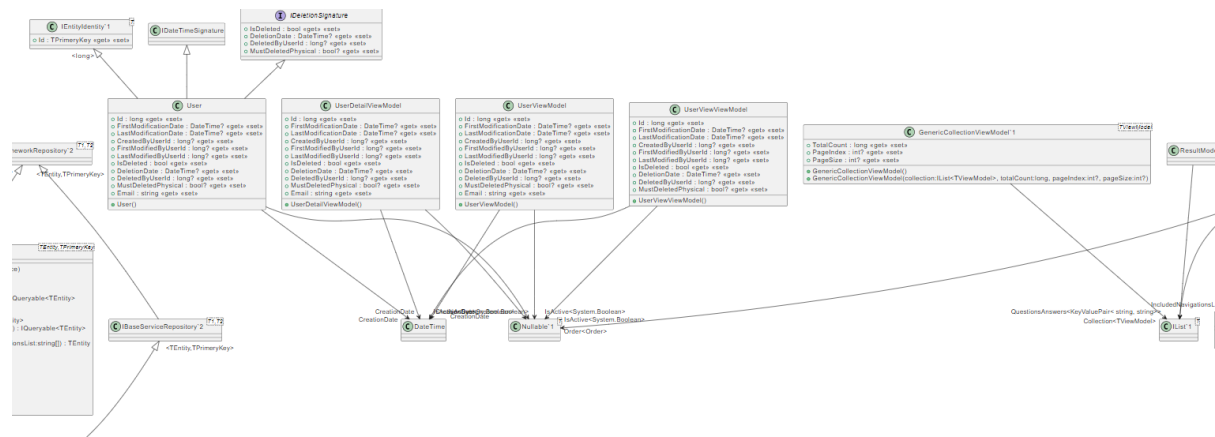


Figure 12 System Class Diagram Part 8

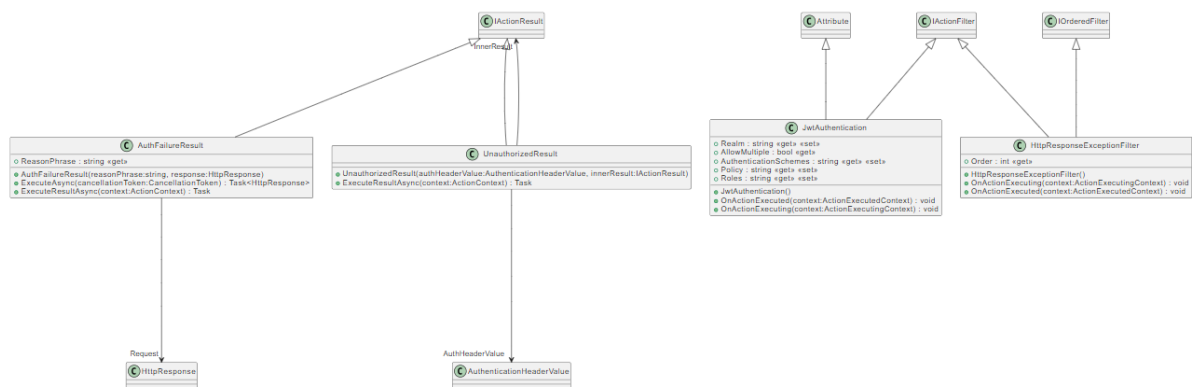


Figure 13 System Class Diagram Part 9

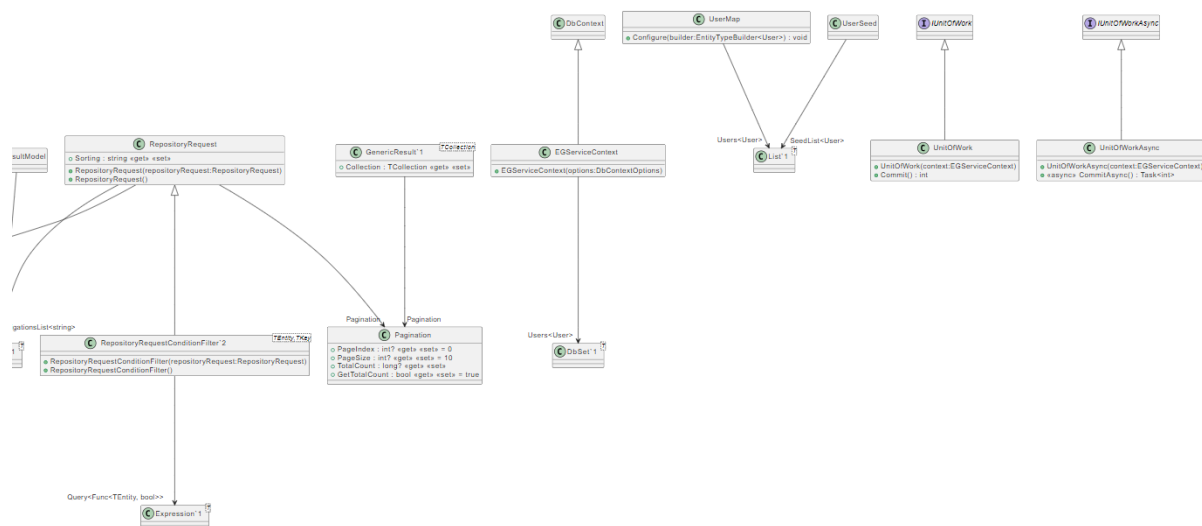


Figure 14 System Class Diagram Part 10



### 3.2.4 Database Diagram

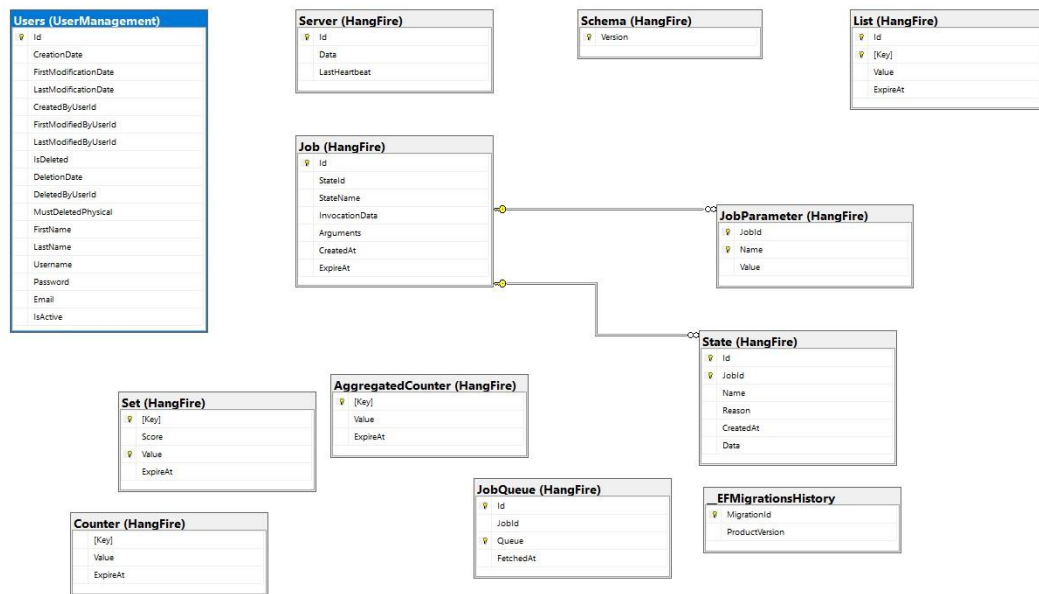


Figure 17 Sytem Database Diagram

## **4- Implementation and Testing**

### **4.1 Functions**

#### **4.1.1 Login**

Users can only interact with the system after they login using an E-mail and password that are already registered on the system.

#### **4.1.2 Register**

Users register on the system by entering the following data: First Name, Last Name, and E-mail. Then the system sends them an E-mail with the password they use to login with.

#### **4.1.3 Edit User Data**

Logged in users can change their email, first name, last name, and password.

#### **4.1.4 Generate Questions**

Logged in users can navigate to the generate questions page then after entering the context (the text that the questions are based on) and choosing the type(s) (WH-questions, MCQ, Complete, and True or false) of questions to generate; after waiting for few seconds the questions with their answers are shown.

### **4.2 Techniques and algorithms**

#### **4.2.1 User Data Storage**

User data is stored in a Microsoft SQL server database with the password hashed. When logging in the entered password is hashed then compared with the stored one.

#### **4.2.2 Generate Questions**

We have two different pipelines, one that generates WH-questions and true or false, and another that generates MCQ, Complete, true or false.

In the first pipeline we utilize four models in the backend. We start with the answer extraction model to extract possible answers then the pipeline is divided in two branches the first: we use interrogative word classifier to get the interrogative word for each answer then the question generation model takes the context, answer, and interrogative word to generate questions. After that, one more time it takes the answers only to generate different questions. The second branch takes the context and answer to generate boolean questions.

In the second pipeline we utilize one model which takes the context to generate distractors then we omit the answers to generate complete questions and add the choices to generate MCQ. Additionally, we replace the omitted text with distractors to generate true and false questions.

## 4.3 New Technologies and Architecture

### 4.3.1 Project Architecture

In this project, we used clean architecture which is divided into multiple layers.

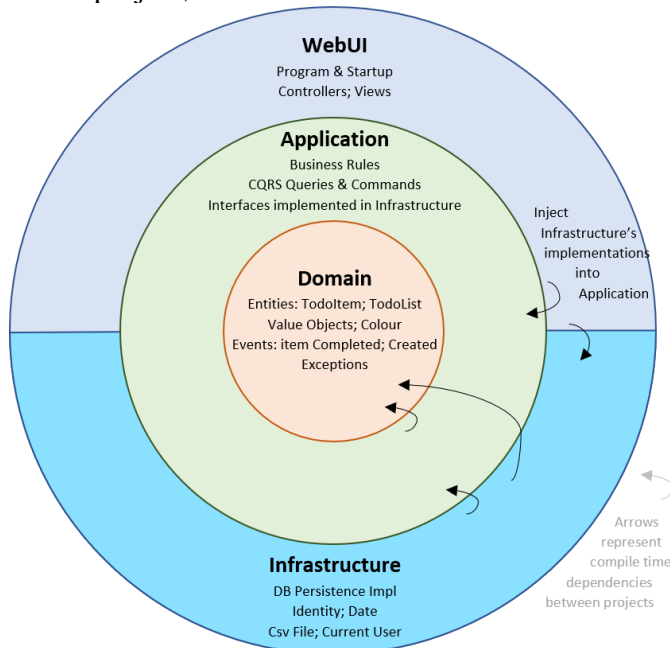


Figure 18 Clean Architecture

#### Domain

The Domain layer only depends on .NET CLR primitive types (int, string, Exception...). Application's aspects like persistence, business rule, network access... etc. are absent from the Domain.

#### Application

The Application layer defines business rules based on the Domain's entities and value objects. It also contains CQRS queries and commands and some interfaces that are implemented in the infrastructure layer.

#### Infrastructure

The infrastructure layer contains implementations of infrastructure interfaces defined in the Application layer. This separation makes it convenient to change any infrastructure implementation at any time.

#### UI

The UI contains Controllers and Pages. It depends on the infrastructure only to inject Infrastructure's implementations within the Application project at startup time. This bootstrap approach makes it easy to maintain and refactor.

### 4.3.2 New Technologies

#### **Backend**

We used ASP.NET Core 6 in backend. ASP.NET is a server-side technology used for developing dynamic websites and web applications. ASP.NET aids developers to create web applications by using HTML, CSS, and JavaScript. ASP.NET is the latest version of Active Server Pages, which Microsoft developed to build websites. It is a web application framework released in 2002.

We used SQL server as our database management system. SQL Server is a relational database management system (RDBMS) developed and marketed by Microsoft. As a database server, the primary function of the SQL Server is to store and retrieve data used by other applications. SQL Server consists of two main components:

- Database Engine
- SQLOS

The core component of the SQL Server is the Database Engine. The Database Engine consists of a relational engine that processes queries and a storage engine that manages database files, pages, indexes, etc. The database objects such as stored procedures, views, and triggers are also created and executed by the Database Engine.

The relational engine contains the components that determine the best way to execute a query. The relational engine is also known as the query processor. The relational engine requests data from the storage engine based on the input query and processed results. Some tasks of the relational engine include querying processing, memory management, thread and task management, buffer management, and distributed query processing. The storage engine oversees storage and retrieval of data from the storage systems such as disks.

We used Python.Net version 3.0.1 library to run Python code in C# with barely any difference in performance. It creates a python environment for Itself only then runs code on it. It allows an easy and smooth integration between C# and python.

The following are the Python libraries used in the backend:

Package Name	Description	Version
NLTK	is a suite of libraries and programs for symbolic and statistical natural language processing for English	3.8.1
SENSE2VEC	sense2vec (Trask et. al, 2015) is a nice twist on word2vec that lets you learn more interesting and detailed word vectors. This library is a simple Python implementation for loading, querying and training sense2vec models.	2.0.2
SPACY	spaCy is an open-source software library for advanced natural language processing.	3.5.3
PYTORCH	PyTorch is a machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing.	1.13.1
TRANSFORMERS	provides APIs and tools to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch.	4.30.2
TENSORFLOW	TensorFlow is a free and open-source software library for machine learning and artificial intelligence.	2.11.0
word2number	Word2number is a free and open-source software library to convert number words (e.g., twenty-one) to numeric digits (21). It works for positive numbers up to the range of 999,999,999,999 (i.e., billions).	1.1

## Frontend

We used the following technologies in the frontend:

1. Angular CLI version 14.1.3, and Angular version: 14.2.7:

Angular is an open-source JavaScript framework written in TypeScript. Google maintains it, and its primary purpose is to develop single-page applications. As a framework, Angular has clear advantages while also providing a standard structure for developers to work with. It enables users to create large applications in a maintainable manner. Angular has many features which facilitate the front-end development process. The first feature is the Document Object Model. DOM treats and XML or HTML document as a tree structure in which each node represents a part of the document. Angular uses regular DOM. Consider that ten updates are made on the same HTML page. Instead of updating the ones that were already updated, Angular will update the entire tree structure of HTML tags. Another feature Angular offers is Data Binding. Data Binding is a process that enables users to manipulate web page elements through a web browser. It uses dynamic HTML and does not require complex scripting or programming. Data binding is used in web pages that include interactive components, such as calculators, tutorials, forums, and games. It also enables a better incremental display of a web page when pages contain a large amount of data. Angular uses the two-way binding.

2. Node version: 16.17.1
3. npm version 8.15.0

## 4.4 UI Design and Wireframes

### ○ Overview

This section provides an overview of the design and wireframes for the project, outlining the visual and interactive elements that will be incorporated into the final product. It serves as a guide for the development team to understand the intended look and feel of the application.

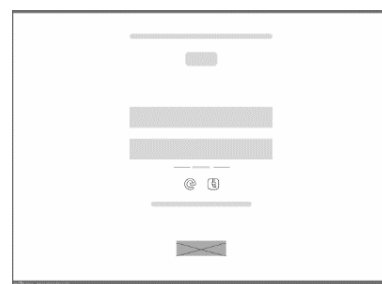
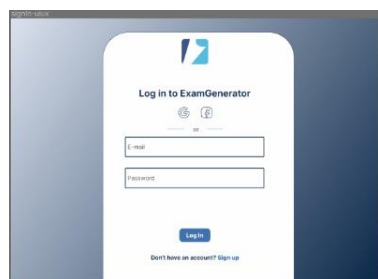
### ○ Design Goals

The design goals for this project are to create a modern, intuitive, and user-friendly interface that aligns with the brand's visual identity. The following design principles will be followed throughout the development process:

- **Consistency:** Maintain a consistent visual style, layout, and interaction patterns across the application.
- **Simplicity:** Strive for simplicity and clarity in design elements and user interactions. **Usability:** Ensure the interface is easy to understand and navigate, minimizing user confusion.
- **Accessibility:** Adhere to accessibility standards to make the application inclusive for users with disabilities.
- **Brand Identity:** Reflect the brand's unique identity through the use of colors, typography, and visual elements.

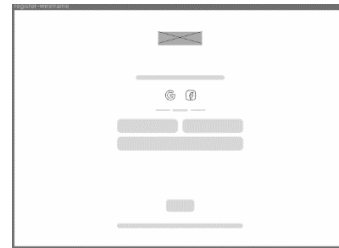
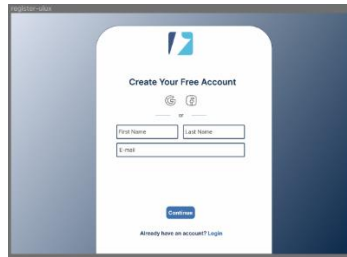
### ○ Wireframes

Wireframes are low-fidelity visual representations that illustrate the layout and structure of the application's screens. They focus on content and functionality rather than precise visual details. The wireframes will be created using Figma and will be iteratively refined throughout the design process.

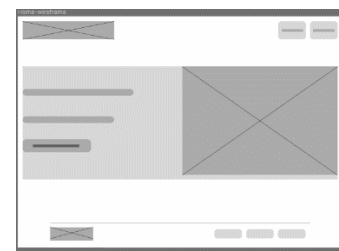
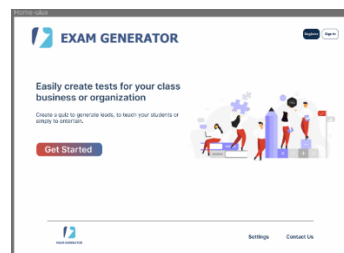


Log in page wireframe and UI design shows how the user could log in into the website.

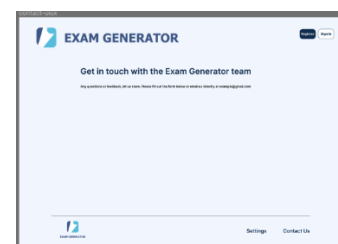




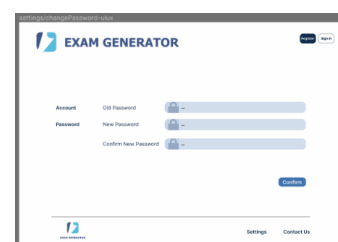
Register page UI and wireframe shows the ability to register with basics information which is first, last name and email.



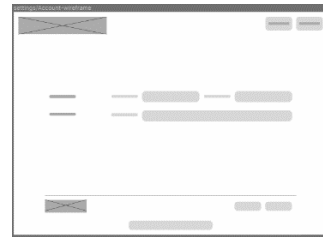
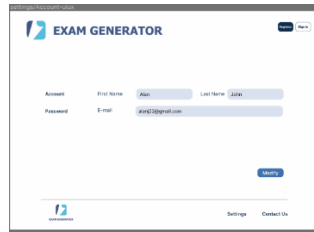
After logged in the home page will appear and the user could discover the website



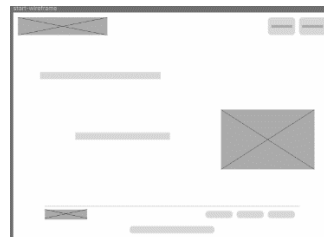
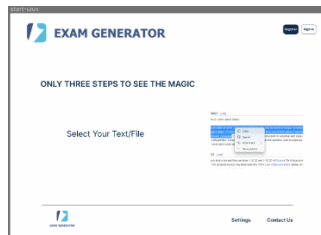
The Contact us page design and wireframe give the ability to the user to contact the development team by showing the email address which the user can copy and use it to send his/her complaint.



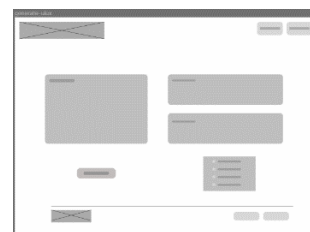
From settings there are two pages the user could enter to it: one of them is which the figure shows above, change password page.



The second page is the account page which shows the details of the user account.



The start page shows the steps to the user on how he/she could use the website in simplest way, after scrolling down to see the instructions the get started button will appear to take the user to Generate page.



The generate page is the main page which contains the main functionality of the website “generate exam” the user could:

- 1- Provide a text in the yellow text area.
- 2- Select the type of questions from the check box.
- 3- Lastly press the generate button to finally the website generates the questions and answers in terms of pairs.

## 4.5 Testing Procedures and Levels

The project encompassed three levels of testing, namely unit testing, integration testing, and system testing. In the unit testing phase, each class was individually tested along with its methods and data entries, including the database modules. Integration testing involved combining all the units and examining their interactions. This level consisted of two types of testing: component integration testing, which focused on interactions and interfaces between integrated components, and system integration testing, which concentrated on interactions and interfaces between systems and packages. In general, the integration testing phase encompassed testing the database, API, and project interfaces. System testing involved evaluating a fully integrated version of the project. The objective of the system test was to assess the system's end-to-end tasks, as well as its non-functional behaviors during the execution of those tasks. In general, the project underwent comprehensive testing, spanning the entire project cycle from signup and login to generating questions and answers based on a given context.

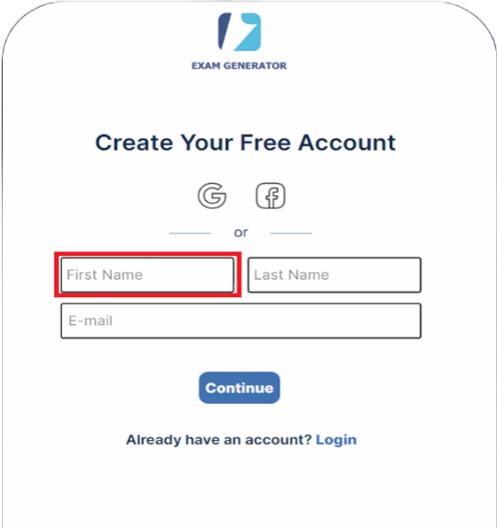
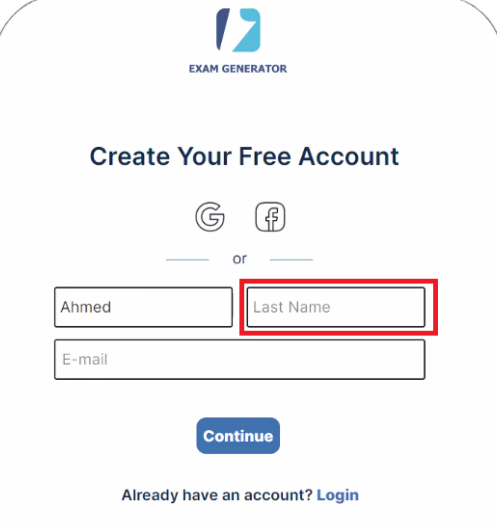
There were primarily two types of testing: manual testing and automation testing. In manual testing, we conducted tests on the locally deployed version of the project using human intervention. This involved manually executing test cases, verifying functionality, and observing the project's behavior to identify any issues or discrepancies. Additionally, we performed exploratory testing to uncover potential defects and evaluate the user experience. The manual testing process allowed us to assess the project's usability, user interface, and overall functionality from a human perspective.

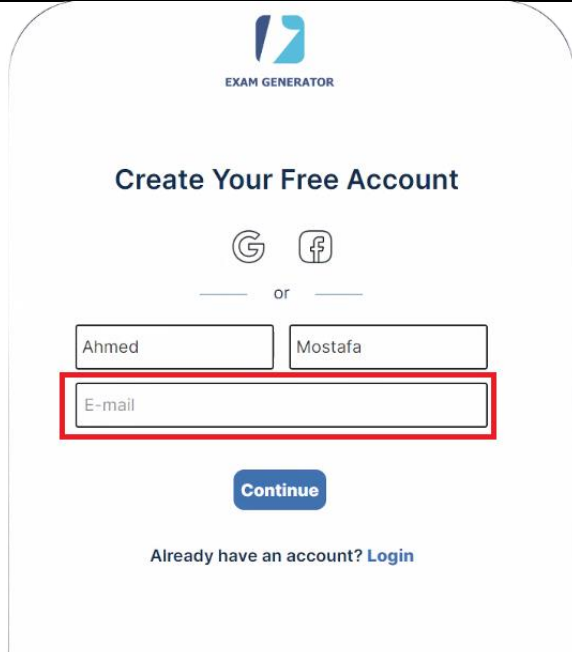
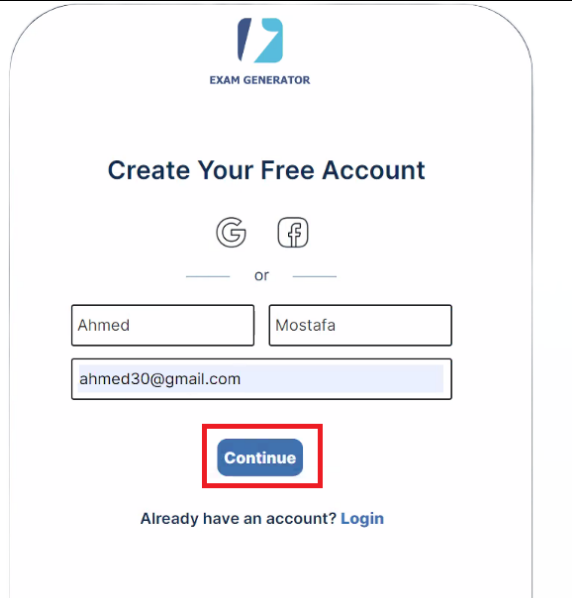
In automation testing, we employed the Selenium framework to create automated test scripts and test suites. We adopted the Page Object Model (POM) design pattern in conjunction with TestNG to generate scripts for each individual page and test their interactions. The POM design pattern allowed us to encapsulate the elements and functionalities of each page into separate classes, promoting code reusability and maintainability. By leveraging TestNG, we were able to organize and execute test cases efficiently, perform assertions, and generate comprehensive test reports. Automation testing with Selenium and POM enabled us to streamline the testing process, increase test coverage, and improve the overall efficiency and accuracy of our testing efforts.




## 5- User Manual & Deployment Guide

### 5.1 User Manual

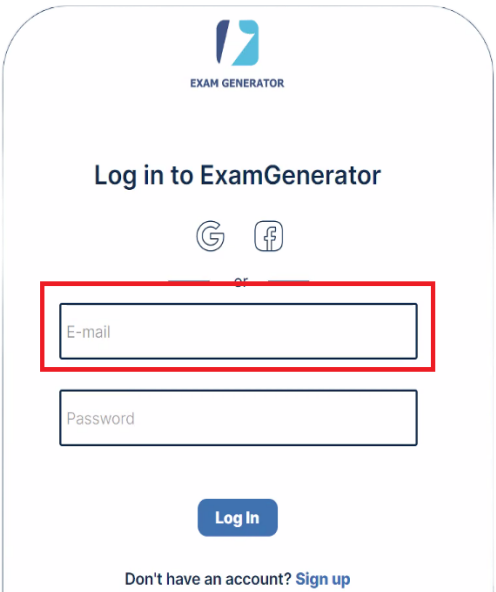
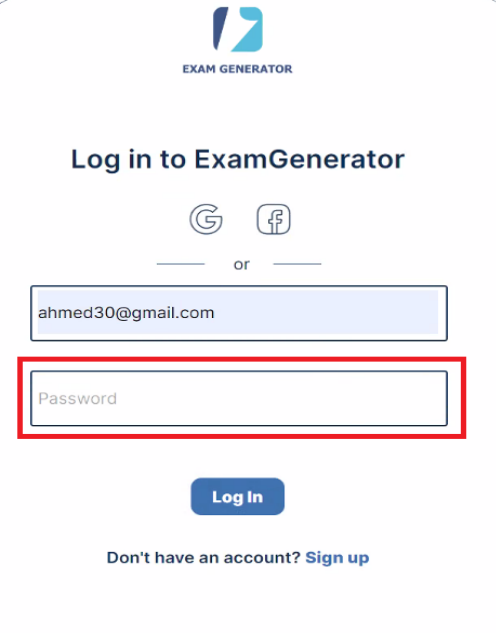
#### Sign up



 <p>The screenshot shows the 'Create Your Free Account' form for 'EXAM GENERATOR'. It includes social login options for Google and Facebook, followed by a 'First Name' field (highlighted with a red box), a 'Last Name' field, and an 'E-mail' field. A 'Continue' button and a 'Login' link are at the bottom.</p>	<p>write your Frist name in Frist name textbox like this image</p>
 <p>The screenshot shows the same 'Create Your Free Account' form. In this instance, the 'Last Name' field is highlighted with a red box. The 'First Name' field contains the text 'Ahmed'.</p>	<p>write second name in second name textbox like this image.</p>

	<p>write your email in the third textbox like this image.</p>
	<p>click continue button like this image.</p>

<div><div> EXAM GENERATOR</div><div>Create Your Free Account</div><div><div></div><div></div><div>or</div></div><div><div><input type="text" value="Ahmed"/></div><div><input type="text" value="Mostafa"/></div></div><div><input type="text" value="ahmed30@gmail.com"/></div><div><div>Continue</div></div><div>Already have an account? <div>Login</div></div></div>	<p>click login to go to login page like this image.</p>
---	---

## Log in

 <p>The image shows the ExamGenerator login interface. At the top is the logo and 'EXAM GENERATOR'. Below is the title 'Log in to ExamGenerator'. There are social media icons for Google and Facebook. Below these is a red-bordered box containing an 'E-mail' input field. Below the email field is a 'Password' input field. At the bottom of the form is a blue 'Log In' button. Below the button is the text 'Don't have an account? Sign up'.</p>	<p>Write your email in email textbox like this image.</p>
 <p>The image shows the ExamGenerator login interface. At the top is the logo and 'EXAM GENERATOR'. Below is the title 'Log in to ExamGenerator'. There are social media icons for Google and Facebook. Below these is a light blue input field containing the email 'ahmed30@gmail.com'. Below the email field is a red-bordered box containing a 'Password' input field. At the bottom of the form is a blue 'Log In' button. Below the button is the text 'Don't have an account? Sign up'.</p>	<p>write your password in password textbox like this image.</p>

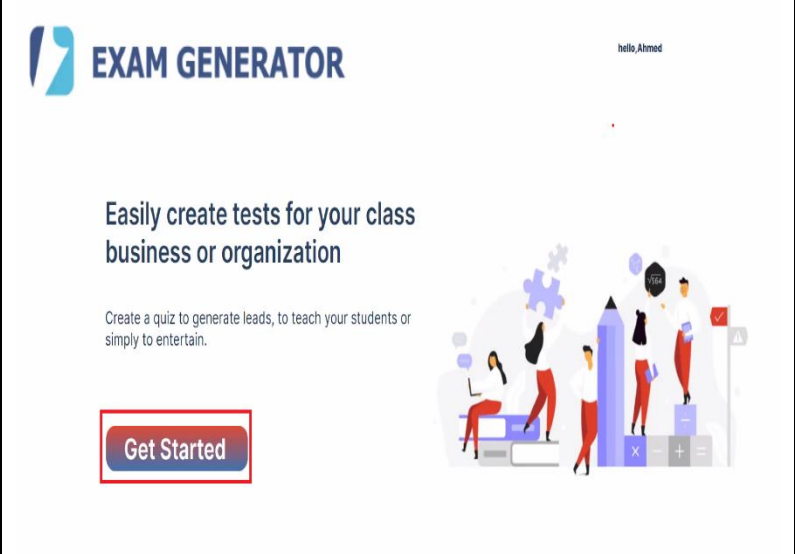
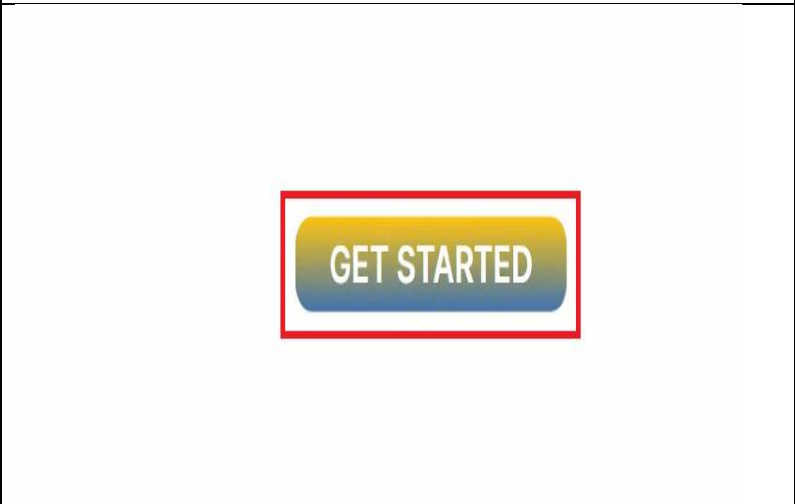
<div><div> EXAM GENERATOR</div><div><h3>Log in to ExamGenerator</h3></div><div><div></div><div></div><div>or</div></div><div><input type="text" value="ahmed30@gmail.com"/></div><div><input type="password" value="....."/></div><div><div>Log In</div></div><div>Don't have an account? <a href="#">Sign up</a></div></div>	<p>click log in button to go to the home page.</p>
--	--



# Modify user details

<div><div><div><div>EXAM GENERATOR</div><div>hello, Ahmed</div></div><div><div>Account</div><div>First Name</div><div>Ahmed</div><div>Last Name</div><div>Mostafa</div></div><div><div>Password</div><div>E-mail</div><div>ahmed30@gmail.com</div></div><div>Modify</div></div></div>	<p>Write your new email here like this image.</p>
<div><div><div><div>EXAM GENERATOR</div><div>hello, Ahmed</div></div><div><div>Account</div><div>First Name</div><div>Ahmed</div><div>Last Name</div><div>Mostafa</div></div><div><div>Password</div><div>E-mail</div><div>newemail@gmail.com</div></div><div>Modify</div></div></div>	<p>Click modify button like this image.</p>

## Home Page

 <p>The screenshot shows the 'EXAM GENERATOR' logo in the top left. In the top right, it says 'hello, Ahmed'. The main heading is 'Easily create tests for your class business or organization'. Below this is a subtext: 'Create a quiz to generate leads, to teach your students or simply to entertain.' A 'Get Started' button is highlighted with a red border. To the right of the text is an illustration of people working together around a large pencil and various icons like a gear, a lightbulb, and a checkmark.</p>	<p>Click get started like this image.</p>
 <p>A close-up of the 'GET STARTED' button, which has a yellow-to-blue gradient and is enclosed in a red rectangular border.</p>	<p>Read the steps guide then Click get started like this image.</p>



## Generate questions page

<div data-bbox="220 286 558 656"><p>Type here...</p></div> <div data-bbox="331 701 448 750"><p>GENERATE</p></div> <div data-bbox="726 701 893 842"><p><input type="checkbox"/> WH-Questions <input type="checkbox"/> MCQ <input type="checkbox"/> Complete <input type="checkbox"/> True or False</p></div>	<p>Write the context that you want to generate questions and answers on, in textbox like this image.</p>
<div data-bbox="228 913 566 1272"><p>Lionel Messi, often regarded as one of the greatest footballers of all time, has left an indelible mark on the world of soccer. Born on June 24, 1987, in Rosario, Argentina, Messi's talent was evident from a young age. Standing at just 5 feet 7 inches, his stature belied his exceptional skill and agility on the field. Messi's mesmerizing dribbling ability, lightning-fast pace, and unmatched vision made him a formidable force on the pitch. Throughout his illustrious career, mainly spent at FC Barcelona, Messi accumulated</p></div> <div data-bbox="331 1339 448 1388"><p>GENERATE</p></div> <div data-bbox="758 1321 956 1503"><p><input type="checkbox"/> WH-Questions <input type="checkbox"/> MCQ <input type="checkbox"/> Complete <input type="checkbox"/> True or False</p></div>	<p>Choose the types of the questions that you want to generate.</p>

Lionel Messi, often regarded as one of the greatest footballers of all time, has left an indelible mark on the world of soccer. Born on June 24, 1987, in Rosario, Argentina, Messi's talent was evident from a young age. Standing at just 5 feet 7 inches, his stature belied his exceptional skill and agility on the field. Messi's mesmerizing dribbling ability, lightning-fast pace, and unmatched vision made him a formidable force on the pitch. Throughout his illustrious career, mainly spent at FC Barcelona, Messi accumulated

GENERATE

- ☒ WH-Questions
- ☐ MCQ
- ☒ Complete
- ☐ True or False



Lionel Messi, often regarded as one of the greatest footballers of all time, has left an indelible mark on the world of soccer. Born on June 24, 1987, in Rosario, Argentina, Messi's talent was evident from a young age. Standing at just 5 feet 7 inches, his stature belied his exceptional skill and agility on the field. Messi's mesmerizing dribbling ability, lightning-fast pace, and unmatched vision made him a formidable force on the pitch. Throughout his illustrious career, mainly spent at FC Barcelona, Messi accumulated

GENERATE

- ☒ WH-Questions
- ☐ MCQ
- ☒ Complete
- ☐ True or False

Click on generate button like this image.

Wait for the loading bar until it finishes.

Lionel Messi, often regarded as one of the greatest footballers of all time, has left an indelible mark on the world of soccer. Born on June 24, 1987, in Rosario, Argentina, Messi's talent was evident from a young age. Standing at just 5 feet 7 inches, his stature belied his exceptional skill and agility on the field. Messi's mesmerizing dribbling ability, lightning-fast pace, and unmatched vision made him a formidable force on the pitch. Throughout his illustrious career, mainly spent at FC Barcelona, Messi accumulated

Elaborate:  
When was Messi born?

Answer: June 24

GENERATE

☒ WH-Questions

☐ MCQ

☒ Complete

☐ True or False

The questions and answers are generated here, and you can scroll down to see all questions and answers like this image.

## 5.2 Deployment

We prepared all deployment files there is no need to do the build steps.

### 5.2.1 IIS Setup

1. Open turn windows features on or off
2. Under .NET framework 4.8 Advanced Services check the following ASP .NET 4.8
  - a. Under WCF Services check the following:
    - i. HTTP Activation
    - ii. TCP port sharing
3. Check Internet Information Services Hostable Web Core
4. Under Internet Information Services check the following
  - a. Under Web Management Tools
    - i. Check IIS Management Console
  - b. Under World Wide Web Services check the following
    - i. Under Advanced Development Features check the following:
      1. .Net Extensibility 3.5
      2. .Net Extensibility 4.8
      3. ASP
      4. ASP.Net 3.5
      5. ASP.Net 4.8
      6. ISAPI Extensions
      7. ISAPI Filters
    - ii. Under common HTTP Features check the following:
      1. Default Document
      2. Directory Browsing
      3. HTTP Errors
      4. HTTP Redirection
      5. Static content
    - iii. Under Health and Diagnostics check the following:
      1. HTTP Logging
    - iv. Under Performance Features check the following:
      1. Static Content Compression
    - v. Under Security check the following:
      1. Request Filtering
5. Click ok.
6. Done.

### 5.2.2 Python Environment

The python environment is already compressed with the deployment files uncompressed at the following path:-

C:\Users\[user]\AppData\Local\

The models are also compressed, uncompress them in any folder and get their paths.

### 5.2.3 Frontend Angular Code Building

#### - Building Step

1. Navigate to Source Code
2. Make sure node\_modules are installed
  - a. If not type: npm install
  - i. If npm install gives an error then type npm install --legacy-peer-deps instead
3. Type npm run publishNoHashTest
4. The build will be found in this directory dist folder

#### - Deployment Step

1. Open IIS
2. Add new website choose ports (2999)
3. Choose physical path .../Frontend
4. Start website

### 5.2.4 Backend .NET (C#) Code Building

#### - Building Step

1. Navigate to Source Code
2. Right-Click on EGService.WebAPI
3. Choose publish
4. Choose build directory
5. Publish

#### - Deployment Step

1. Open IIS
2. Add new website choose ports (port:46770)
3. Choose physical path ... / Backend
4. Open appsettings.json
  - a. Change connection string to your database
  - b. Change model paths to the path of saved models
  - c. Change the PythonDLL path to C:\Users\[user]\AppData\Local\python-3.7.3-embed-amd64\python37.dll
5. Start website



## **6- Conclusion and Future Work**

### **6.1 Conclusion**

This research project presents a comprehensive and innovative approach to question generation in the field of Natural Language Processing (NLP). By combining natural language understanding (NLU) and natural language generation (NLG) tasks, an end-to-end question generation system has been developed with practical applications in the educational domain. The system comprises an Answer Extraction module and a Question Generation module, each serving specific functions in the question generation process. By extracting question-worthy key phrases from the given content and employing specialized models for different question types, including WH, true/false, MCQ, and complete. The proposed system demonstrates its ability to generate high-quality and contextually relevant questions. Through quantitative analysis and comparison with baseline models, the effectiveness of the system is validated in surpassing existing approaches and its suitability for educational applications. The system's ability to generate high-quality, contextually relevant questions not only streamlines the question creation process but also facilitates the educational experience. Educators can leverage the system to efficiently generate exam-style questions that align with the curriculum and learning objectives. Students, on the other hand, benefit from engaging and challenging questions that promote active learning and deeper understanding of the subject matter.

### **6.2 Future Work**

In forthcoming endeavors, the goal is to enhance the performance of the Boolean Question Generation model through the development of a fresh dataset. Accomplishing this goal can be realized by implementing the methodologies utilized in the Multiple-Choice Question and Complete Question modules. By applying these techniques to the contexts found within a reading comprehension dataset, a new dataset can be constructed that will be utilized for training the model. Additionally, expanding the capabilities of the system beyond the confines of the English language holds promise for the field of education. Leveraging multilingual models and datasets will undoubtedly bring numerous benefits to the system and its applicability in a wider range of linguistic contexts.

# References

- Klein, T., & Nabi, M. (2019). Learning to answer by learning to ask: Getting the best of gpt-2 and bert worlds. arXiv preprint arXiv:1911.02365.
- Sun, X., Liu, J., Lyu, Y., He, W., Ma, Y., & Wang, S. (2018). Answer-focused and position-aware neural question generation. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (pp. 3930-3939).
- Liu, B., Zhao, M., Niu, D., Lai, K., He, Y., Wei, H., & Xu, Y. (2019, May). Learning to generate questions by learning what not to generate. In The World Wide Web Conference (pp. 1106-1118).
- Kang, J., Roman, H. P. S., & Myaeng, S. H. (2019). Let me know what to ask: Interrogative-word-aware question generation. arXiv preprint arXiv:1910.13794.
- Kumar, V., Joshi, N., Mukherjee, A., Ramakrishnan, G., & Jyothi, P. (2019). Cross-lingual training for automatic question generation. arXiv preprint arXiv:1906.02525.
- Stasaski, K., Rathod, M., Tu, T., Xiao, Y., & Hearst, M. A. (2021, April). Automatically generating cause-and-effect questions from passages. In Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications (pp. 158-170).
- Ren, S., & Zhu, K. Q. (2021, May). Knowledge-driven distractor generation for cloze-style multiple choice questions. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 35, No. 5, pp. 4339-4347).
- Kim, Y., Lee, H., Shin, J., & Jung, K. (2019, July). Improving neural question generation using answer separation. In Proceedings of the AAAI conference on artificial intelligence (Vol. 33, No. 01, pp. 6602-6609).
- Zhao, J., Deng, X., & Sun, H. (2019). Easy-to-hard: Leveraging simple questions for complex question generation. arXiv preprint arXiv:1912.02367.
- Krishna, K., & Iyyer, M. (2019). Generating question-answer hierarchies. arXiv preprint arXiv:1906.02622.
- Trask, A., Michalak, P., & Liu, J. (2015). sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings. arXiv preprint arXiv:1511.06388.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research, 21(1), 5485-5551.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
- Subramanian, S., Wang, T., Yuan, X., Zhang, S., Bengio, Y., & Trischler, A. (2017). Neural models for key phrase detection and question generation. arXiv preprint arXiv:1706.04560.

Yuan, W., Yin, H., He, T., Chen, T., Wang, Q., & Cui, L. (2022, April). Unified question generation with continual lifelong learning. In Proceedings of the ACM Web Conference 2022 (pp. 871-881).

Smacchia, P. (2023, March 23). *Clean architecture for ASP.NET core solution: A case study*. NDepend. <https://blog.ndepend.com/clean-architecture-for-asp-net-core-solution/>

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020, October). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations (pp. 38-45).

Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002, July). Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting of the Association for Computational Linguistics (pp. 311-318).

Denkowski, M., & Lavie, A. (2014, June). Meteor universal: Language specific translation evaluation for any target language. In Proceedings of the ninth workshop on statistical machine translation (pp. 376-380).

Lin, C. Y. (2004, July). Rouge: A package for automatic evaluation of summaries. In Text summarization branches out (pp. 74-81).