

MINISTRY OF EDUCATION AND RESEARCH



TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Continuous Improvement Process (CIP) Tool with Semantic Text Similarity Analysis

LICENSE THESIS

Graduate: Teodora Irina MĂRGINEAN
Supervisor: As. Eng. Zoltán CZAKO

2021



FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Teodora Irina MĂRGINEAN**

Continuous Improvement Process (CIP) Tool with Semantic Text Similarity Analysis

1. **Project proposal:** The project described in this paper has the purpose of designing and implementing a system which allows employees in companies to submit innovative or work-related ideas and the leaders to review them. Also, a comparative analysis between state-of-art text similarity algorithms has been performed, which resulted in the integration of the best method into the system to compute the semantic similarity between ideas
2. **Project contents:** Introduction - Project Context, Project Objectives and Specifications, Bibliographic research, Analysis and Theoretical Foundation, Detailed Design and Implementation, Testing and Validation, Testing and Validation, User's manual, Conclusions, Bibliography, Appendix A GUI Mockups
3. **Place of documentation:** Technical University of Cluj-Napoca, Computer Science Department
4. **Consultants:**
5. **Date of issue of the proposal:** November 1, 2020
6. **Date of delivery:** July 8, 2021

Graduate: _____

Supervisor: _____



FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

**Declarație pe proprie răspundere privind
autenticitatea lucrării de licență**

Subsemnatul(a)

_____, legiti-
mat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării _____

elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facul-
tatea de Automatică și Calculatoare, Specializarea _____
din cadrul Universității Tehnice din Cluj-Napoca, sesiunea _____ a an-
ului universitar _____, declar pe proprie răspundere, că această lucrare este
rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor
obținute din surse care au fost citate, în textul lucrării și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au
fost folosite cu respectarea legislației române și a convențiilor internaționale privind drep-
turile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte
comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile admin-
istrative, respectiv, *anularea examenului de licență*.

Data

Nume, Prenume

Semnătura

Contents

Chapter 1	Introduction - Project Context	1
1.1	Project context	1
1.2	Short overview of the system	2
1.3	Short summary of the utilised technologies	3
Chapter 2	Project Objectives and Specifications	4
2.1	Database	6
2.1.1	Functional requirements	6
2.1.2	Non-functional requirements	6
2.2	Backend application	6
2.2.1	Functional requirements	6
2.2.2	Non-functional requirements	8
2.3	Machine Learning application	8
2.3.1	Functional requirements	8
2.3.2	Non-functional requirements	9
2.4	Web application	9
2.4.1	Functional requirements	9
2.4.2	Non-functional requirements	10
Chapter 3	Bibliographic research	11
3.1	Continuous Improvement Process	11
3.1.1	Definition	11
3.1.2	Similar tools	14
3.2	Text Similarity	14
3.2.1	Introduction to Natural Language Processing	14
3.2.2	Text similarity algorithms	15
Chapter 4	Analysis and Theoretical Foundation	20
4.1	Short overview of the necessary architectures, components, algorithms and technologies	20
4.2	Use case modelling	21
4.2.1	Submit Idea	22

4.2.2	View idea details	26
4.2.3	Export Statistics to PDF	27
4.2.4	Edit Idea	28
4.2.5	Review idea	31
4.2.6	Confirm implementation	32
4.3	Description of utilised architectures and protocols	34
4.3.1	MVC Architecture	34
4.3.2	Three Tier Architecture	36
4.3.3	HTTP	37
4.3.4	REST Services	37
4.4	Analysis of the experimented text similarity algorithms	38
4.4.1	Cosine Similarity	39
4.4.2	Approach #1 - GloVe Embeddings and Cosine Similarity	40
4.4.3	Approach #2 - NLTK, Gensim and Cosine Similarity	41
4.4.4	Approach #3 - SBERT and Cosine Similarity	41
4.4.5	Experimental results	43
4.5	Description of utilised technologies	45
4.5.1	ASP.NET Core	45
4.5.2	Python	45
4.5.3	Angular	46
Chapter 5	Detailed Design and Implementation	47
5.1	Architecture	47
5.2	SQL Server Database	48
5.3	Backend application	50
5.4	Machine Learning application	58
5.5	Frontend application	60
Chapter 6	Testing and Validation	65
6.1	Functional testing	65
6.2	Text similarity algorithm validation	71
Chapter 7	User's manual	73
7.1	Installation guide	73
7.1.1	MS SQL database	73
7.1.2	ASP.NET Core Web API	73
7.1.3	Python Machine Learning application	74
7.1.4	Angular frontend application	74
7.2	System utilization	75

Chapter 8	Conclusions	84
8.1	Summary and contributions	84
8.2	Critical analysis of the results	85
8.3	Future improvements	85
Bibliography		86
Appendix A	GUI Mockups	89

Chapter 1

Introduction - Project Context

1.1 Project context

With each passing day, the business market develops into a more and more competitive environment, forcing top companies around the globe to face the difficult challenge of increasing profitability by augmenting production and diminishing spending, regardless of the companies' endeavors, with them being either products, services or both. Nonetheless, boosting productivity, while also saving significant resources such as money or time, sounds far more simple than it actually is in a real-world scenario and may not even be enough to reach a greater profit. If the business wants to not only survive, but thrive and expand, it needs to consider a smarter approach in dealing with this issue, while at the same time invest more in quality, rather than quantity.

One solution to this issue lies in the Continuous Improvement Process (CIP), which came into existence after World War II in Japan, whose economy had been hit hard by the conflict which ended up in its defeat. As the United States wished for the nation to slowly rebuild after the devastation left by the war, the Americans put up a group of leading experts of their own, having proficiency in areas such as statistics, and sent them to Japan to help them revive their economy. The strategy proposed by the experts meant shifting companies' focus on processes rather than outcomes and on the continual improvement of those processes, which could only be achieved by combining the efforts and involvement of every member of the organization in gradually and infinitely correcting imperfections and, after all, growing the business. By the 1970s, many Japanese businesses had adopted this practice and began to see the benefits almost instantaneously, the most popular example being represented by Toyota Production System, which actually came up with its own approaches of continually improving processes, including just-in-time manufacturing (JIT) and Total Quality Management (TQM). Ever since, western companies took notice of the prosperity of Japanese organizations and began to show interest in this new way of handling the business and finally embraced it themselves.

In the present day, CIP is being implemented in many companies, especially in larger organizations, and is considered a standard practice. People who are certified in Quality

management System (QMS), according to ISO 9001, are required to work on the basis of Continuous Improvement Process, as the standard demands the approach be adopted in all fields of the business. Companies employ such people in order to adhere to the international standard and make them responsible for this process.

However, the process responsible cannot implement CIP on their own, as the process needs the involvement of all employees in the organization. As a consequence, dealing with the gathering and keeping track of the improvement points from a substantial number of employees, particularly in large corporations, proves to be quite a troublesome task. Most companies provide quite rudimentary means to their employees for implementing this process with an eye toward cost effectiveness. Such procedures include communicating the improvement element via Google Forms, Microsoft Forms or even e-mail, while watching over its progress and storing it in Microsoft Excel files, along with other improvement ideas. Managing such a process turns out to be more labour intensive than it actually needs to be in this day and age where technology should save time instead of wasting it on simple and repetitive tasks.

Continuous Improvement Process (CIP) Tool will aid its users in the management side, as well as the employee side, in the implementation of this necessary practice. The application will provide an automated interface between these parties, while also centralizing the data and performing similarity verifications on it, thus encouraging more and more employees to provide their input, as well as improving the efficiency of the process and minimizing the time it requires to be powerfully put into practice.

1.2 Short overview of the system

This system is composed of three different applications that work together for the purpose of giving every employee the opportunity to come up with improvement ideas or project proposals and have them reviewed by their leaders in an effective and time saving manner.

The storage of the data is performed using a relational database, so that the information could be easily saved, accessed, managed and updated in a persistent way. The backend of the system is represented by the server which manages the data present in the database by performing logical operations and applying functions, while at the same time ensuring its correctness. The backend is not directly accessed by the user, unlike the frontend. This layer of the system serves as the infrastructure that upholds the provision of the information to the frontend as a service. It corresponds to the data access layer, as well as business logic layer of the system. The server is implemented as a Web Application Programming Interface (API) that exposes a collection of endpoints containing the resources which are handled by the frontend.

The similarity checks are performed using an Application Programming Interface (API) that employs Machine Learning practices and computer algorithms with the aim of computing similarity percentages between ideas. Machine Learning is a branch of Artificial

Intelligence which focuses on applications that have the capacity to automatically learn from large datasets and improve results from experience on themselves without being programmatically specified to do so in an explicit manner. The focus of the algorithms used in this system is on Natural Language Processing (NLP), which is a subset of Machine Learning that deals with the interpretation and generation of text that is normally understood and perceived only by humans in the form of language. This part of the system is included in the backend of the tool. It also accesses the database in order to get information about the ideas that have been previously submitted in the tool. This server works in a similar fashion to the previous one, by providing a smaller number of endpoints to the frontend, where the information is further processed and displayed.

The part of the system that the user interacts with directly is the frontend application, which corresponds to the presentation layer of the tool. This layer is situated above the business logic layer of the system. The frontend is a Graphical User Interface (GUI) that is meant to offer a smooth and effortless experience to the user. The frontend communicates with the backend servers via Hypertext Transfer Protocol (HTTP) requests, which are sent from the frontend to the backend, that processes the requests along with the data present in the database and, in turn, sends responses to the frontend, where they are displayed in a more user-friendly manner.

1.3 Short summary of the utilised technologies

The system meant to enable employees and associates to submit and review ideas in more effective ways was designed to implement only software components that employ some of the latest and most powerful technologies on the market used for the development of such systems. These software components include:

- a Microsoft SQL Server relational database used to save information about the users and their ideas, such as: NT username, department, group, leaders, as well as idea title, current context, target state, description and financial report,
- a server-side Web API developed using C# and ASP.NET Core 3.1 which establishes the models used in the database employing Entity Framework Core and Identity Framework Core, as well as handles the business logic of the tool,
- a Python server application used to compute the semantic similarity between the current idea and ideas already submitted in the tool which adopts several libraries used for determining the similarity between ideas, such as PyTorch, BERT, Sentence-Transformers, scikit-learn, pandas and numpy, as well as libraries used for managing data from the Microsoft SQL Server database and exposing resources as endpoints, such as SQLAlchemy and Flask respectively,
- a client-side application developed using Angular 9, HTML, CSS and TypeScript, as well as several libraries such as PrimeNG, Angular Material and Bootstrap.

Chapter 2

Project Objectives and Specifications

The subject of this thesis consists of the design and the implementation of a system which will allow employees from companies all around the world to put forward their input regarding the Continuous Improvement Process, in the form of improvement ideas or project proposals. The input will be further reviewed by the management in an efficient manner and in a centralized and automated environment.

In addition to this point, the research and the application of a method whose aim is to compute the semantic similarity percentage between the input of the associates of the organization is also an important objective of the tool. This similarity check will save quite a significant amount of time on the management side, where there is always too much to do in too little time, when reviewing ideas. What is more, the employees will be motivated to come up with original proposals, which will be rewarded at a later time with a bonus.

This product will replace the current idea submission and review process done via methods that are harder to use, such as e-mail communication between associates and management, with a more user-friendly and intuitive platform which will centralize all the data and encourage more and more associates to improve their work environment. Unlike manual idea submission, review and storage, this product will be easy to use and, as a consequence, will incentivize more and more associates to be more proactive and to submit more improvement ideas. The tool should be designed as a standalone application and it will be completely independent.

The system is composed of several modules, consisting of only software components. Each module possesses a set of functional specifications, along with a set of non-functional specifications, which will be briefly presented in this chapter. The functional requirements refer to the behaviour of the system. In other terms, it specifies what the system should do. On the flip side, the non-functional requirements, also known as quality attributes, represent the constraints that are imposed on the behaviour that the functional requirements describe.

The following figure is intended to illustrate the high-level structure of the system, while also pointing out the relationships and the connections which have been established between the modules of the tool.

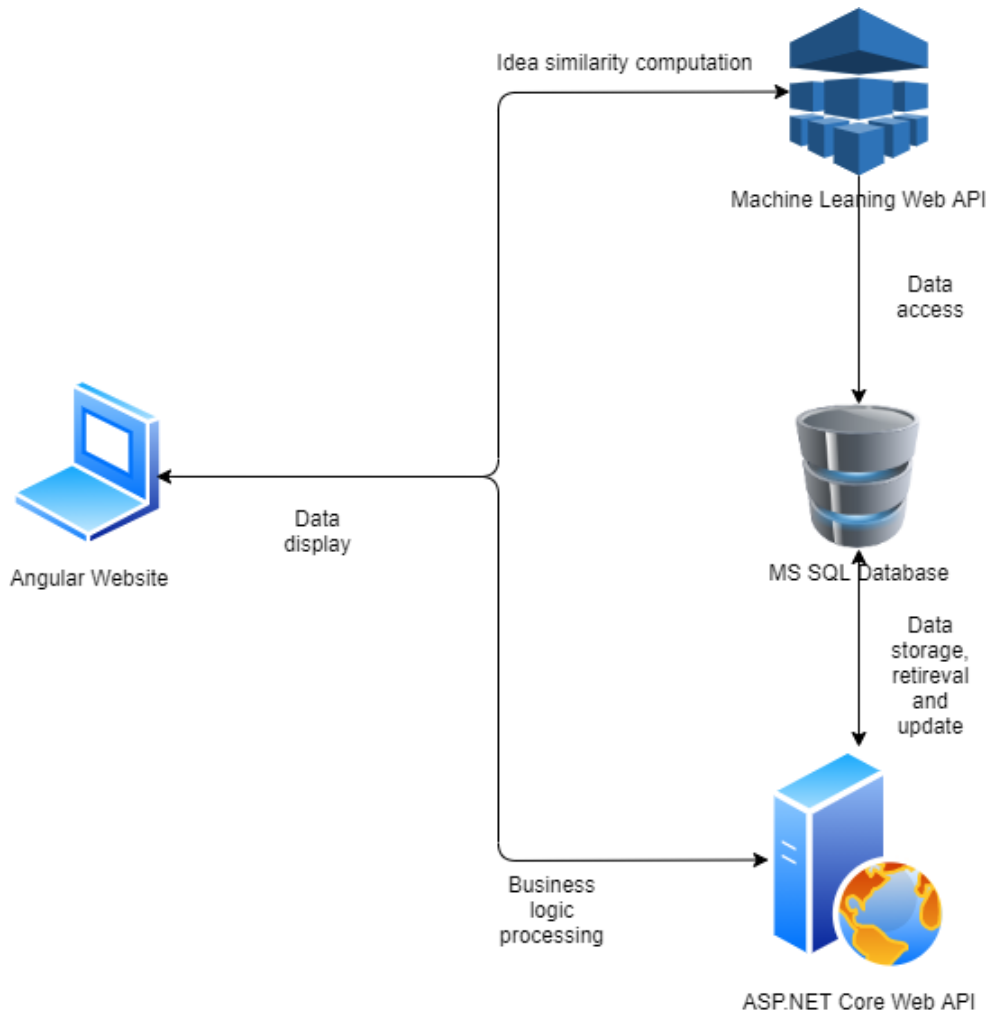


Figure 2.1: The system's structure diagram

As shown in the previous figure, the system consists of four larger modules which have been interconnected in a circular way. The first module is represented by the Microsoft SQL Server Database. This module is connected to the ASP.NET Core Web API in a bidirectional manner, as well as the Machine Learning Web API in a unidirectional manner. The second part of the system is composed of the Angular client application which will be rendered by the browser and whose responsibility lies in the user-friendly display of the data. This module sends requests to the two backend servers of the systems and receives responses from them. The ASP.NET Core Web API is the third module present in the platform. It is responsible for the logical operations on the data is accesses and updates from the MS SQL Database. The fourth part is represented by the Machine Learning Web API, which computes the similarity percentage between the current idea and the ideas already submitted in the tool, by employing Machine Learning practices and algorithms.

2.1 Database

2.1.1 Functional requirements

- The system must store the data in the database in a persistent manner.
- The database must offer the possibility to modify its contents.
- The database must offer the possibility to modify its structure.
- The database must allow applying searching or sorting queries on the data it stores.

2.1.2 Non-functional requirements

- *Security* - The database must be secured with a server name and a password.
- *Availability* - The database must be available 24/7 in any location.
- *Reliability* - The database must be reliable.
- *Recovery* - The database must be backed up on a weekly basis.
- *Performance* - The database must respond to a synchronous request within 1000 milliseconds.
- *Interoperability* - The database must be set up on a Windows machine server.
- The technology used for the database must be Microsoft SQL Server.
- The database must be normalized in the Boyce-Codd normal form (BCNF).

2.2 Backend application

2.2.1 Functional requirements

- The backend application must communicate directly with the frontend application.
- The backend server must be provided with direct access to the database and rights to manage and modify it.
- The backend server must process the data it receives from the client application.
- The backend server must respond to all requests it receives from the client application.
- The Web server must allow authentication for both associates and leaders using a username or an e-mail address, and a password.

- The endpoints the backend server exposes must allow the associates the completion of following operations:
 - The action of adding a new idea, having information such as title, current context, target state, description, idea responsible, attachment files and financial report.
 - The computation of the bonus award from the financial report of an idea.
 - The automatic provision of user related information, such as associate name, group, department, group leader and department leader when submitting a new idea.
 - The delivery of confirmation e-mails when submitting a new idea.
 - The retrieval of the current associate's own ideas as a list of the overview of those ideas.
 - The retrieval of the all ideas existent in the tool as a list of the overview of those ideas.
 - The retrieval of any submitted idea's details.
 - The retrieval of attachment files linked to ideas.
 - The editing of any of the current associate's ideas.
 - The monitoring of statistics regarding the idea statuses, in the form of pie and bar charts.
- The endpoints the backend server exposes must allow the leaders the completion of the following operations, besides the actions an associate can do:
 - The review of an idea of associates from the leader's group. The leader must be able to give a response status and a comment.
 - The retrieval of the pending ideas to review.
 - The retrieval of the previous responses.
 - The delivery of notification e-mails when new ideas are waiting for approval.
- The endpoints the backend server exposes must allow the administrators the completion of the following operations, besides the actions an associate can do:
 - The review of any idea in exceptional circumstances.
 - The retrieval of the pending ideas to review.
 - The retrieval of the previous responses.
 - The delivery of notification e-mails when new ideas are waiting for approval.
 - The editing of any idea.

2.2.2 Non-functional requirements

- *Security* - The backend must be secured using ASP.NET Core Identity. It must implement authentication and authorization for each role (associate, leader, administrator).
- *Portability and compatibility* - The system runs on Windows machines.
- *Performance* - The backend server must respond to the client within 5000 milliseconds.
- *Scalability* - The backend application must be able to continue functioning as expected, even when the work load is large.
- *Testability* - The backend server must easily testable.
- *Clean code* - The coding style must obey the clean code practices.
- The backend server must reply to requests with responses minimal enough to provide sufficient information.
- The backend server must communicate with the frontend via REST Web services.
- The communication protocol must be HTTP.
- The communication must be done using JSON objects.

2.3 Machine Learning application

2.3.1 Functional requirements

- The Machine Learning application must communicate directly with the frontend application.
- The Machine Learning application must be provided direct access to the database and rights to read the data it contains.
- The Machine Learning server must process the data it receives from the client application.
- The Machine Learning server must respond to all requests it receives from the client application.
- The endpoint the Machine Learning server exposes must allow the computation of the semantic similarity between the current idea and ideas previously submitted in the tool.

2.3.2 Non-functional requirements

- *Portability and compatibility* - The system runs on Windows machines.
- *Performance* - The Machine Learning server must respond to the client within 10000 milliseconds.
- The Mean Squared Error of the text similarity algorithm must be smaller than 10.
- *Scalability* - The Machine Learning application must be able to continue functioning well, even when the work volume is large.
- The server must communicate with the frontend via REST Web services.
- The communication protocol must be HTTP.
- The communication must be done using JSON objects.

2.4 Web application

2.4.1 Functional requirements

- The Web application must allow the authentication of any type of user, no matter if they are an associate, leader or admin, using a username or an e-mail address, and a password.
- The client application must save in the browser the current user logged in, even after reloading or refreshing the tool.
- The Web application must allow the associates the completion of following operations:
 - The action of adding a new idea, having information such as title, current context, target state, description, idea responsible, attachment files and financial report.
 - The input added when submitting a new idea must be checked for potential errors.
 - The view of the current associate's own ideas as a list of the overview of those ideas.
 - The view of the all ideas existent in the tool as a list of the overview of those ideas.
 - The view of any submitted idea's details.
 - The download of attachment files linked to ideas.

- The editing of any of the current associate’s ideas.
- The monitoring of statistics regarding the idea statuses, in the form of pie charts and bar charts.
- The Web application exposes must allow the leaders the completion of the following operations, besides the actions an associate can do:
 - The review of an idea of associates from the leader’s group. The leader must be able to give a response status and a comment.
 - The view of the pending ideas to review.
 - The view of the previous responses.
- The Web application must allow the administrators the completion of the following operations, besides the actions an associate can do:
 - The review of any idea in exceptional circumstances.
 - The view of the pending ideas to review.
 - The view of the previous responses.
 - The editing of any idea.

2.4.2 Non-functional requirements

- *Usability* - The graphical user interface must be user friendly, intuitive and easy to use. It must not require any graphical instructions on how to operate the system.
- *Maintainability* - The faults in the system must be easily found and fixed.
- *Scalability* - The application should support the unlimited growth of the number of associates and engineering centers.
- *Security* - The Web pages should be secured with a username or e-mail address, and a password.
- *Clean code* - The coding style must obey the clean code practices.
- *Testability* - The frontend application must easily testable.
- The architectural pattern must be MVC.
- The technology used for the frontend application must be Angular.
- The communication with the backend servers must be done using JSON objects.
- The communication protocol must be HTTP.

Chapter 3

Bibliographic research

3.1 Continuous Improvement Process

3.1.1 Definition

Continuous Improvement Process represents the method which states that the management continually and gradually improves processes, services and products in a suitable, adequate and effective manner, with the purpose of easily adapting to change requests in customer requirements, reducing the cost, price and time of the development and increasing the quality of the outcome and of the work environment, as explained in paper [1].

There are several models of continuous improvement:

- **Six Sigma**

Six Sigma is a model which has the goal of reducing or completely eliminating defects, faults and the variation in processes in order to maintain consistency and improve process robustness and quality. Six Sigma is defined as the ratio of defect products over millions of products and has its statistical success rate of 99.9996%. This model is used especially in manufacturing [2].

- **LEAN Technology**

This model is one of the many others designed by Toyota. It has the aim to optimize the production cycle while also keeping focus on customer requirements at the same time. Its goal is to gain a clear understanding of the most important customer requirements and values in order to remove any part of the development that is considered unnecessary, thus reducing waste in the form of resources, time and costs [3].

- **Kaizen**

Kaizen comes from the Japanese equivalent of the word "improvement" and originated in Japan after World War II. It is the most popular model of Continuous

Improvement and is used all around the world as a standard. Kaizen aims to involve everyone in the business hierarchy, from key essential workers layer, where most wastefulness in the production is identified, to executive layer of the business. Similar to LEAN Technologies, Kaizen has the ultimate goal of cutting back squandering and misspending by incentivising all employees to give suggestions on how processes can be improved. This is a bottom-up, process focused approach [2].

- **Top Quality Management**

Top Quality Management (TQM) is a method similar to both Six Sigma and Kaizen models. Its goal is to implicate all people responsible for the production output in order to reduce errors and faults. This is considered a top down approach.

- **“Fail fast, fail forward”**

This Continuous Improvement Process method refers to the principle of continually learning from mistakes as soon as possible, while the mistakes are still small and comfortable to deal with and solve, with the goal of reaching better solutions. According to this principle, mistakes are merely steps on the way towards favourable outcomes.

It is crucial to state the fact that this mindset is not about wanting to fail at achieving big goals or not being able to deliver, but rather making small steps and testing the waters at early stages in order to handle tiny failures instead of large process deviations and, as a consequence, achieving optimal solutions [2].

- **Perpetual beta**

Perpetual beta is a strategy that implements the Continuous Improvement Process according to which no product or service can ever be faultless or completed, always leaving room for improvement.

This method has its roots in software engineering and development. The most used way to implement and ship software to its users, perpetual beta has shifted in a positive direction from waiting until the implementation is finished and the product is in its final form to delivering the product in a reasonable state at the beginning, then improving it in the future step by step in a continual manner based not only on performance and completeness, but also on customers’ feedback and ever changing needs [2].

This project focuses on the Kaizen method for continuously improving processes. Continuous improvement is the fifth principle of QMS which stands for Quality Management System and represents a set of business processes and policies which focus on customer requirements and their satisfaction. It is implemented through high quality means which consist of the PDCA cycle or Deming’s quality cycle. The PDCA cycle is a process management method that coordinates the continuous improvement in a company

and it represents an important part of it due to its dynamism. The completion of a cycle is considered a step of process enhancement. This ceaseless cycle begins with the mindful planning phase which generates an improvement action that is followed by another step of cautious planning which belongs to the next cycle. It has practical applications in shipping cost, safety, quality of the output and employee and customer satisfaction, as described in paper [4].

the four PDCA cycle steps are described as follows:

- **Plan**

This is the first step in the PDCA cycle. It involves the analysis and planning on what should be changed in order to bring improvements into the processes.

- **Do**

This is the second step in the PDCA cycle. It involves the implementation of the changes planned and decided upon in the Plan phase.

- **Check**

This is the third step in the PDCA cycle. It includes the verification of the changes made in the Do step according to the requirements in the Plan phase, as well as the reporting on the outcomes of the implementation.

- **Act**

This is the fourth and final step in the PDCA cycle and it represents the phase where the changes are adopted and made known for everyone. After this step, the current cycle finalizes and a new one begins with the Plan phase.

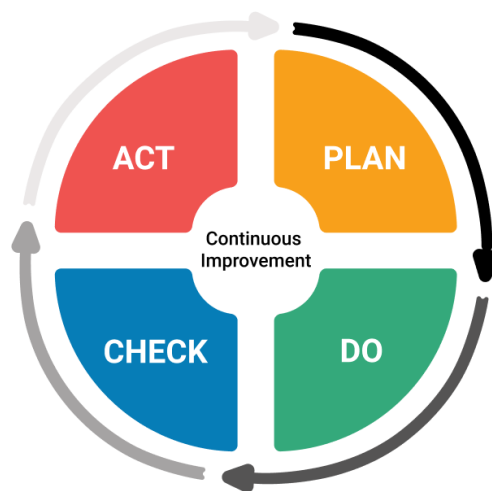


Figure 3.1: PDCA cycle

The goal of the PDCA cycle is improve processes by preventing errors and faults and, if they still happen, correct them in the most effective manner possible, with the purpose of enhancing the quality of the results using the suitable quality assurance tools, as explained in [4].

3.1.2 Similar tools

There are similar systems available on the market, including Impruver, Kainexus, Reverscore, KPI Fire, Braineet and The Lean Way. They provide user friendly interfaces, among with the most important features in the process improvement. However, their price make them rather inaccessible for most companies.

CIP Tool is a cost effective, simple to use and intuitive application which enhances associates' experience when desiring to improve their company or workplace. What is more, it offers the similarity check feature which evaluates the semantic similarities between ideas.

3.2 Text Similarity

CIP Tool's similarity check feature is a key piece of the system. It computes the semantic similarity between the idea an associate is trying to submit against all the other ideas that have already been registered in the tool and displays a warning if similarities are found. This feature's importance comes from the knowledge base that has been shifted from the management part to the system. In this way, the process responsables no longer have to remember if an idea is similar to others or not and the management can be changed without affecting the Continuous Improvement Process.

3.2.1 Introduction to Natural Language Processing

Natural Language Processing (NLP) is a field of Machine Learning and Artificial Intelligence which deals with the translation of the natural language that humans speak to something that machines can understand and further process. NLP has gained a fair amount of attention in recent years and breakthrough developments have been made.

This area includes Low-level NLP tasks, such as Sentence boundary detection, Tokenization and Part-of-speech assignment to individual words, and High-level NLP tasks which are usually problem-specific and built on top of low-level tasks, such as Spelling/grammatical error identification and recovery, Word sense disambiguation (determining a homograph's correct meaning) and Relationship extraction. The most common Machine Learning techniques used in Natural Language Processing are Support vector machines (SVMs), Hidden Markov models (HMMs), Conditional random fields (CRFs) and N-grams, as presented in the article [5].

3.2.2 Text similarity algorithms

In article [6], a brief overview about text similarity algorithms is presented. The similarity measure between pieces of text represents a significant part in research and practical applications in the field of Natural Language Processing. Its related tasks include question generation, question answering, text classification, text summarization, topic detection and plagiarism detection.

The similarity between pieces of text, paragraphs, sentences and phrases is based on the similarity between words. The similarity between words can be lexical or semantic. The lexical similarity consists of character sequences which look the same or word structures which bear some degree of resemblance. In contrast, the semantic similarity applies between words with the same meaning, words with opposite meaning, words in which one is a subtype of the other or words which are generally used in the same context. For instance, the words "click" and "clock" are lexically similar, but they are not semantically similar. On the contrary, the words "laptop" and "computer" are semantically similar, but they are not lexically related.

Lexical similarity algorithms

Lexical similarity is computed using String-Based algorithms. String-Based algorithms are applied on character and string sequences and the similarity metric is used for text matching and comparison. These algorithms are generally split into two major categories: Character-Based algorithms and Term-Based algorithms. The lexical similarity detection techniques are language independent.

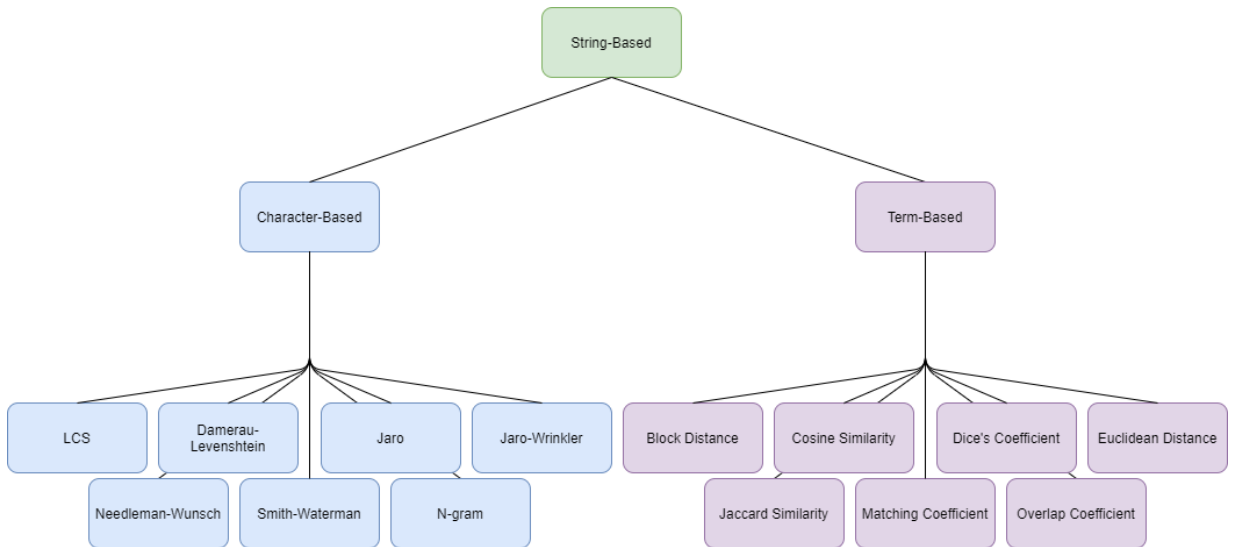


Figure 3.2: String-based similarity hierarchy

Some of the **Character-Based Similarity** measures will be briefly presented below,

according to article [6]:

- **Longest Common Substring (LCS)** is an algorithm which detects the similarity between strings based on the length of common substring sequences which are present in both strings.
- **Jaro** is a similarity algorithm which is based on the common characters between two strings, more specifically on their count and order.
- **N-gram** is a text similarity algorithm which analyzes a piece of text and then takes n element sections or sub-sequences, and then performs a comparison between the generated n-grams from the two input strings.

Some of the **Term-Based Similarity** measures will be briefly presented below, according to article [6]:

- **Block Distance** or Manhattan distance or city block distance algorithm computes the distance from one point to another point in a grid-like environment. The final Block distance is considered to be the sum of differences between the two items.
- **Cosine similarity** is defined as the cosine function applied on the angle formed by two vectors in an inner product space.
- **Jaccard similarity** is computed as the division between the number of common terms and the number of all unique terms in both strings.

Semantic similarity algorithms

Semantic similarity algorithms are split into Corpus-Based and Knowledge-Based measures. The semantic similarity detection techniques are dependent on the language and it requires word ontologies for each language in order to be applicable.

Corpus-Based similarity computes the semantic similarity based on data extracted from large collections of written and spoken text, called corpora.

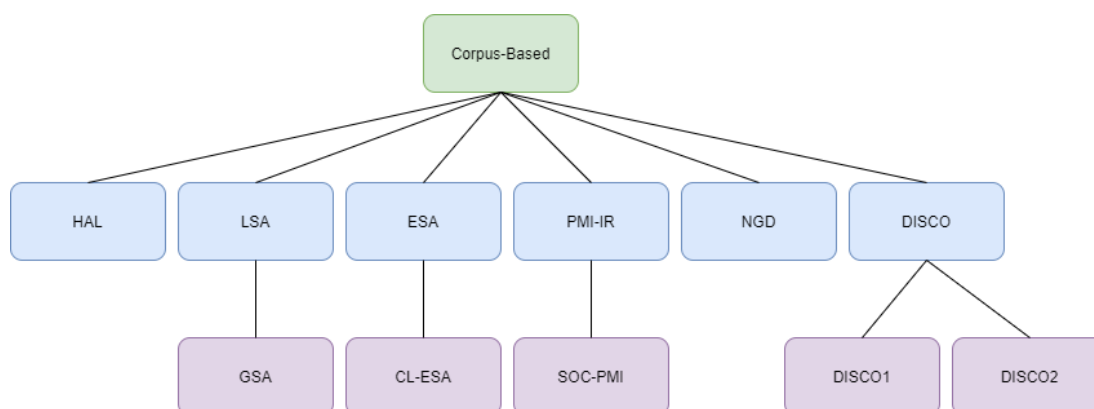


Figure 3.3: Corpus-based similarity hierarchy

Some of the **Corpus-Based Similarity** measures will be briefly presented below, in accordance to article [6]:

- **Latent Semantic Analysis (LSA)** is the most well known and widely used measure in Corpus-Based algorithms. It is a technique which puts into practice the belief that words with the same meaning appear in similar, related or the same writings. It constructs a matrix having the word counts for each paragraph from a large piece of writing then uses a measure used in Mathematics to reduce the number of columns. Then, it uses a method similar to cosine similarity to make the comparison between the pieces of text.
- **Normalized Google Distance (NGD)** detects the semantic similarity between words based on the number of Google results generated by each word or paragraph by a set of keywords. According to this technique, keywords with approximately the same number of Google results are similar in meaning as well.

Knowledge-Based similarity is a type of semantic similarity techniques which uses information gained from semantic networks, where class of words, such as nouns and verbs are grouped together, forming cognitive synonyms called synsets, which represent different topics and concepts.

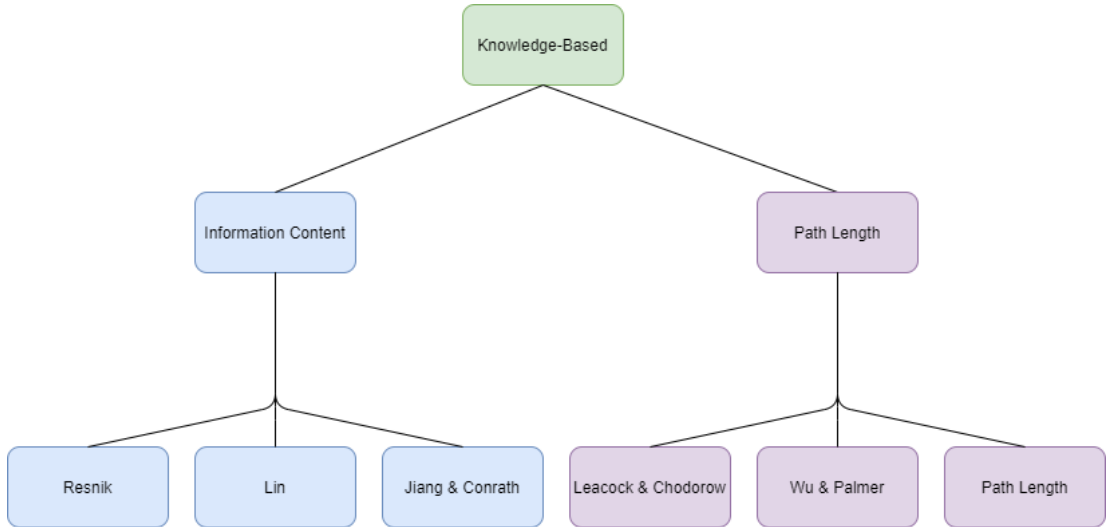


Figure 3.4: Knowledge-based similarity hierarchy

The Knowledge-Based semantic similarity is split into Information Content algorithms including Resnik, Lin and Jiang & Conrath, and Path Length algorithms including Leacock & Chodorow, Wu & Palmer and Path Length.

Resnik, Lin and Jiang & Conrath algorithms use the Least Common Subsumer and its information content, with Resnik considering the value of the similarity equal to the

value of the information content, Lin considering the measure as the sum of the information content of the concepts formed by the two words being compared and Jiang & Conrath considering the difference, as explained in [6].

Leacock & Chodorow computes the semantic similarity as the shortest path between the word meanings and the maximum depth in the taxonomy which contains the word meanings, while Wu & Palmer’s similarity also considers the maximum depth, as well as the maximum depth of the word meanings in the Least Common Subsumer.

Hybrid similarity algorithms

Hybrid similarity algorithms combine some of the methods previously presented, including Character-Based, Term-Based, Corpus-Based and Knowledge-Based approaches with the purpose of obtaining better metrics. Such techniques cover Monge Elkan and SortTFIDF algorithms.

Performance comparison between text similarity algorithms

Paper [7] presents an experiment conducted on three pairs of words with the purpose of comparing the performance of popular text similarity measures. The pairs of words used are:

- **Pair 1** - "book" and "cook", intended for lexical similarity,
- **Pair 2** - "car" and "wheel", intended for semantic similarity,
- **Pair 3** - "antique" and "ancient", intended for both lexical and semantic similarity.

The table below contains the similarity as a value between 0 and 1, with 0 meaning that the words in the pair are unrelated and 1 meaning that the words are similar.

No.	Algorithm/Measure	Pair 1	Pair 2	Pair 3
1	Jaro-Winkler	0.8333	0	0.7714
2	N-gram	0.375	1.0	0.5
3	Cosine similarity	0.4999	0	0
4	Jaccard	0.5	0	0.2
5	LSA	0.1485	0.5080	0.1164
6	Wu Palmer	0.5	0.9091	0.8696
7	Lin	0.1647	0.7355	0
8	Path	0.1429	0.3333	0.25
9	Monge Elkan	0.75	0.4	0.3714
10	SoftTFIDF	0.8333	0	0

Table 3.1: Similarity detected by each algorithm for each word pair

The figure below is a bar chart containing the data extracted from the previous table, having the purpose of obtaining a better visualization of the results of the experiments in the paper.

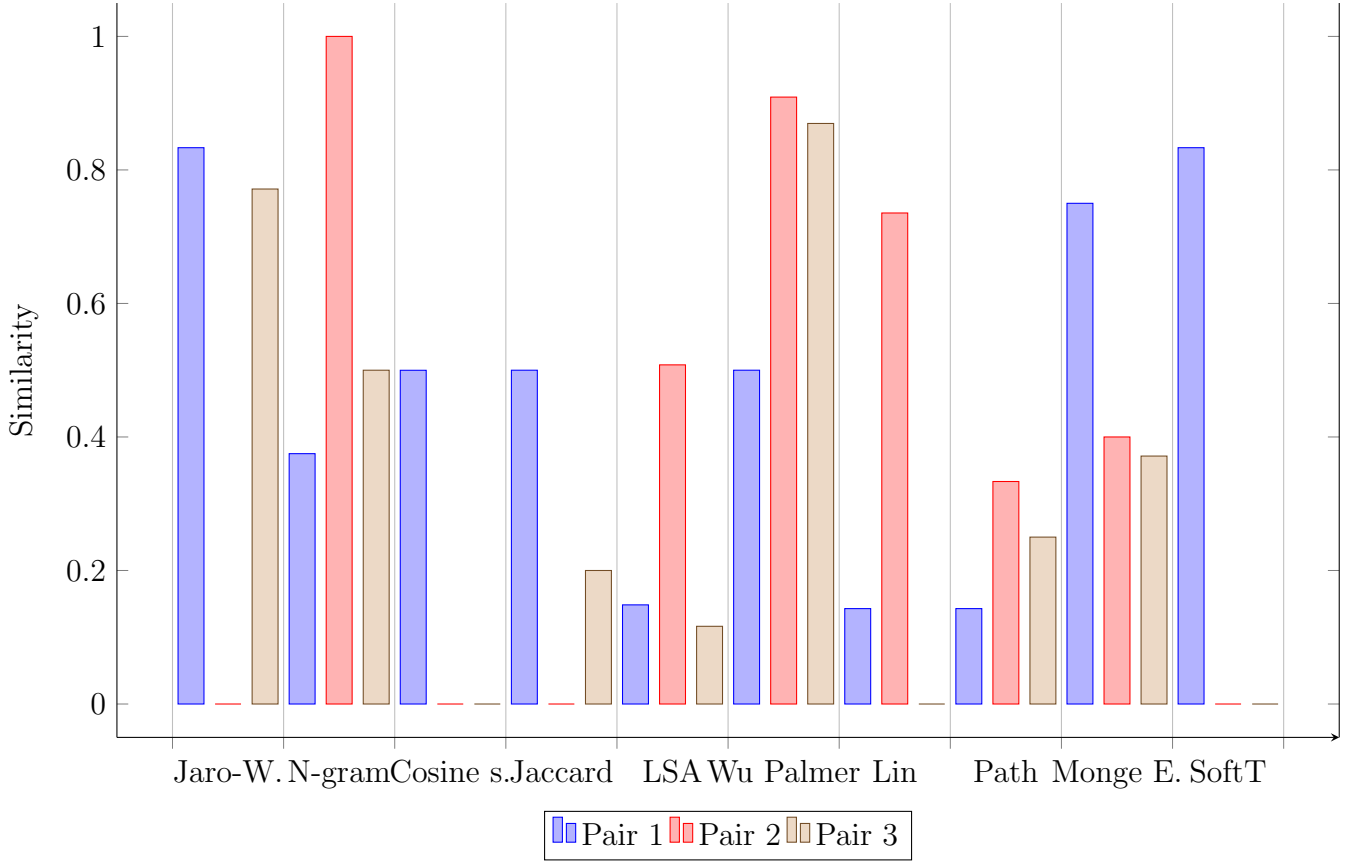


Figure 3.5: Plot of the results in the experiment

Analysing the results of the experiment, some conclusions can be drawn. For Pair 1, containing lexically similar words, Jaro-Wrinkler, SoftTFIDF and Monge Elkan offer the best results, while LSA, Path and Lin offer the poorest similarity percentages. In the case of Pair 2, containing semantically similar words, N-gram and Wu-Palmer yield the most accurate scores, while Jaro-Wrinkler, Cosine similarity, SoftTFIDF and Jaccard similarity detect no similarity between the words in the pair. Regarding Pair 3, containing both lexically and semantically similar words, Wu-Palmer and Jaro-Wrinkler offer the best scores, while Cosine Similarity, Lin and SoftTFIDF cannot detect the similarity between the words.

According to this study and its results, Wu-Palmer appears to be the best technique in detecting the similarity between words, without any regard to the type of similarity between them, being it lexical, semantic or both.

Chapter 4

Analysis and Theoretical Foundation

This chapter contains a detailed analysis of the architectures and components of the system, the algorithms used for detecting the similarity percentage between ideas, the technologies adopted in the application, as well as the use case modelling, all making up the proposed solution for the topic of this paper.

4.1 Short overview of the necessary architectures, components, algorithms and technologies

The preliminary analysis of the problem solved by the system and of the design of the application revealed the primary components which were necessary for the implementation of the proposed solution. These components include:

- A Rest API which ensures a secure and fast connection between the database and the backend. It also communicates with the frontend Web application via HTTP requests by exposing endpoints which can be accessed using the HTTP methods GET, POST, PUT, PATCH and DELETE in a RESTful manner.

The secured connection with the frontend web application is realized using a unique token via the JSON Web Token (JWT) technology, containing the username, role, display name, first name and last name of the current user. The API itself is secured using authentication and role filters. The roles available in the system are Associate, Leader and Admin.

It employs the RESTful API architecture and Design principles, as well as the MVC architecture, concepts which will be detailed in a following section.

The technology chosen for this application is ASP.NET Core 3.1 which offers all the tools necessary to implement the features of this system. The frameworks Entity Framework Core and Identity Framework have been used to model the entities and manage and secure the users.

- A Rest API which employs Machine Learning algorithms used to compute the semantic similarity between an idea and all the ideas which are accessed from the database.

This API communicates with the frontend Web application as well, only via a GET request, which returns a list containing the ideas similar to the current one and the percentage of semantic similarity. It employs the RESTful API architecture and design principles.

The technologies used in this application include Python, Flask, PyTorch, Scikit-Learn, NLTK, pandas, numpy and SBERT.

- A Microsoft SQL Server database responsible for storing user information and their ideas' details. The data in this database can be accessed by both .NET Core and Python Rest APIs, although it can be created, modified or deleted only by the backend API written in .NET Core.

Since the tool's target public is made up of large companies and corporations, the amount of data stored becomes bigger and bigger over time. As a measure for handling the large data size and increasing the performance, the database has been normalized to the Boyce-Codd Normal Form.

- A frontend Web application which will allow the visualization of the ideas in the tool, the statistics generated based on the tools, the submission of a new idea, the edit of already existing ideas and the review of ideas by leaders or administrators.

This application ensures a user friendly and smooth navigation through the features previously mentioned, while also offering a secured environment with authentication and role guards on every page. It communicates directly with both REST APIs and employs the MVC architecture.

The technologies used in this application are TypeScript and Angular 9, as well as the PrimeNG, Bootstrap and Angular Material libraries.

4.2 Use case modelling

This section contains the description of the main use cases of the tool.

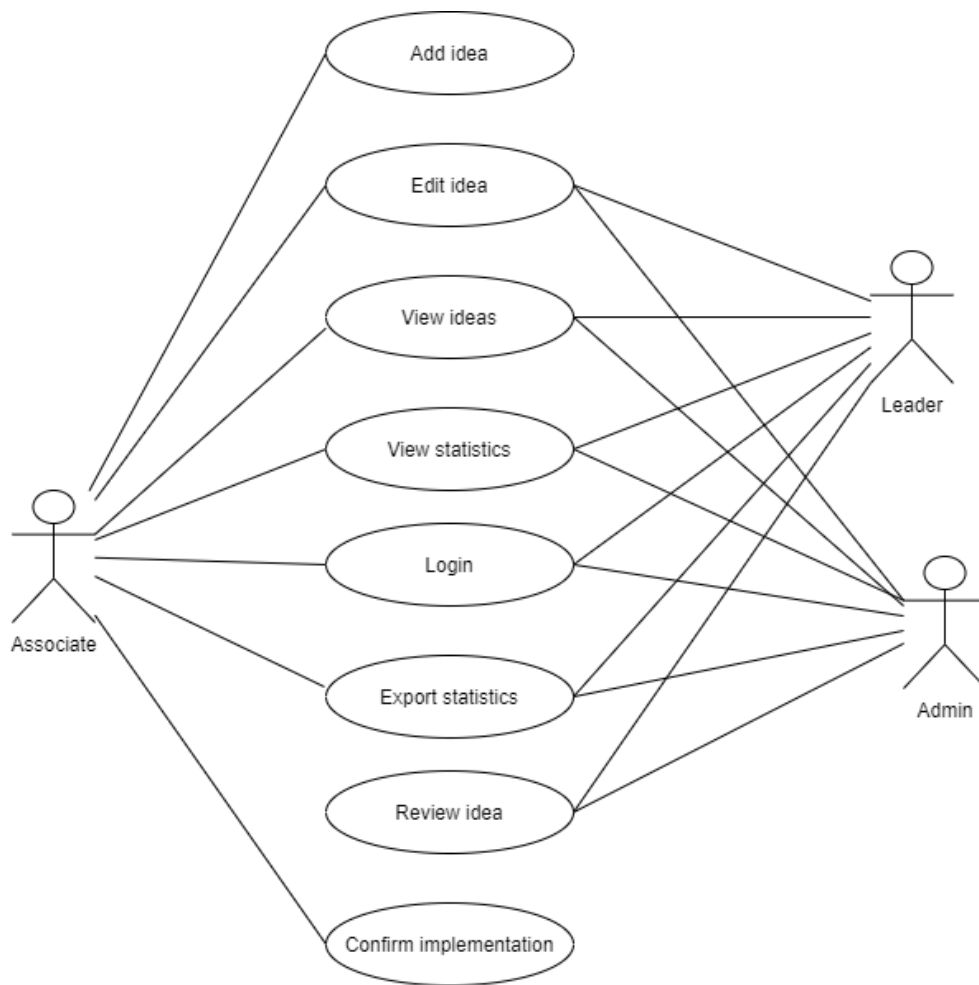


Figure 4.1: Use case diagram of the system

4.2.1 Submit Idea

Brief description: The purpose of this use case is to capture the flow of events that an actor must follow in order to submit a new idea in the CIP Tool for it to be reviewed by their leader.

Primary Actor: Associate

Stakeholders and Interests

- **CIP Responsible** - interested in having an easy to use and intuitive user interface on the idea submission page. It should require no trainings on how to use this page. Error handling on the user's inputs should be done in order not to allow the user to make any mistakes. The user should be notified of any typos in his inputs. Also, the response time should be as short as possible.

- **Department head** - is interested in making the idea submission process as easy as possible.
- **Project Manager** - is interested notifying the user if the idea they want to submit is semantically similar to other ideas already existing in the tool.

Preconditions: The actor is authenticated.

Postconditions: The idea is available in the My ideas and All ideas sections and the idea review request is available in the Leader Response section.

Basic Flow

Use-Case Start

The use case starts when the actor wants to submit a new idea.

1. The actor goes to the Add Idea page.
2. The system displays the Add idea page, along with the pre-filled idea owner fields, such as Name of the idea owner, Group, Group leader, Department and Department leader.
3. The actor completes the Title, Current context, Target state and Description fields.
4. The actor chooses the Idea categories.
5. The actor adds attachments.
6. The system opens a File Explorer window.
7. The actor chooses which files to attach one by one.
8. The actor fills in the Financial Report related fields, such as Planned savings and Planned expenses.
9. The system completes the balance and bonus fields considering the values previously inserted in the Financial Report. The column in the Financial Report table corresponding to the bonus will be highlighted.
10. The actor submits the idea.
11. The system displays the most similar ideas to the current one.
12. The actor confirms the submission.
13. The system sends a confirmation email to the actor.
14. The system sends a review request email to the actor's leader and the administrators.

Use-Case End

The system redirects the actor to the My Ideas page. The idea appears in the Waiting for approval ideas table.

Alternative Flow 1

The actor still has attachments to upload.

Alternative Flow Starts

1. The actor adds a new attachment.
2. The system opens a File Explorer window.
3. The actor chooses which file to upload.
4. Go to step 8 in Basic Flow.

Alternative Flow Ends

The system displays the new attachment in the Attachments section.

Alternative Flow 2

Title field is empty.

Alternative Flow Starts

1. The actor forgets to fill in the Title field.
2. The actor submits the idea.
3. Go to step 4 in Basic Flow.

Alternative Flow Ends

The system notifies the actor that the Title field is mandatory.

Alternative Flow 3

The actor decides they do not want to submit the idea anymore after viewing the similar ideas and the similarity percentages detected by the tool.

Alternative Flow Starts

1. The actor cancels the submission of the idea.
2. Go to step 3 in Basic Flow.

Alternative Flow Ends

The system displays Add Idea page with the previously completed fields already filled in.

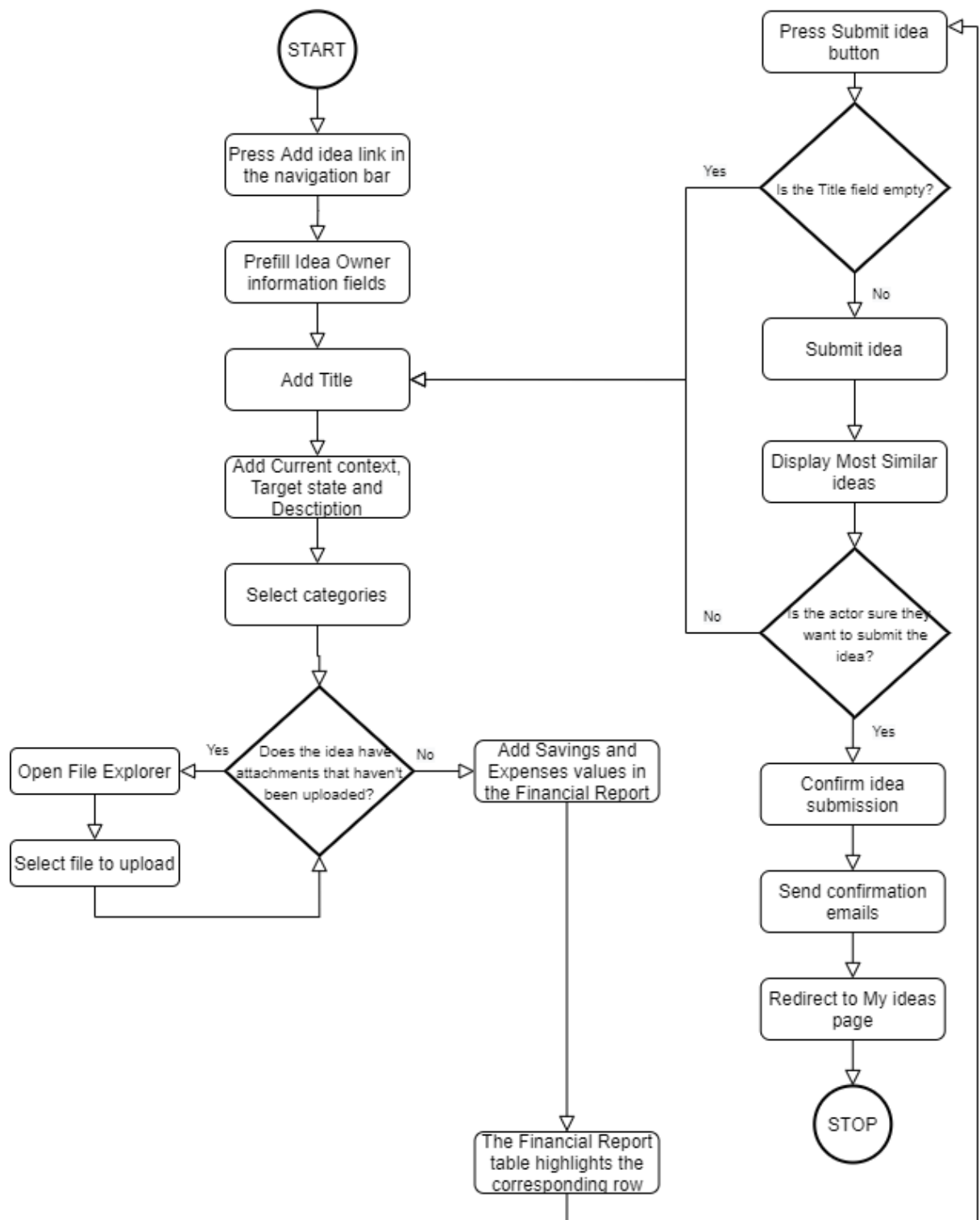


Figure 4.2: Flow diagram of the Submit Idea Use-Case

4.2.2 View idea details

Brief description: The purpose of this use case is to capture the flow of events that an actor must follow in order to view the details of an idea already submitted in CIP Tool.

Primary Actor: Associate

Stakeholders and Interests: The CIP Responsible, Department Head and Project Manager are interested in having an easy to use and intuitive user interface. It should require no trainings on how to reach the idea details page. Also, the response time should be as short as possible.

Preconditions: The actor is authenticated.

Postconditions: The correct idea details are displayed.

Basic Flow

Use-Case Start

The use case starts when the actor wants to view the details of an already submitted idea.

1. The actor goes to the All Ideas page.
2. The actor selects the searched idea from one of the Waiting for approval ideas, Approved ideas, Postponed ideas, Declined ideas and Implemented ideas tables.

Use-Case End

The system redirects the actor to the idea details page of the idea they want to view.

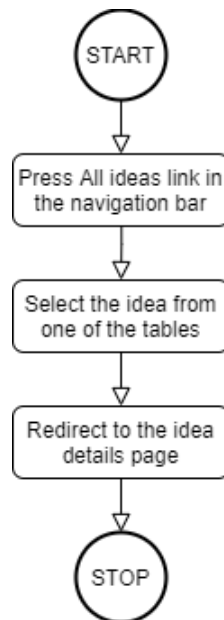


Figure 4.3: Flow diagram of the View Idea Details Use-Case

4.2.3 Export Statistics to PDF

Brief description: The purpose of this use case is to capture the flow of events that an actor must follow in order to export to PDF the idea statistics in CIP Tool.

Primary Actor: Associate

Stakeholders and Interests: The CIP Responsible, Department Head and Project Manager are interested in having an easy to use and intuitive user interface. It should require no trainings on how to export the idea statistics to PDF.

Preconditions: The actor is authenticated.

Postconditions: A PDF containing idea statistics is available to the user.

Basic Flow

Use-Case Start

The use case starts when the actor wants to export the idea statistics generated by the tool to PDF.

1. The actor goes to the Idea Status/Statistics page.
2. The actor exports the idea statistics to PDF.

Use-Case End

The system generates and downloads a PDF containing idea statistics.

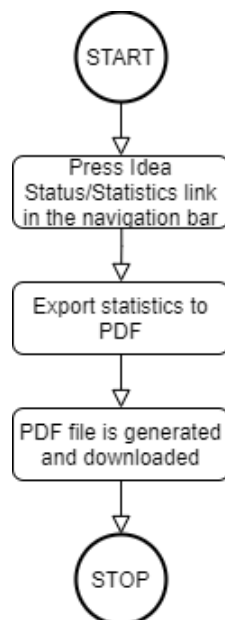


Figure 4.4: Flow diagram of the Export Statistics to PDF Use-Case

4.2.4 Edit Idea

Brief description: The purpose of this use case is to capture the flow of events that an actor must follow in order to edit an idea already submitted in CIP Tool.

Primary Actor: Associate

Stakeholders and Interests: The CIP Responsible, Department Head and Project Manager are interested in having an easy to use and intuitive user interface. It should require no trainings on how to edit an idea.

Preconditions: The actor is authenticated.

Postconditions: The idea fields which have been modified are correctly saved and displayed in the idea details page.

Basic Flow

Use-Case Start

The use case starts when the actor wants to edit an already existing idea.

1. The actor goes to the All Ideas page.
2. The actor selects the idea from one of the Waiting for approval ideas, Approved ideas, Postponed ideas, Declined ideas and Implemented ideas tables.
3. The actor is redirected to the idea details page of the idea they want to edit.
4. The system displays the idea details.
5. The actor enters the edit mode.
6. The actor modifies the Title, Current context, Target state and Description fields.
7. The actor chooses the Idea categories and adds the PDCA dates.
8. The actor removes unwanted attachments.
9. The actor adds new attachments.
10. The system opens a File Explorer window.
11. The actor chooses which files to attach one by one.
12. The actor modifies the Financial Report related fields, such as savings and expenses.
13. The system completes the balance and bonus fields considering the values previously inserted in the Financial Report. The column in the Financial Report table corresponding to the bonus will be highlighted.
14. The actor saves the modifications made to the idea.

Use-Case End

The system redirects the actor to the idea details page where the modifications are available.

Alternative Flow 1

The actor still has attachments to upload.

Alternative Flow Starts

1. The actor adds a new attachment.
2. The system opens a File Explorer window.
3. The actor chooses which file to upload.
4. Go to step 8 in Basic Flow.

Alternative Flow Ends

The system displays the new attachment in the Attachments section.

Alternative Flow 2

The actor decides they do not want to save the changes made to the idea anymore.

Alternative Flow Starts

1. The actor cancels the submission of the idea.
2. Go to step 3 in Basic Flow.

Alternative Flow Ends

The system displays idea details with the fields the same as before entering the edit mode.

Alternative Flow 3

Title field is empty.

Alternative Flow Starts

1. The actor forgets to fill in the Title field.
2. The actor submits the idea.
3. Go to step 4 in Basic Flow.

Alternative Flow Ends

The system notifies the actor that the Title field is mandatory.

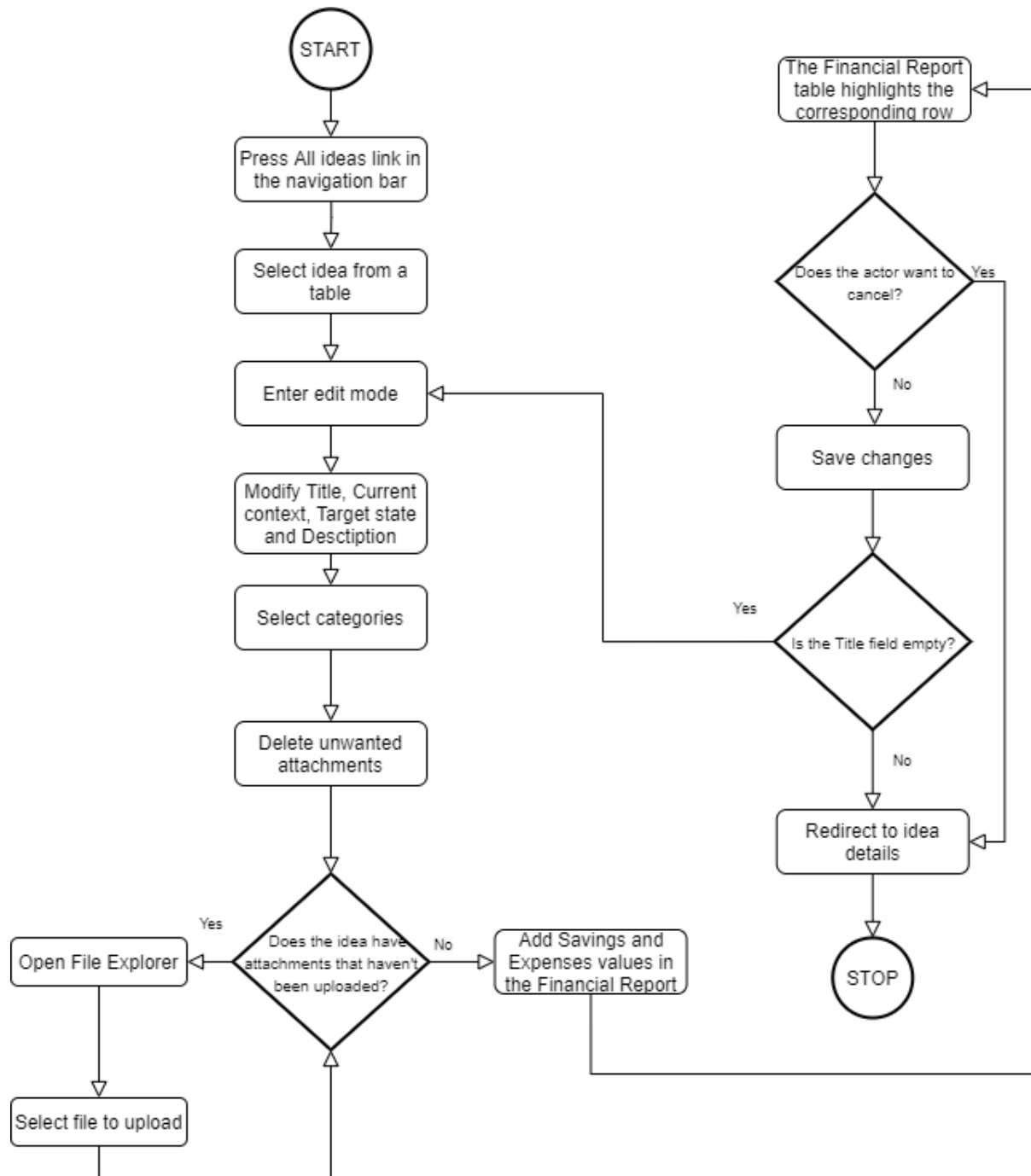


Figure 4.5: Flow diagram of the Submit Idea Use-Case

4.2.5 Review idea

Brief description: The purpose of this use case is to capture the flow of events that an actor must follow in order to review an idea in CIP Tool.

Primary Actor: Leader

Stakeholders and Interests: The CIP Responsible, Department Head and Project Manager are interested in having an easy to use and intuitive user interface. It should require no trainings for the leaders on how to review an idea after being requested to do so.

Preconditions: The actor is authenticated. The actor is either in the Leader or the Administrator role.

Postconditions: The status of the idea changes according to the response left by the actor. A review comment is also saved if available.

Basic Flow

Use-Case Start

The use case starts when the actor wants to review an idea.

1. The actor goes to the Leader Response page.
2. The actor selects the idea they want to review.
3. The actor approves the idea.
4. The actor saves the changes.

Use-Case End

The actor is redirected to the Leader Response overview page, where the status of the idea has been modified.

Alternative Flow 1

The leader wants to postpone or decline the idea.

Alternative Flow Starts

1. The actor adds a comment to support their decision.
2. The actor postpones or declines the idea.
3. Go to step 4 in Basic Flow.

Alternative Flow Ends

The actor is redirected to the Leader Response overview page, where the status of the idea has been modified.

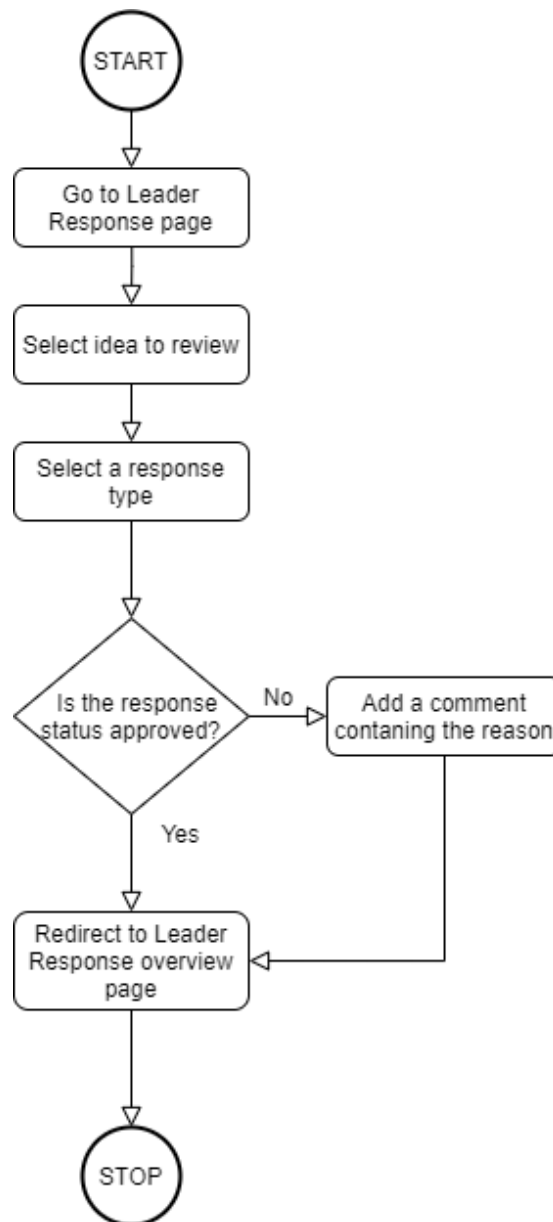


Figure 4.6: Flow diagram of the Review Idea Use-Case

4.2.6 Confirm implementation

Brief description: The purpose of this use case is to capture the flow of events that an actor must follow in order to confirm the implementation of an idea in CIP Tool.

Primary Actor: Associate

Stakeholders and Interests: The CIP Responsible, Department Head and Project Manager are interested in having an easy to use and intuitive user interface. It should

require no trainings how to confirm the implementation of an idea in CIP Tool.

Preconditions: The actor is authenticated. The idea they want to confirm must be in Approved status.

Postconditions: The status of the idea changes from Approved to Implemented.

Basic Flow

Use-Case Start

The use case starts when the actor wants to confirm the implementation of an idea.

1. The actor goes to My Ideas page.
2. The actor selects the idea they want to confirm as implemented.
3. The actor confirms the implementation.

Use-Case End

The actor is redirected to My Ideas overview page, where the status of the idea has been modified.

Alternative Flow 1

The actual implementation date field is empty.

Alternative Flow Starts

1. The actor enters the edit mode.
2. The actor adds the actual implementation date.
3. The actor saves the changes.
4. Go to step 3 in Basic Flow.

Alternative Flow Ends

The actual implementation date is not empty.

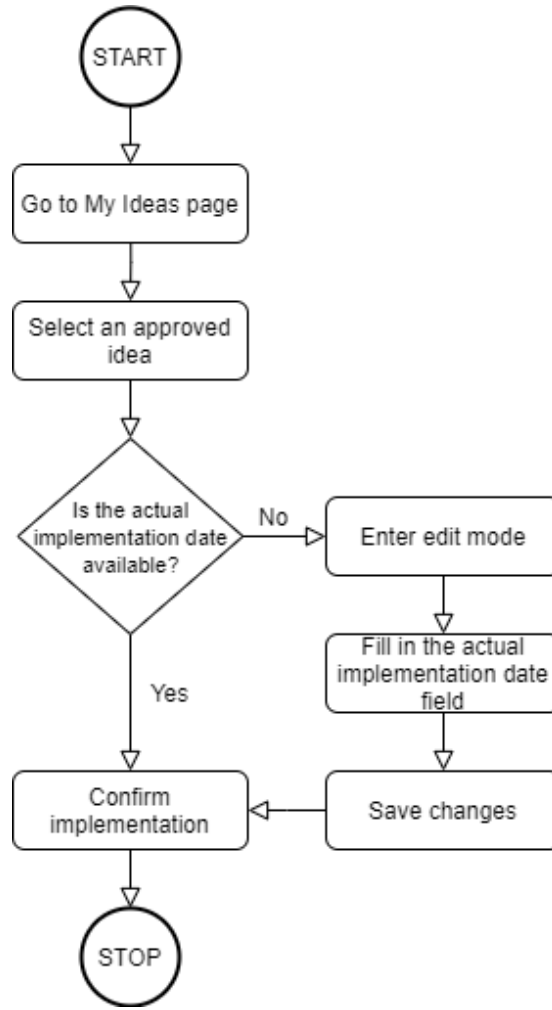


Figure 4.7: Flow diagram of the Confirm Implementation Use-Case

4.3 Description of utilised architectures and protocols

This section of the paper presents the theoretical concepts used in the architectures and protocols employed by the tool in a more detailed manner.

4.3.1 MVC Architecture

MVC stands for Model-View-Controller. In article [8], it is presented as an architectural pattern, which is currently being widely used in many applications. As its name suggests, this paradigm allows the separation of concerns via three main components: Models, Views and Controllers. Separation of concerns ensures that the business logic and the infrastructure of the application have a clear barrier of separation between them and

the user interface.

The diagram below illustrates the three groups of components in the MVC pattern and the connections established between them:

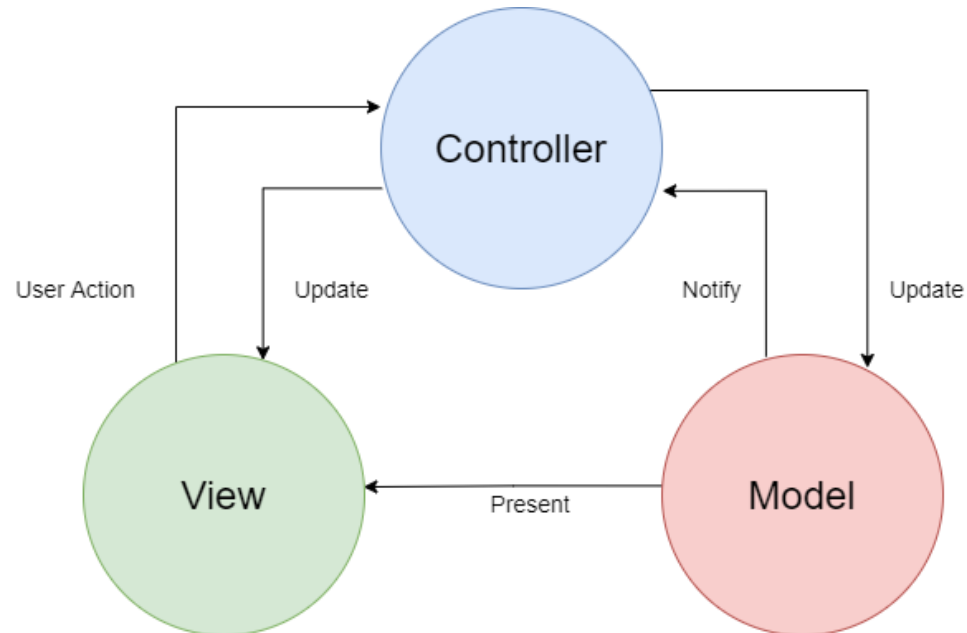


Figure 4.8: Diagram of the MVC Architectural Pattern

As shown in the diagram, when a user completes an action from the View (the user interface), the Controller (the brain) retrieves, creates, modifies or deletes data using the Model. Then, the updated data is returned using the Model in the Controller, then passed and displayed in the View to the user.

The major roles of the three main components are:

- **Model** - represents the state of the application. The operations in the Controller are performed using this data. This data is also transferred between the Controller and the View.
- **View** - represents the User Interface where the data in the Model component is presented to the user.
- **Controller** - represents the brain of the application. It is responsible for dealing with the user interaction which is performed through the View, then applying logical operations on the data using the Model, then presenting the modified data back in the View to the user.

Benefits of the MVC architectural pattern:

- The separation of concerns into three distinct groups of components

- Allows multiple developers to work together in the same project more easily
- Fast application development
- Easy debugging of the application
- Easy testing of the application
- Fast application update
- Errors and bugs are minimized
- Offers multiple Views for the same Model

4.3.2 Three Tier Architecture

Article [9] presents an overview of the Three Tier Architecture, also known as Three Layer Architecture. It is a client-server type of architectural pattern where the application is clearly separated into three distinct layers which encompass the logic of the application, the data access and the presentation part of the application.

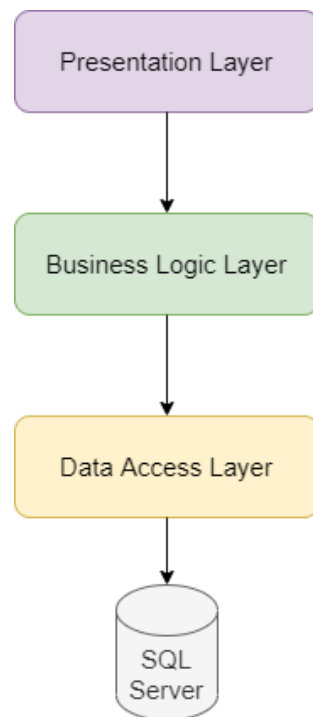


Figure 4.9: Diagram of the Three Tier Architectural Pattern

The three layers in this architectural pattern are:

- **Data Access layer** - is the bottom layer. It contains the database where the storage of the data is performed. This layer is independent of all the other ones, although it is directly accessed only by the Business Logic Layer.
- **Business Logic Layer** - is the middle layer. It contains the business logic permed by application, as its name suggests. It is accesses the Data Access Layer and is accessed by the Presentation Layer.
- **Presentation Layer** - is the top layer. It represents the frontend of the application and displays the data to the user in the form of a Graphical User Interface. The user interacts directly with this layer. It accesses the Business Logic Layer, although it cannot communicate with the Data Access Layer head on.

The critical advantage offered by this type of architectural pattern is that the implementation of any layer can be replaced with another implementation without affecting the other layers.

4.3.3 HTTP

Article [10] defines and describes the HTTP protocol. HTTP stands for Hypertext Transfer Protocol and it is a global standard in the World Wide Web to load web pages via hypertext URLs. HTTP is a protocol found in the Application Layer of the OSI (Open Systems Interconnection) model, which is the closest to the user and the only layer that the user directly interacts with.

Usually, HTTP is used when a client is making a request to a server in order to ask for the data needed to load a web page and receives back a response from the server which contains that information. A request can contain the HTTP version type, a URL, an HTTP method, a body and request headers. Similarly, a response typically contains a status code, HTTP response headers and a body.

The main HTTP request types are:

- **GET** - retrieves a resource
- **POST** - submits a resource to the server
- **PUT** - usually updates a resource
- **DELETE** - deletes a resource

4.3.4 REST Services

Article [11] presents the REST architectural style in more details. It stands for Representational State Transfer and it represents a set of constraints used in the implementation of APIs (Application Programming Interfaces).

The 6 principles in REST architectural style are:

- **Client-server** - the application must implement this architectural style, so that the separation of concerns is respected and the user interface and the business logic have a clear barrier of separation between them.
- **Stateless** - the context on the server must not be considered and the requests must contain all the relevant data in order to receive the desired, as well as correct response from the server. The state of the session must be stored by the client. The requests have no connection between them.
- **Cacheable** - the distinction between cacheable and non-cacheable data must be made implicitly or explicitly within an HTTP response, so that the cacheable data could be reused at a later time.
- **Uniform interface** - a standard imposed in the communication between the components, which is simplified and more visible, including:
 - resources identification
 - resource manipulation via representations, which provide enough information in order to be handled by the client
 - self-descriptive messages
 - hypermedia which allows the client to use hyperlinks in order to access resources
- **Layered system** - the system is organized hierarchically into components such that a component can access only the other components which are in a neighbour layer.
- **Code on demand** - an optional guideline which allows the server to pass executable code to the client after a request has been made.

4.4 Analysis of the experimented text similarity algorithms

This section presents a comparative analysis of various text similarity algorithms whose accuracy and error rate range from poor to very good. At the end of this chapter, the algorithm which offers the best results for the current problem, that of computing the semantic similarity between ideas, will be picked from one of the algorithms which will be presented in the following pages.

The algorithms' accuracy was computed using the Mean Squared Error and the Maximum Error against a test dataset containing pairs of sentences and the similarity between them computed as a number between 1 and 5. The dataset is available at [12].

An example of such a pair with a high similarity is:

- **Sentence 1** - *U.S. prosecutors have arrested more than 130 individuals and have seized more than \$17 million in a continuing crackdown on Internet fraud and abuse.*

- **Sentence 2** - *More than 130 people have been arrested and \$17 million worth of property seized in an Internet fraud sweep announced Friday by three U.S. government agencies.*
- **Similarity** - 4.600

An example of such a pair with little to no similarity is:

- **Sentence 1** - *Revenue rose 3.9 percent, to \$1.63 billion from \$1.57 billion.*
- **Sentence 2** - *The McLean, Virginia-based company said newspaper revenue increased 5 percent to \$1.46 billion.*
- **Similarity** - 1.200

The performance of the algorithms was computed using the most well-known evaluation metric in regression problems: Mean Squared Error. Mean Squared Error (MSE) is a metric which computes the distances to a regression line from a set of points then squaring all the distances, which are considered errors. As its name suggests, the result is an average of the errors previously computed [13].

Formula:

$$MSE = \frac{1}{n} \sum (actual - forecast)^2,$$

where:

- n - number of data points,
- actual - original or observed y-value (observed value),
- forecast - y-value from regression (predicted value)

Maximum Error metric has also been used as a secondary comparative point. It is computed in a similar way to the Mean Squared Error, but instead of calculating the average of the errors, it only takes the highest error in the set of distances.

4.4.1 Cosine Similarity

Cosine similarity is a lexical type of similarity. It is used to measure the cosine of the angle formed by two vectors, which are representations of the data objects from the dataset as arrays of numbers, in an inner product space, as presented in article [14].

The formula of cosine similarity is:

$$similarity = \cos(\Theta) = \frac{A \cdot B}{||A|| ||B||}$$

The output of the cosine similarity can be interpreted in the following way:

- **-1** - vectors are complete opposites, so the texts are dissimilar
- **0** - vectors are perpendicular, so the texts are somewhat similar
- **1** - vectors are practically overlapping, so the texts are completely similar

This method is used in all the approaches presented in the following sections as the final step in computing the similarity score.

4.4.2 Approach #1 - GloVe Embeddings and Cosine Similarity

The models typically used in Machine Learning algorithms use inputs consisting of arrays of numbers which are represented as vectors. In real-world scenarios in the area of NLP, algorithms should take text as input instead of numbers, because text is easily comprehended by humans. Thus, a conversion must be made from text data to number data so that the algorithms could be applied. There are a few strategies for doing so including: one-hot encoding, the encoding of each word with a unique number and using word embeddings. The focus of this paper is set on the word embeddings method.

Word embeddings are an efficient strategy for translating words in natural language to numerical values, as presented in article [15]. This dense depiction allows words similar in meaning to also have close encodings which need not be specified manually, but rather trained to discover the accurate weights. According to its definition, an embedding is a dense vector containing floating-point numbers. Word embeddings are n -dimensional embeddings, with n ranging between small values, such as 4, and 1024, depending on the size of the dataset. Embeddings of larger dimensions offer the most precise results, although the training needs a significant amount of data to learn from.

GloVe stands for Global Vectors for Word Representation and is an unsupervised natural language processing algorithm developed at Stanford University and used for generating word embeddings from large collections of text called corpora, as explained in article [16] and paper [17]. It is a log-bilinear (LBL) regression model which is based on the frequency statistics of corpora related to the global word-to-word occurrences used in the training phase. The training step was executed on five corpora of different dimensions obtained from distinct versions of Wikipedia text and Gigaword text containing billions of tokens.

The GloVe Embeddings and Cosine Similarity approach simply applies the pre-trained GloVe embeddings on the two input pieces of text to obtain the numerical representation of them, applies the cosine similarity, which was explained in the previous section, on the obtained encodings, then finally converts the value obtained from applying the cosine function to a percentage, which is easily interpreted by humans.

4.4.3 Approach #2 - NLTK, Gensim and Cosine Similarity

As the title mentions, this second approach uses NLTK and Gensim packages and the features they offer, as well as the Cosine Similarity, which was previously presented in section 4.4.1 *Cosine Similarity* of this paper. This method is introduced and detailed in article [18].

The first step in this approach is to tokenize the input sentences using NLTK, in order to get the mean number of words in a sentence. Then, the words of each sentence are tokenized individually and added to a dictionary of key-value pairs, the key being a unique number identifier and the value being the token.

In the following step, the tokenized words and their ids are passed to the Gensim version of *doc2bow*, explained in the documentation [19], through the dictionary previously created, which returns a bag of words. The bag of words is represented as the vector formed of the frequency of the apparitions of each word in the previously created dictionary. The bag of words is in essence a list of tuples of the form (*id_of_token*, *frequency_of_token*).

The next phase consists of applying the TF-IDF (Term Frequency - Inverse Document Frequency) on the bag of words generated in the previous step. TF-IDF is a method applied for determining the importance and relevance of a word in a sentence, unlike simple TF (Term Frequency) which only counts how many times the words appear in a sentence. For instance, the words *and*, *the* and *a* typically appear with a high frequency count in sentences, although they may not be very relevant in determining their meaning.

In the next step, the Gensim similarity object is created, which has the purpose of generating the index for a collection of pieces of text and then further dividing the generated index into sub-indexes, forming an index matrix. In the final step, the similarity score is computed using the cosine similarity, in a similar manner to the previous approach, then the results are converted to percentages.

4.4.4 Approach #3 - SBERT and Cosine Similarity

The third and final approach consists of using the Sentence-BERT embeddings, then applying the cosine similarity on them, similarly to the GloVe Embeddings and Cosine Similarity approach. Sentence-BERT (SBERT) is a highly efficient and performant method employed for performing the computation of word embeddings, which is built on top of pre-trained BERT networks using siamese and triplet networks and fine-tuned to obtain more accurate results.

BERT is the abbreviation for Bidirectional Encoder Representations from Transformers, as presented in article [20]. It is a pre-trained NLP framework developed by Google to determine the meaning of pieces of text using not only the general meaning of the words themselves, but also the context of the word sequence determined by the surrounding text. Taking the surrounding context into consideration changes the meaning of the words, just as in real-life. The massive text dump used to pre-train BERT was extracted from Wikipedia pages and Brown Corpus. It was pre-trained using only unlabeled

data to serve as its foundation. BERT can be further trained in order to obtain more finely-grained results depending on the topic of the text.

BERT's bidirectionality comes from its ability to read input text from left to right, as well as right to left at the same time. The Transformer aspect of BERT is the crucial part in its logic owing to the fact that it is used to gather the context of a word dynamically using all other words in the text, instead of dealing with words individually, like GloVe.

However, BERT's architecture makes it difficult to use in real-world scenarios due to the large processing time it takes, even though it offers quite accurate results, as described in paper [21]. SBERT solves the problem of long processing time by reducing it significantly, while also maintaining the same degree of accuracy offered by BERT. SBERT's architecture is illustrated in the Figure 4.10.

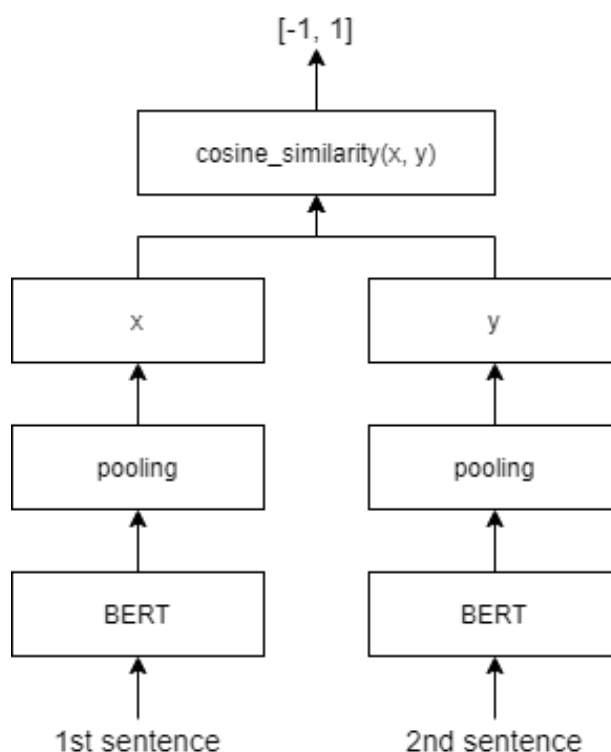


Figure 4.10: SBERT's architecture used for text similarity, as presented in [21]

As previously explained, the architecture of SBERT consists of a siamese or twin network, meaning that the pieces of text being compared are evaluated at the same time and in the same manner. In both branches of the architecture, the base layer consists of BERT, whose output is fed into the second layer which is responsible for the pooling operation, meaning that the differing size outputs of the previous layer are converted to obtain sentence embeddings of the same length. In the final step, the embeddings obtained in the preceding layer are simply compared using the cosine similarity approach followed in the two other methods.

4.4.5 Experimental results

After running each approach presented above against the Mean Squared Error and the Maximum Error, the results in the following table have been obtained:

No.	Algorithm/Measure	Mean Squared Error	Max Error
1	GloVe and Cosine similarity	555.1939954	60.87
2	Gensim and Cosine similarity	0.3861765	0.991899
3	SBERT and Cosine similarity	0.0226849	0.397588

Table 4.1: Mean Squared Error and Maximum Error for each experimented approach

Furthermore, the results obtained after performing the analysis which are present in the previous table have also been illustrated using the following bar chart to be able to visualise them more easily:

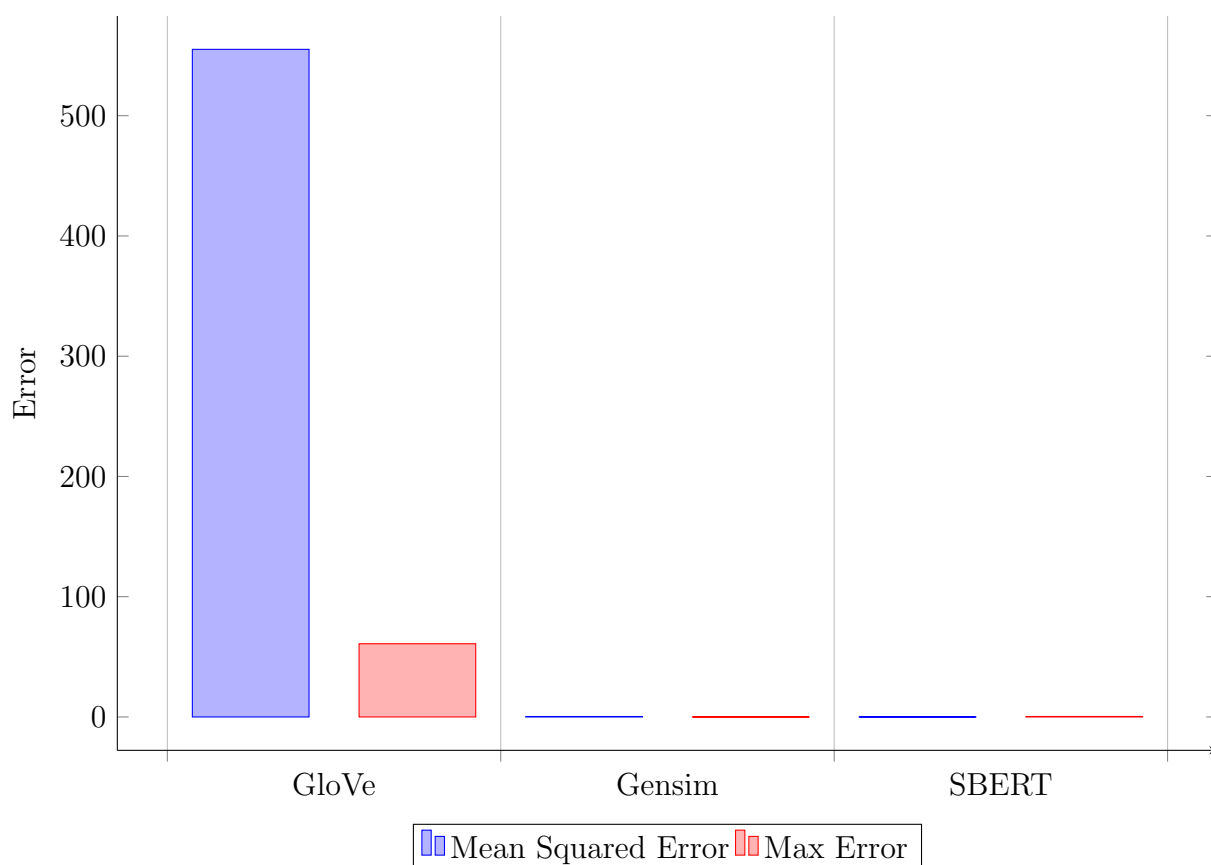


Figure 4.11: Plot of the results in the experiment

The results measured with Mean Squared Error and Maximum Error are considered to be better as they get closer to 0. Taking this into account, it is clear that the GloVe and Cosine similarity approach generates the poorest results of all three experimented methods.

Since in the bar chart above the Gensim and Cosine similarity and SBERT and Cosine similarity approaches seem to offer much better results than the GloVe and Cosine similarity method and the outcomes appear similar in this chart, an additional plot has been created in order to analyse better the differences between the results.

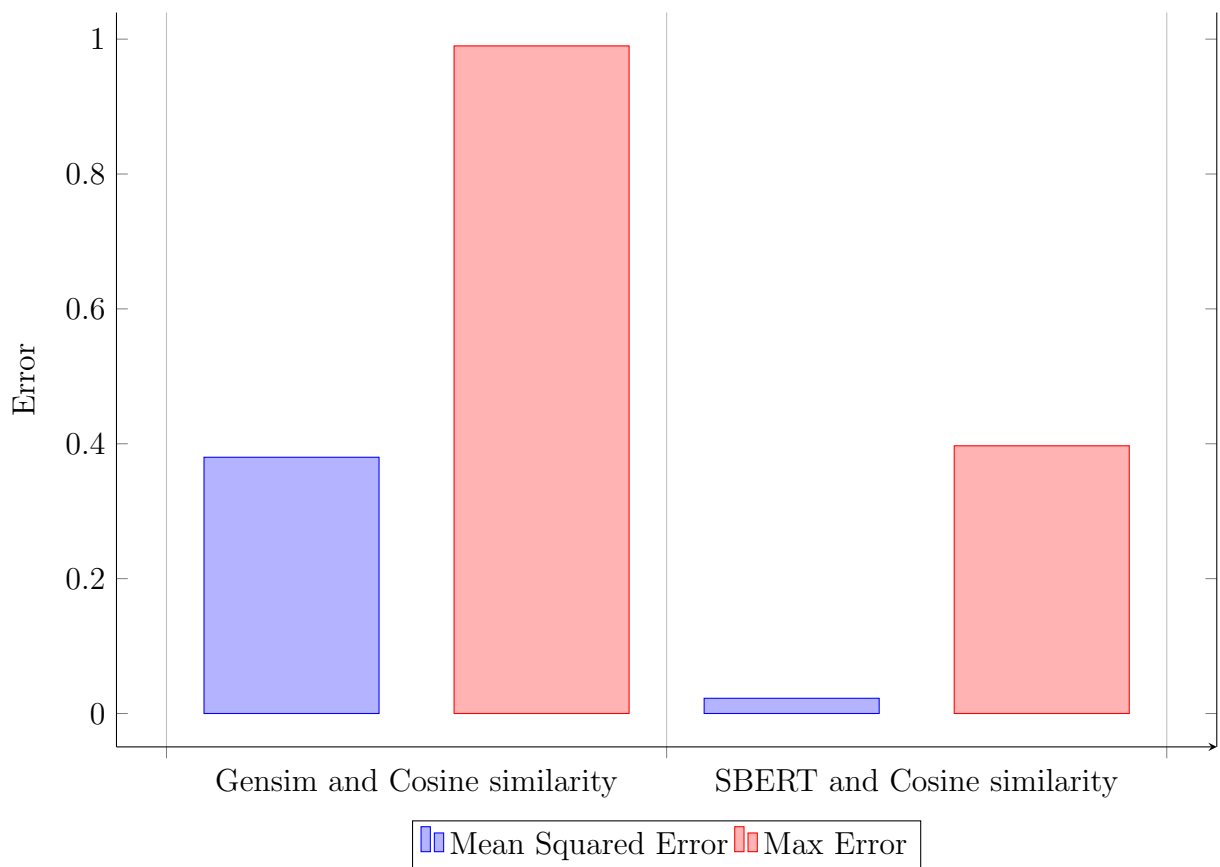


Figure 4.12: Plot of the two best results in the experiment

According to the plot above, the method which yields the best results for both Mean Squared Error is the SBERT and Cosine similarity approach, with the MSE 0.02 and the Max Error 0.39, in contrast with Gensim and Cosine similarity which had the MSE 0.38 and Max Error 0.99.

As a result of the previous comparative analysis, the obvious approach chosen to be integrated in the Continuous Improvement Process Tool is SBERT and Cosine similarity.

4.5 Description of utilised technologies

This section of the paper presents the technologies used in the development of the tool in a detailed fashion.

4.5.1 ASP.NET Core

ASP.NET Core is a free, open-source and cross-platform framework developed by Microsoft as a successor for ASP.NET framework. It uses the C# programming language and the .NET Core framework. It is used for developing web applications including Web APIs, as well as MVC multiple-page applications. It runs on Windows, Linux and Mac.

What is more, two other main frameworks have been used in the development of the .NET backend application: Entity Framework Core and Identity Framework Core. Entity Framework Core is an ORM (object-relational mapper) which is used for creating and manipulating databases with POCOs (Plain Old C# Objects), without the need to write any SQL code or scripts. Identity Framework Core is a platform used in ASP.NET Core for performing a handful of features out of the box, including managing users, authentication, authorization based on roles, e-mail confirmation and database templating.

4.5.2 Python

Python is a high-level multi-purpose programming language. It is interpreted and dynamic, which makes it easy to use and has a shallow learning curve. It is used in Web development for the design of backend applications, in Scientific and Numeric computing, in the development of GUIs for Desktop applications, in Business Applications and as a general Software Development tool.

Several Python frameworks and libraries have been used in the development of CIP Tool. They include:

- **Flask** - a micro-framework used in Python for developing small Web applications
- **PyTorch** - a Machine Learning library which offers optimised features for deep learning in the fields of computer vision and natural language processing using CPUs and GPUs
- **SentenceTransformers** - a framework developed for natural language processing which deals with text and image embeddings
- **NLTK** - a library used in natural language processing which includes features such as stemming, tokenization and classification
- **scikit-learn** - a library used in Machine Learning which deals with supervised and unsupervised algorithms

- **numpy** - a Python library used for data structures such as arrays and matrices
- **pandas** - a Python library used for the analysis of data which offers additional data structures
- **SQLAlchemy** - a Python library which offers the features of an Object Relational Mapper (ORM) for SQL databases; the lightweight equivalent of Entity Framework Core in .NET Core

4.5.3 Angular

Angular is a frontend framework used for developing Single Page Applications (SPAs) with TypeScript. TypeScript is a programming language built on top of JavaScript which offers static typing. Angular was created by Google in 2009 and it is the successor of AngularJS.

The benefits of choosing Angular over other frontend technologies include:

- Out of the box functionality
- It offers a consistent structure which is split into modules, services and components, which helps in keeping the code well-organized
- It offers two-way binding
- It offers static type checking which makes the detection of errors in the code easier and much faster
- The use of components makes the unit testing easier
- It uses the MVC architectural style implicitly
- It offers templates for the UI
- It uses structural and attribute directives
- It is easily maintainable in large code bases
- It uses a single HTML page to load the content
- It uses dependency injection

The libraries used to customize the frontend application are Material UI, Bootstrap and PrimeNG.

Chapter 5

Detailed Design and Implementation

The aim of this chapter is to present the main functions of the application, its architecture, the main components and modules, the relevant diagrams and the organization of the code.

The system is composed of four different main components, which were briefly presented in the previous chapters: SQL Server Database, backend ASP.NET Core application, Python Machine Learning application and frontend Angular application.

5.1 Architecture

The tool was designed to implement multiple architectural patterns including Client-Server, Three-Tier Architecture and MVC so as to achieve a better structure of the code which could be easily maintainable, and the clear separation of concerns which substantially improves the flexibility of the system.

As mentioned previously, the whole system is split into multiple components. In order to visualize better the physical connections between the components, the UML deployment diagram of CIP Tool was drawn to highlight the execution architecture of the application.

In the diagram below, the relationships between all main components can be clearly observed. The SQL Server database has a direct communication path with both Python Machine Learning (ML) API and ASP.NET Core Web API. However, there is no direct linkage between the SQL Server database and the frontend Angular application to prevent the misuse of the data and to avoid possible attacks from the outside. Furthermore, as the diagram below shows, Python ML Flask API and the ASP.NET Core Web API run independently and do not communicate directly since they should not know the inner workings and implementation of each other. Nevertheless, both of them have a direct connection with the frontend Angular application and they communicate via HTTP requests.

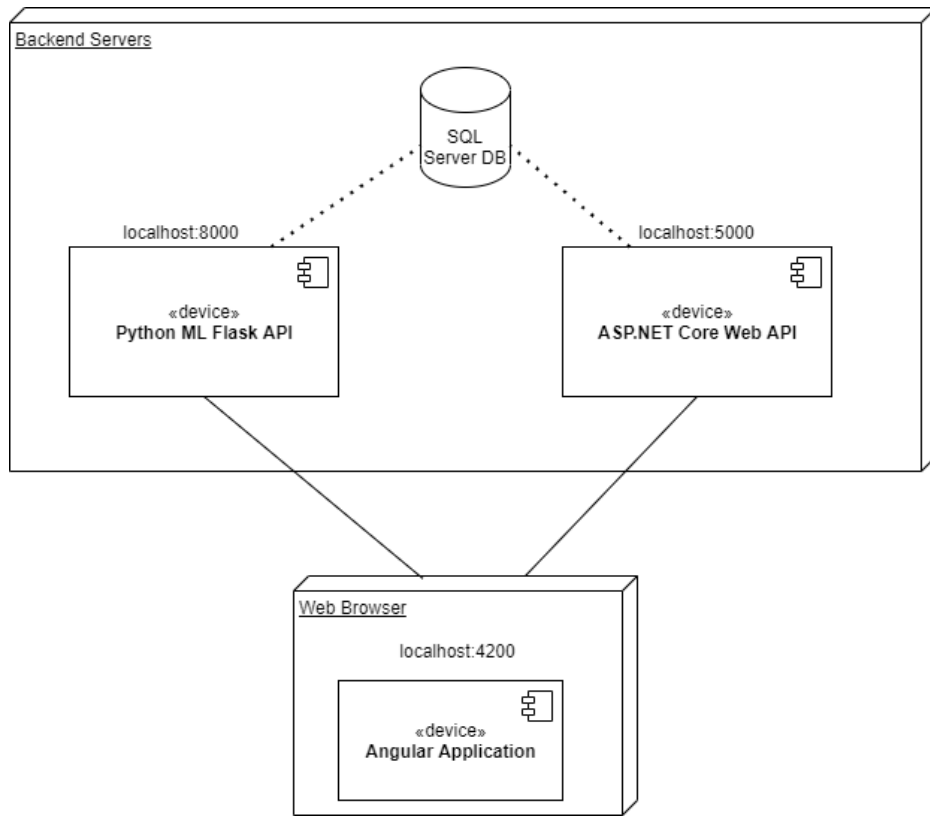


Figure 5.1: Deployment Diagram of the system

5.2 SQL Server Database

The database is a fundamental component of the system. It stores the data in the tool in a persistent manner, meaning that the data will be saved even if the system is stopped or restarted, which is crucial in CIP Tool. Microsoft SQL Server was chosen as the relational database management system because it offers the advantages of ensuring security over the stored data, organization of data in tables containing records as rows, simplicity in managing the data and performance optimization.

The database itself and its organization in tables have been created through migrations generated via C# code in the backend application using Entity Framework Core and Identity Framework Core. Each time a change is made in the model in the backend application, a migration should be created and then applied on the database through the following commands in the Package Manager Console:

```
Add-Migration <<MigrationName>>
Update-Database
```

The Entity-Relationship Diagram of the system is presented below.

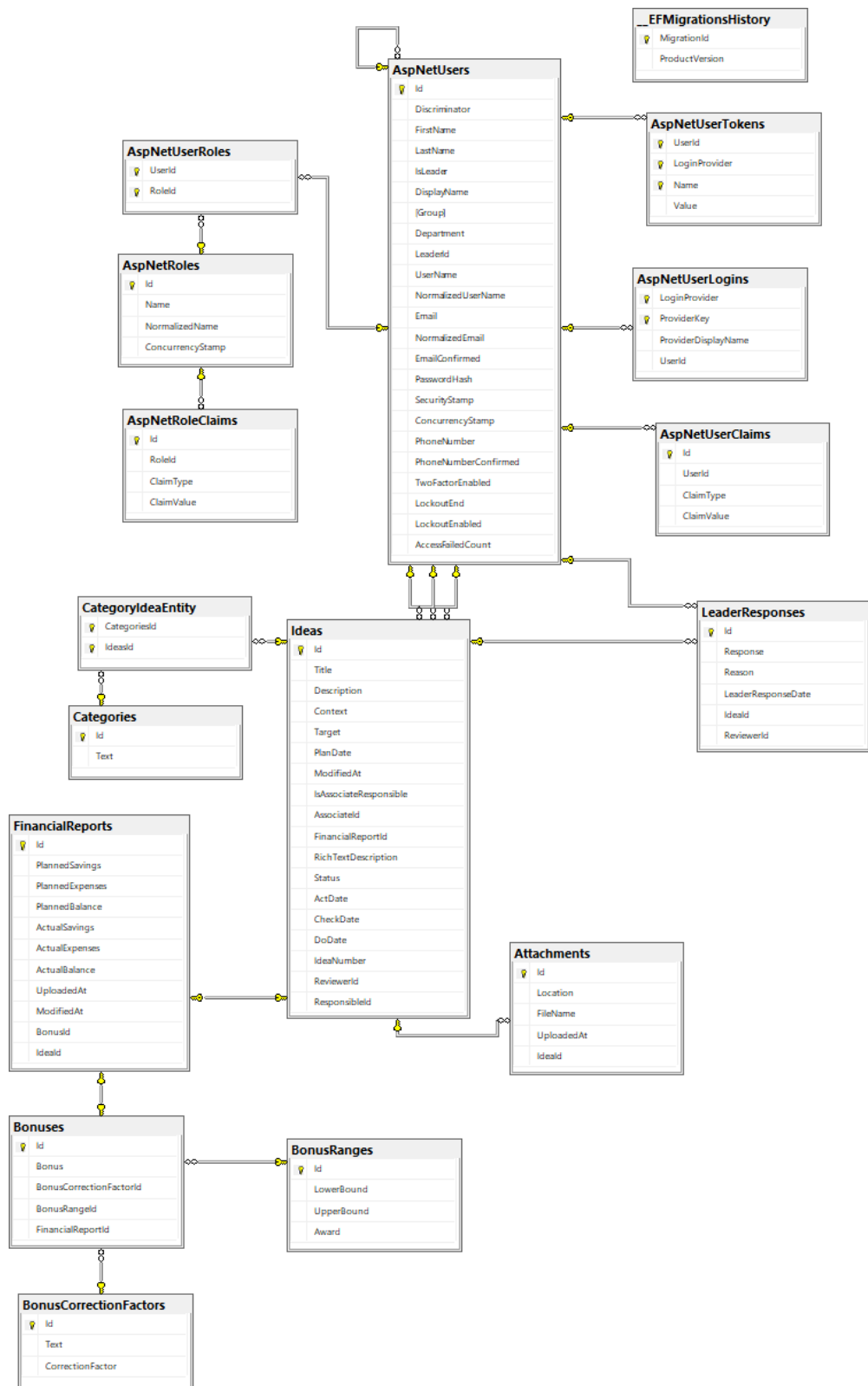


Figure 5.2: ER Diagram of CIP Tool

The diagram above is quite complex, as it incorporates additional tables generated using Identity Framework Core, apart from the tables needed to model the problem to be solved by the tool. These tables are used for the secure and complete management of the users in the tool and include *AspNetUsers*, *AspNetUserRoles*, *AspNetRoles*, *AspNetRoleClaims*, *AspNetUserTokens*, *AspNetUserLogins* and *AspNetUserClaims*. The database also contains a table named *EFMigrationsHistory* which is used to keep track of the migrations applied on the database.

The database has been normalized in the BCNF, which stands for Boyce-Codd Normal Form, by removing all non-trivial functional dependencies of attributes on anything other than a superset of a candidate key in order to avoid inconsistencies and anomalies in the insertion, update and deletion of the data in the database caused by data duplication and redundancy, according to the BCNF definition.

The database is in Boyce-Codd Normal Form if and only if:

- All tables are in 3NF.
- For every table, if a functional dependency exists, denoted as $X \rightarrow Y$, then X must be the super key of the table.

As Figure 5.1 suggests, the First Normal Form (1NF) is applied on the database because it contains no table with any composite attribute, only single attributes.

Furthermore, the Second Normal Form (2NF) is also implemented since the database is in 1NF and all tables fulfill the condition that all non-primary key attributes are fully functionally dependent on the primary key attribute of their corresponding table such that no partial dependency exists.

What is more, the Third Normal Form (3NF) is applied on the database as well since both 1NF and 2NF are also applied and no transitive dependency exists between a non-primary key and another non-primary key.

Consequently, the first condition for this database to be in Boyce-Codd Normal Form has been fulfilled, since the database is already in the Third Normal Form as proven before, as well as the second condition which states that no prime attribute can depend on a non-prime attribute, as it can be seen in Figure 5.1.

5.3 Backend application

The backend application is the core of the system. Its role is to provide access to the data stored in the database, as well as perform business logic operations on it by acting as the server and sending responses to the client containing the data it manipulates when requested via HTTP requests, which have been explained in detail in subchapter 4.3.3, in a RESTful manner, concept which has been presented in subchapter 4.3.4 in the fourth chapter of this paper containing the Analysis and Theoretical Foundation of this tool.

As mentioned in the previous chapter, the technology used in the development of the backend application is ASP.NET Core Web API 3.1 and the C# programming language. This technology was chosen due to its high performance, flexibility and its effortlessness in manipulating the SQL Server database which interacts with this kind of application very well, as well as its simplicity in exposing secured endpoints to other applications, namely the frontend Angular application in this case.

The backend application was designed in such a way as to implement the best practices in programming.

Firstly, the solution has been divided into four different projects: three Class Library projects named *DataAccessLayer*, *BusinessObjectLayer* and *BusinessLogicLayer*, and a Web API project named *CIPTool*. The connections between the projects in the solution can be visualized in the following figure.

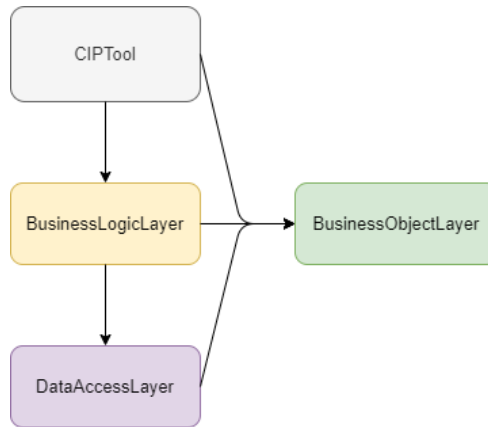


Figure 5.3: Project structure of the backend application

According to the structure of the project illustrated in Figure 5.3 which resembles the Three Tier Architecture, the *BusinessObjectLayer* which defines the entity classes and the Data Transfer Object (DTO) classes is accessed by all the other projects in the solution. Moreover, the *CIPTool* Web API project represents the Presentation Layer of the backend application, thus it only accesses *BusinessLogicLayer* apart from *BusinessObjectLayer*. The *BusinessLogicLayer*, which contains the business logic and the main functions of the application, is only accessed by *CIPTool*, but it accesses directly the *DataAccessLayer* and *BusinessObjectLayer*. The *DataAccessLayer* project, which is responsible with the access and mapping of the data from the database, is only accessed by *BusinessLogicLayer* and communicates directly with the *BusinessObjectLayer*, just as the other projects in the backend application.

Secondly, the practices of DRY (Don't Repeat Yourself), YAGNI (You Aren't Gonna Need It), KISS (Keep It Stupidly Simple), appropriate naming conventions, consistent indentation, avoidance of deep nesting and proper file and folder structure have been employed in the backend application.

What is more, the SOLID principles, encompassing Single Responsibility Principle (S), Open-Closed Principle (O), Liskov Substitution Principle (L), Interface Segregation Principle (I) and Dependency Inversion Principle (D), have been carefully implemented into the application in an effort to develop clean code which brings long-term benefits for the project. The Dependency Inversion Principle was given the utmost importance since ASP.NET Core supports dependency injection by default through built-in containers represented by *IServiceProvider* interface called services.

As mentioned in the previous chapter, Entity Framework Core was used to map the plain C# objects to the tables in database to make the manipulation of data easier for the .NET programmer, as well as Identity Framework Core, which was used to provide templates for the user related tables in the database and out-of-the box functionality for the authentication and role-based authorization.

In order to increase testability using dependency injection principle, improve flexibility in case the database provider needs to be changed or the way the data is manipulated, and reduce duplicate code, the Repository Pattern was implemented inside *DataAccess-Layer*. It uses repositories as mediators between the models and Entity Framework Core to manipulate the data in the database in an object-oriented manner, while also providing the separation between the entities and the persistence layer. This pattern was implemented by creating a generic interface *IRepository* containing the declarations of CRUD methods, which is implemented by interfaces created for each repository, including *IAttachmentRepository*, *IUserRepository* and *IdeaRepository*, as illustrated in the partial class diagram below representing the abstract part of the pattern.

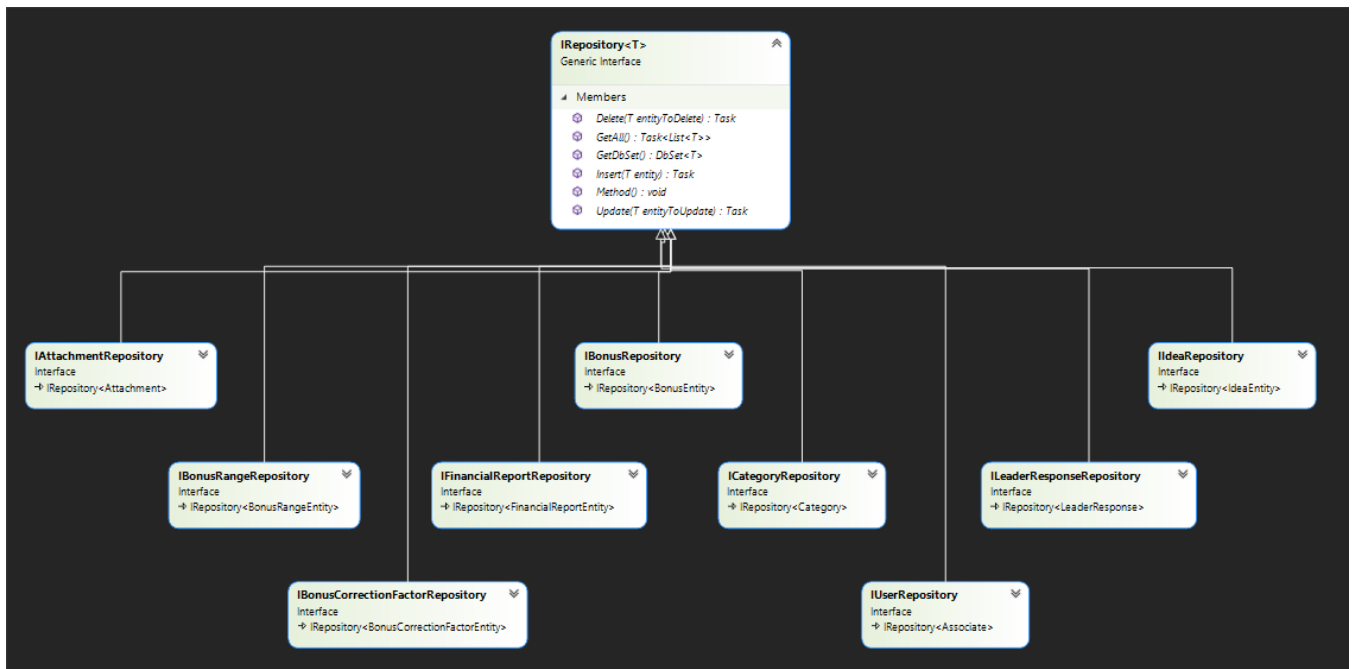


Figure 5.4: Diagram of the abstract part of the Repository Pattern

The concrete part of the repository pattern is presented in the class diagram below. The generic abstract class *BaseRepository* contains the concrete generic implementation of the methods in the *IRepository* interface, as it implements the interface. Furthermore, concrete classes have been created for each repository, including *AttachmentRepository*, *UserRepository* and *IdeaRepository*. Each of these classes implement their respective interface, e.g. *LeaderResponseRepository* implements *ILeaderResponseRepository*. They also extend the *BaseRepository* class, e.g. *LeaderResponseRepository* extends *BaseRepository<LeaderResponse>*.

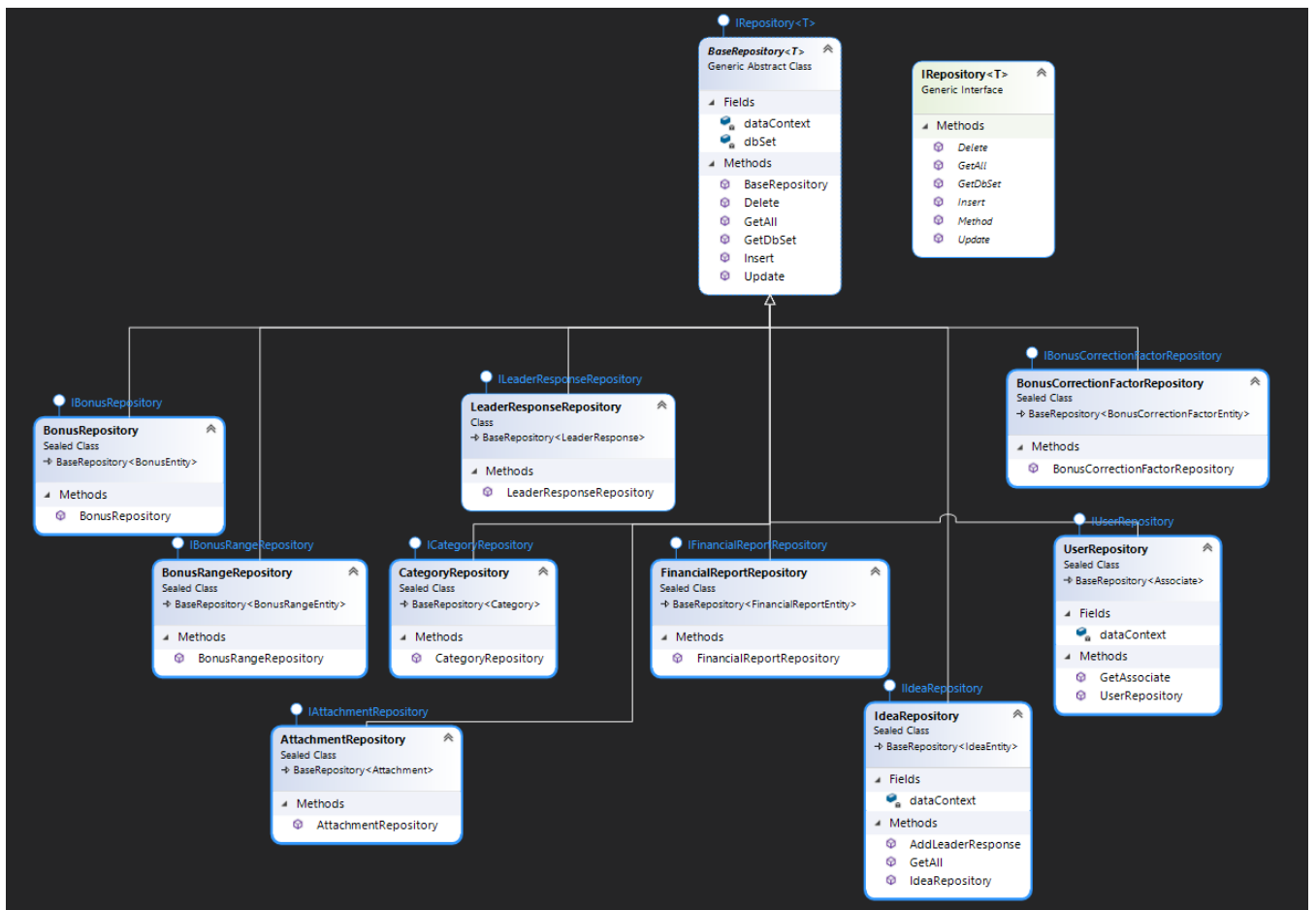


Figure 5.5: Diagram of the concrete part of the Repository Pattern

Without the employment of this design pattern, the number of lines of code needed to be written for the mapping of the data layer into plain C# objects would have been almost 9 times larger. Thus, it does not only provide a clean separation of concerns, but it also reduces large amounts of duplicate code and saves time during the implementation and the testing.

The injection of concrete dependencies is simply performed in *CIPTool* project in the *Startup* class using the out-of-the-box inversion of control feature that ASP.NET Core provides, like in the following example:

```
services.AddScoped<ILeaderResponseRepository, LeaderResponseRepository>();
```

The dependency injection principle was also implemented in the *BusinessLogicLayer* for the creation of services. For each service, an abstraction has been created in the form of an interface which is implemented by a concretion in the form of a class. Moreover, the repositories needed for the business logic of each service have been injected in the service classes as well in order to take advantage of the benefits offered by the Inversion of Control SOLID principle. *BusinessLogicLayer* contains 4 distinct services: *FinancialReportService*, *StatisticsService*, *IdeaService* and *UserService*.

IdeaService is concerned with various CRUD operations on the ideas, attachments, categories and leader response and provides the necessary implementation of those operations. It uses *AttachmentRepository*, *CategoryRepository*, *IdeaRepository* and *LeaderResponseRepository* from *DataAccessLayer* project via dependency injection. The figure below represents the class diagram of the idea service component.

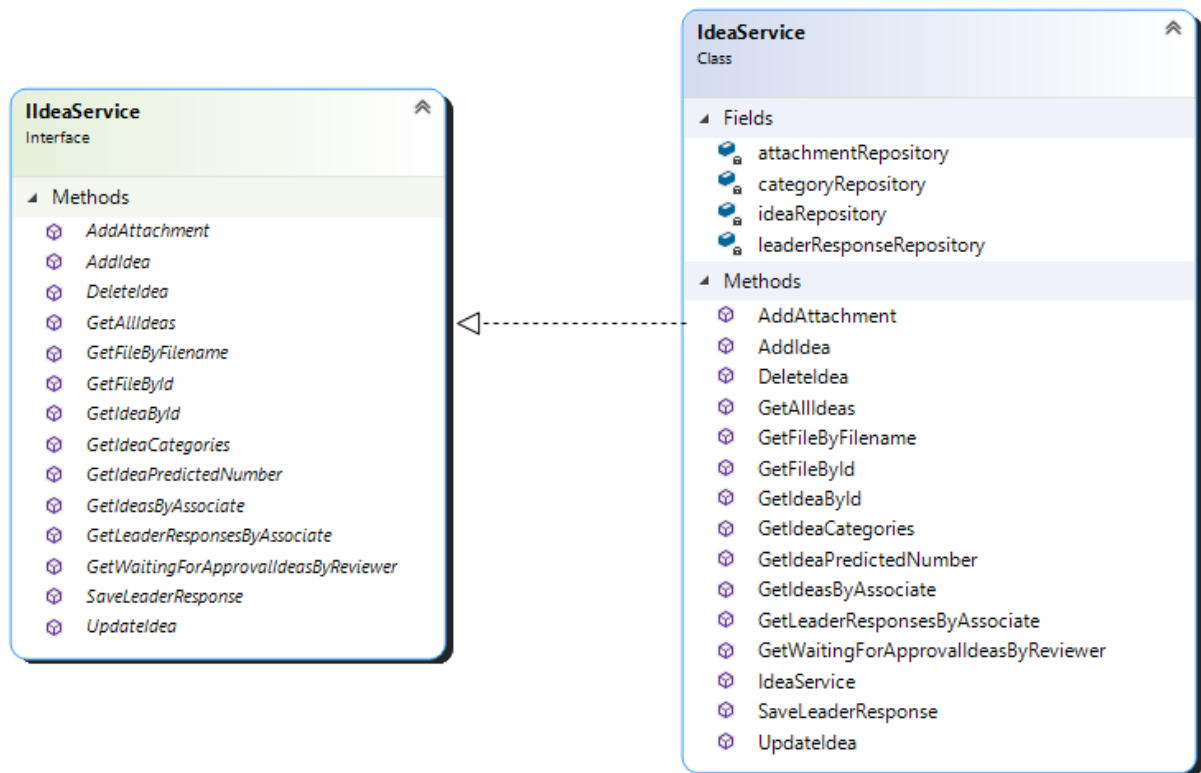


Figure 5.6: Class diagram of the idea service component

FinancialReportService is responsible for the CRUD operations on the financial reports, as well as the computation of bonuses based on the financial reports, and provides the necessary implementation of those operations. It uses *BonusCorrectionFactorRepository*, *BonusRepository* and *FinancialReportRepository* from *DataAccessLayer* project via dependency injection. The figure below represents the class diagram of the financial report service component.

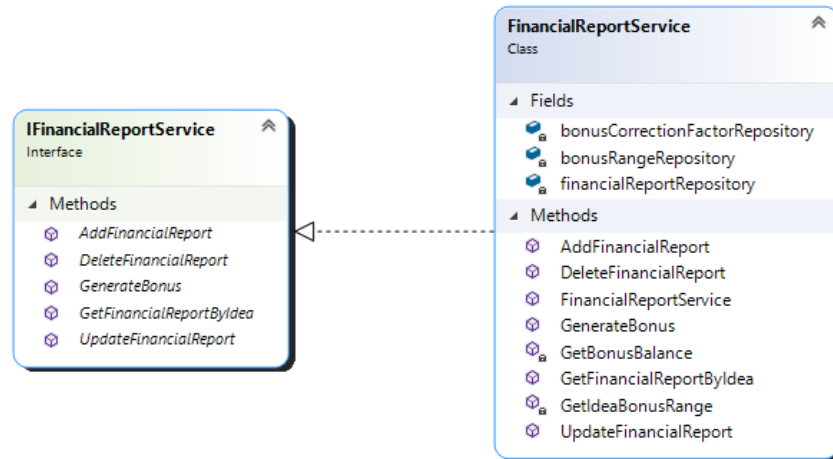


Figure 5.7: Class diagram of the financial report service component

UserRepository is concerned with various CRUD operations on the users and provides the necessary implementation of those operations. It uses *UserRepository* from *DataAccessLayer* project via dependency injection. The figure below represents the class diagram of the idea service component.

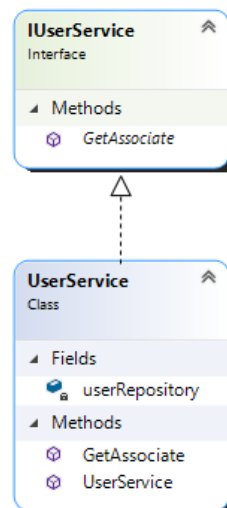


Figure 5.8: Class diagram of the user service component

StatisticsService is employed with the implementation of all necessary operations for computing statistics based on ideas registered in the tool. Such statistics are quite varied and include computing the percentage of implemented ideas from all ideas, the OTD Decision value, which represents the difference in days between the leader response date submission date and the of an idea, and OTD Implementation value, which represents the difference in days between the actual implementation date and the planned implementation date of idea. It uses *IdeaService* from the same project, *BusinessLogicLayer* project, via dependency injection, with no need to access any repository from *DataAccessLayer*, since *IdeaService* already implements the methods it needs for accessing the necessary data. The figure below represents the class diagram of the statistics service component.

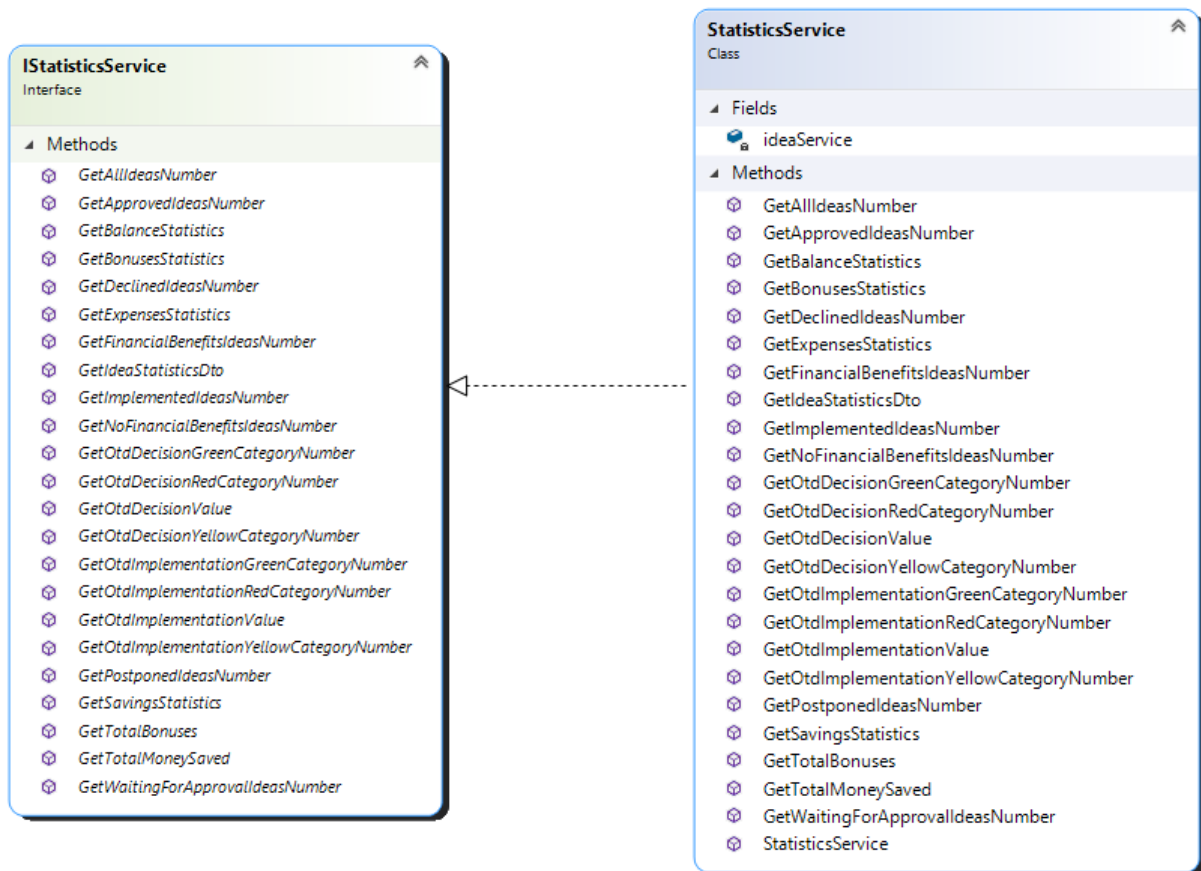


Figure 5.9: Class diagram of the statistics service component

All of the services presented in this subchapter are further injected in the controllers in the *CIPTool* project through the interfaces they implement, also highlighting the importance of Dependency Inversion principle and the flexibility it offers. In essence, these services act as the meeting ground between the data in the repositories and the actions in the controllers, where the business logic and operations are applied on the data.

There are three controllers which expose endpoints through actions they define: *AuthenticationController*, *IdeasController* and *StatisticsController*. All controllers are secured using the [Authorize] attribute which requires all methods in the controller, except methods with the [AllowAnonymous] attribute, to be performed only if the user is authenticated into the tool. The only methods having the [AllowAnonymous] attribute are the *LoginByEmail* and *LoginByUsername* actions in the *AuthenticationController*, which obviously must not be secured in order to allow authentication in the first place. Furthermore, some specific methods in these controllers are also secured with a role-based authorization filter using the roles existent in the tool *Associate*, *Leader* and *Admin*, like in the example: [Authorize(Roles=“Leader, Admin“)]. All such role-based authorized methods are defined in the *IdeasController* and include *GetLeaderResponseOverviewByAssociate* and *GiveLeaderResponse*. The authentication and authorization are performed using JSON Web Tokens (JWT), which is a tool used for transmitting data between services using JSON objects, containing the username, role, display name, first name and last name of the current user.

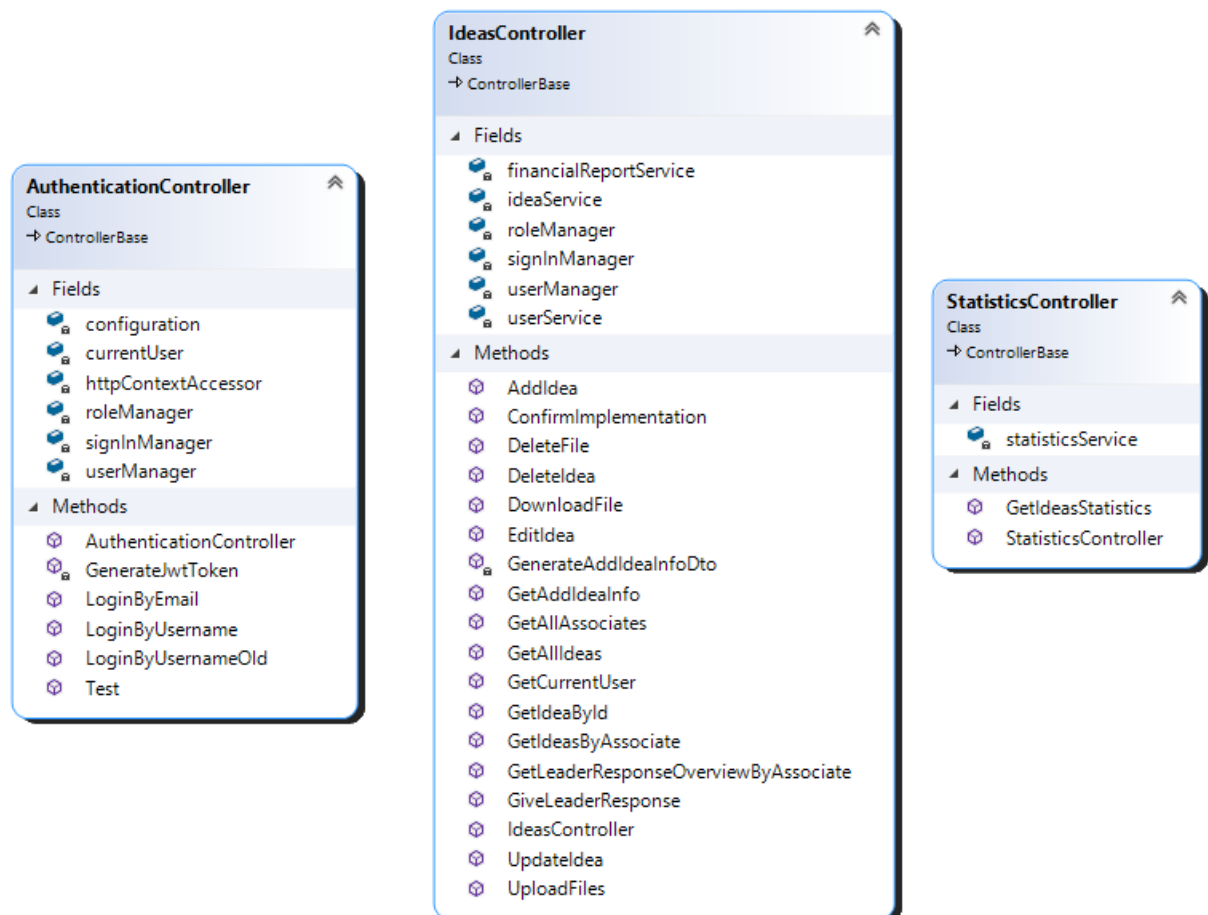


Figure 5.10: Controllers in CIPTool project

5.4 Machine Learning application

The Machine Learning application which performs the text similarity computation on the current idea which is being submitted against ideas already registered in the tool. The application was created using the Python programming language and several Python libraries and frameworks including PyTorch and Flask. It consists of an API which exposes a single endpoint that is being called by the frontend application whenever a new idea is being submitted. The endpoint will be presented using the following table:

Method Type	URL	Description
POST	/similarity	Takes the current idea fields (title, context, target, description, categories) as message body and responds with the list of ideas in the tool with the similarity score $> 40\%$

Table 5.1: Endpoint description table in the Machine Learning application

Python was chosen as the programming language for this application thanks to the variety of resources it offers in the area of Machine Learning, especially in the Natural Language Processing. Moreover, since this API exposes only an endpoint, Flask was chosen as the web framework because it is simple and easy to use. However, with Flask being such a minimal framework, it does not implement the data abstraction layer and does not include an Object Relational Mapper (ORM) by default, unlike more complex frameworks such as Django which has this functionality built into the framework. The ORM is needed for accessing the database without writing SQL scripts. For this reason, SQLAlchemy was used in the project as the ORM, being the recommendation for Flask applications.

This application accesses the database previously created by the backend ASP.NET Core application and does not modify the data or the database structure, it only retrieves it. Only three entities have been used for the access of data related to the similarity computation: Idea, Category and CategoryIdeaEntity. They are presented in the diagram below.

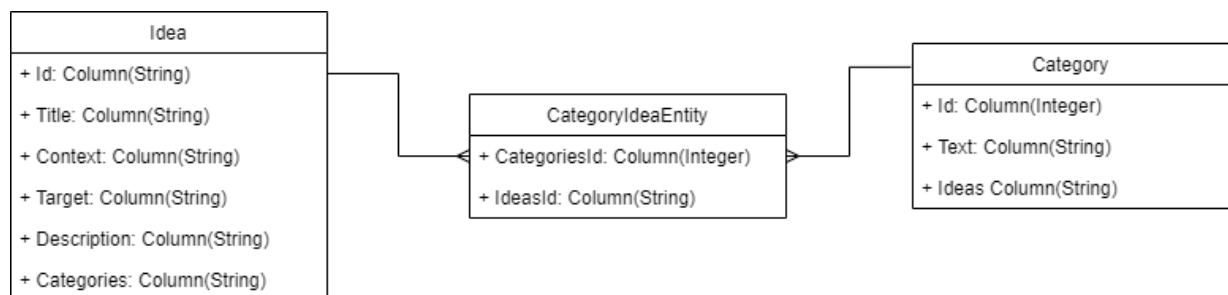


Figure 5.11: The entities used for accessing data about ideas from the database

The fields of an idea which are needed for being able to compute the similarity are: title, current context, target state, description and categories. The title, context, target and description fields are all stored in the Ideas table in the database. The categories of each idea are stored using a many-to-many relationship between the Idea and Category entities through the CategoryIdeaEntity. The ID, title, context, target and description of ideas are retrieved using a simple query on the Ideas table, while the categories are retrieved using a join query written in Python though SQLAlchemy on the Category and CategoryIdeaEntity entities based on the IDs of the ideas.

Each of these fields of an idea are given a weight in the final similarity percentage, as illustrated in the following pie chart.

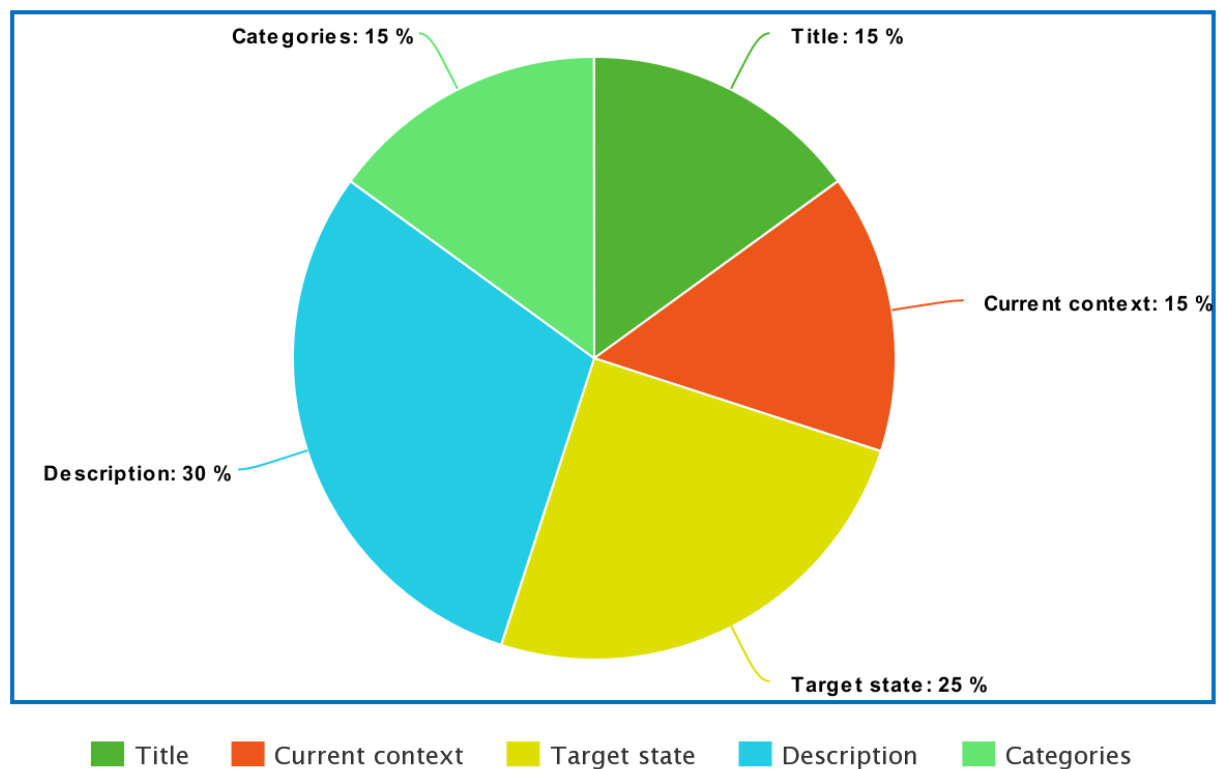


Figure 5.12: The weight of each idea field in the final similarity percentage

Despite the fact that the fields of ideas considered in the final similarity score are given different weights, all similarity percentages corresponding to each field are computed individually and in the same manner.

For the computation of the similarity percentage between the same fields of two different ideas, the SentenceTransformers Python library was used, containing the 'paraphrase-distilroberta-base-v1' pre-trained model. Firstly, the sentence embeddings are computed for each field of the current idea and for the fields of already submitted ideas using the model previously mentioned. A list of tensors is returned after computing the embeddings,

a tensor being a multi-dimensional array consisting of elements of a single data type implemented in PyTorch. Then, the cosine similarity between the sentence embeddings of the fields of the current idea and the sentence embeddings of the already registered ideas. Due to the fact that the cosine similarity method is applied, the similarity is initially expressed as a number between -1 and 1, which is further converted to a percentage value between 0 and 100. Finally, the similarity between ideas is computed as the weighted average of the previously computed scores using the following formula:

$$final_similarity = \frac{0.15 \times title_similarity}{5} + \frac{0.15 \times context_similarity}{5} + \frac{0.25 \times target_similarity}{5} + \frac{0.3 \times description_similarity}{5} + \frac{0.15 \times categories_similarity}{5}$$

In the end, a list containing the most similar ideas containing the id, title, context, target, description and similarity percentage against the current idea sorted in a descending order by the similarity score is returned and further displayed in a user friendly manner on the frontend application.

5.5 Frontend application

The frontend application was developed using Angular 9 framework, TypeScript programming language and PrimeNG, Angular Material and Bootstrap component and styling libraries. Angular framework uses the MVC pattern and works especially well in keeping the code structure clean by using components. Components are typically smaller packages consisting of multiple files corresponding to a page or an item on the UI, e.g. a custom dropdown or the navigation bar. A component can consist of a TypeScript file, an HTML file, a CSS file, a SCSS file and a TypeScript file used for testing the component. By using components, the application becomes more maintainable, dynamic, scalable and easier to work with in team, while at the same time providing the clean and powerful separation of concerns.

In a similar manner to the backend ASP.NET Core application, dependency injection is also used in the implementation of the Angular application through directives to benefit from the advantages of Single Page Applications (SPAs).

The Web application has direct links with the backend application and the Machine Learning application through the endpoints they expose which are further called by the frontend application, with no direct access to the database. It represents the presentation layer of the whole system and is the only application which the user can directly access and interact with.

The Web application is organized into multiple modules consisting of models, services, guards, components and assets, as illustrated in the package diagram below.

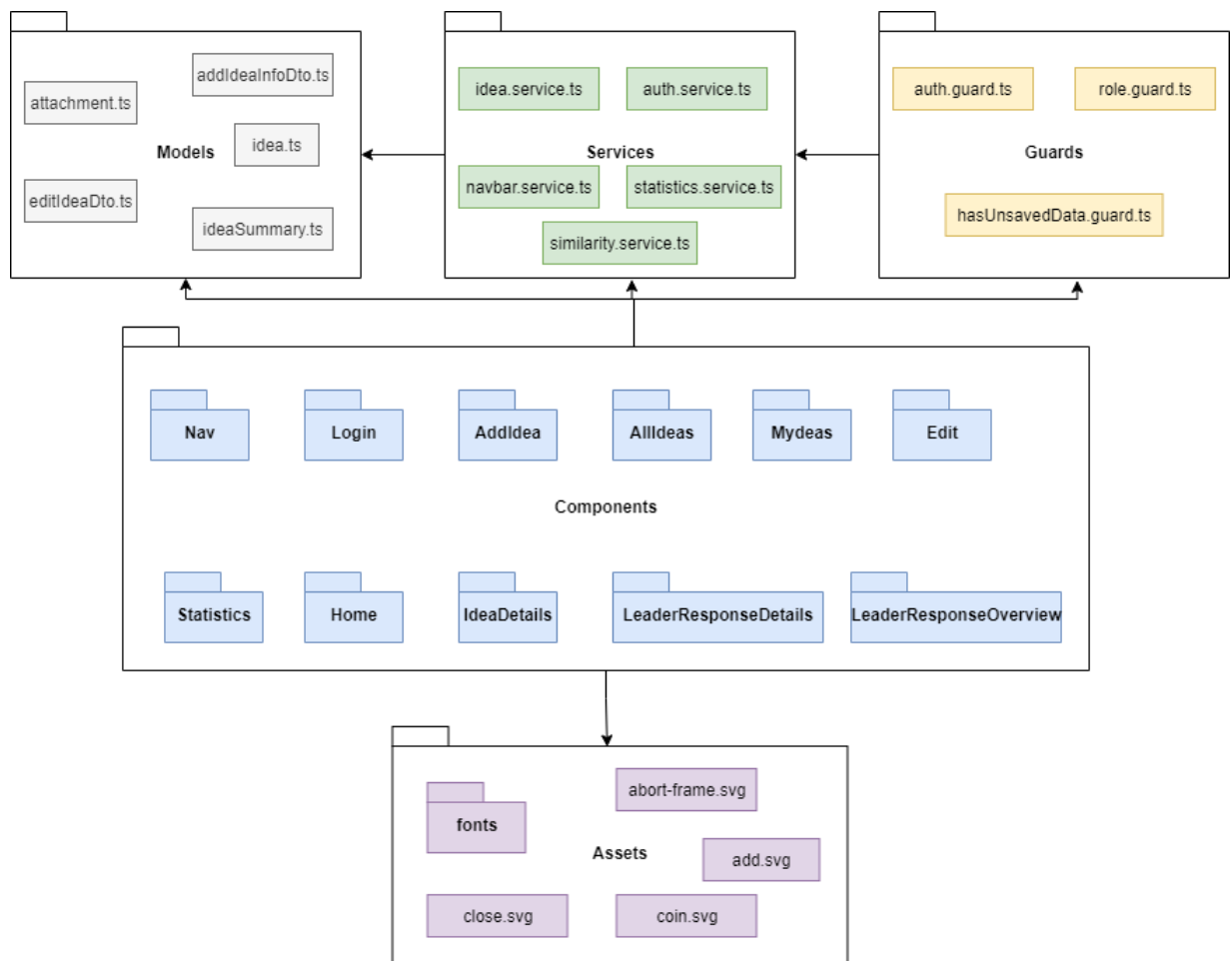


Figure 5.13: Package diagram of the frontend Web application

Each package and its main functions will be presented below:

- **Models** - contains the interfaces which define the data structures used by the Angular application, which are a particularly useful feature offered by TypeScript, as static typing improves bug spotting, readability and predictability provided by its Object Oriented Programming style. It includes the *FinancialStatisticsDto*, *Idea*, *IdeaSummary* and *AttachmentDetails* interfaces, as well as many others. This package is used by the Services and Components packages, as illustrated in Figure 5.13.
- **Services** - contains various service classes whose purpose is to implement logic which could be reused in multiple components via dependency injection. The services use the Models package and are used by the Guards and Components packages. They include:
 - *IdeaService*: contains the methods necessary for calling the idea related endpoints from the ASP.NET Core Web API

- *SimilarityService*: contains the method needed for calling the endpoint exposed by the Machine Learning API
 - *StatisticsService*: contains the method needed for calling the statistics related endpoints from the ASP.NET Core Web API
 - *AuthService*: contains the method needed for calling the authentication related endpoints from the ASP.NET Core Web API, as well as methods used for handling the JWT token and the information it contains about the logged in user
 - *NavbarService*: a service used for handling the cases when the navigation bar should be displayed, as well as the cases when it should not be displayed (e.g. in the login page)
 - *JwtInterceptor*: a service used for setting the bearer token in the authorization header for all requests by default in order to allow the verification of user credentials in the backend application
- **Guards** - contains three guards, which are interfaces that specify if a user is allowed to access a route or not
 - *AuthGuard*: a guard implementing the CanActivate guard, which is applied on routes that should be accessed only by authenticated users
 - *RoleGuard*: a guard implementing the CanActivate guard, which is applied on routes that should be accessed only by users in a certain role (e.g. the leader response pages should only be accessed by users in the Leader or Admin roles)
 - *HasUnsavedDataGuard*: a guard implementing the CanDeactivate guard, which is applied on routes corresponding to pages where the user provides input and is used for preventing the user from leaving a page containing their modifications
 - **Components** - contains the components which are behind the pages displayed to the user. It uses the Models, Services, Guards and Assets packages. These components include:
 - *NavComponent*: contains the structure, styling and functionality of the navigation bar which is displayed in all pages, except the login page
 - *LoginComponent*: represents the structure, styling and functionality behind the login page; it uses the AuthService through dependency injection
 - *HomeComponent*: represents the way the home page is displayed and how its functionality is handled
 - *AddIdeaComponent*: represents the structure, styling and functionality behind the add idea page; it uses the IdeaService and SimilarityService through dependency injection; also implements form validation in order not to allow the user to introduce empty or erroneous data

- *AllIdeasComponent*: is the representation of the all ideas page; it implements the way it displays and styles the data it receives through the IdeaService; it displays the overview of all ideas in the tool in the form of five tables, each containing ideas based on their status: Waiting for implementation, Approved, Postponed, Declined and Implemented
 - *MyIdeasComponent*: is the representation of the my ideas page; it implements the way it displays and styles the data it receives through the IdeaService and works in a similar manner to the AllIdeasComponent, except that it handles only the ideas of the currently logged in user
 - *IdeaDetailsComponent*: is the template used for displaying and handling the details of any idea in the tool; it uses the IdeaService via dependency injection as well
 - *EditIdeaComponent*: is the component used for handling the idea update feature; it also uses IdeaService and has a similar look and feel to IdeaDetailsComponent
 - *LeaderResponseOverviewComponent*: is the component used for displaying the ideas which have the current logged in leader as reviewer; it is similar to both AllIdeasComponent and MyIdeasComponent, except for the fact that it splits the ideas into two categories: Waiting for approval ideas and My responses; it also injects IdeaService
 - *LeaderResponseDetailsComponent*: is the component used for allowing leaders to give responses to ideas; it uses the IdeaService via dependency injection as well and the way it is displayed is similar to the idea details page
 - *StatisticsComponent*: is the component responsible for rendering the statistics it obtains using the StatisticsService and for allowing the export to PDF of the statistics page
- **Assets** - contains all the external resources needed in the rendering of the Web frontend application, such as various types of fonts, as well as various images used to improve the user friendliness of the website

The routing of the application is also an important aspect which the system handles. It uses paths defined by the programmer which are linked to components and have one or multiple guards attached to them. Furthermore, by being a part of a distributed system, the frontend application provides a route which will be directly accessed by the end-user. However, the great benefit comes from the fact that end-users do not access the other two applications in the system directly, which helps in preventing attacks from malicious users.

The site map of the frontend application is illustrated below. It represents the visual list of pages provided by the frontend application and is used to display the way they are organized.

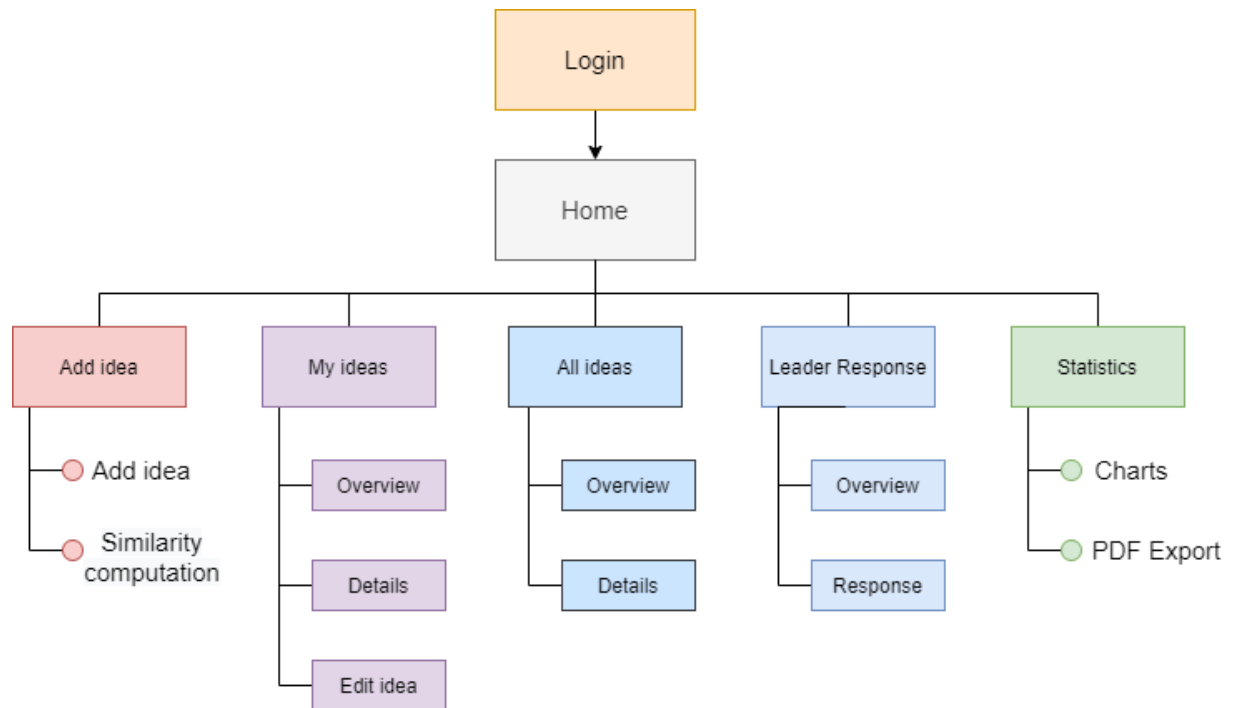


Figure 5.14: Site Map diagram of the frontend Web application

Chapter 6

Testing and Validation

This chapter has the purpose to present the methods used for testing and validating the functionality of the system in order to verify if the features offered by the tool perform as expected. The testing step of any application should never be missed, especially when dealing with a system destined to be used in the corporate world where reliability comes first and nothing less is expected.

The testing techniques used for the validation of the tool consist of the functional testing and the evaluation of the text similarity algorithms used for computing the semantic similarity between ideas.

6.1 Functional testing

Functional testing refers to the validation of the functional requirements based on the use cases presented in section 4.2 Use case modelling of Chapter 4 Analysis and Theoretical Foundation of this paper. This type of testing is a black-box testing, which means that the tester should not know or take into consideration how the code behind the application was written.

The focus on the functional testing was set on the Add Idea use case, where a user adds details about an idea, such as title, description and categories, submits the idea, receives the text similarity report and then confirms the registration of the idea.

The first step consists of selecting the Add Idea page from the navigation bar presented in the figure below.

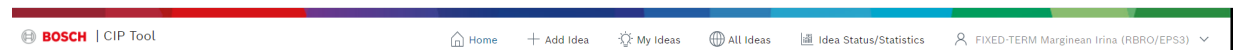


Figure 6.1: Navigation Bar

In the following step, pre-calculated idea information is loaded into the Add Idea page. It includes the idea owner name, group leader name and their group and the department leader and their department, as well as the predicted idea number.

The third step represents the filling in of the Idea title, Current context, Target state and Description text fields, as presented in the figure below along with the second step.

Figure 6.2: Pre-filled idea information and idea summary fields

The next step represents the selection of the categories the idea belongs to, as shown in the figure below.

Figure 6.3: Selected idea categories

Furthermore, the planned implementation date of an idea is selected by the user, as shown in the figure below.

A form field for the planned implementation date. It has a label "Planned implementation date *" and a text input containing "26/06/2021". To the right of the text is a calendar icon.

Figure 6.4: Planned implementation date of the idea

In the following step, the attachments of the idea are selected from the File Explorer and then uploaded to the tool.

The following figure shows the Idea Description & Attachments section of the Add Idea page. The Idea Description rich text editor is shown as filled by the user in the third step. The Attachments list is displayed as empty, since at this point no attachment was uploaded.

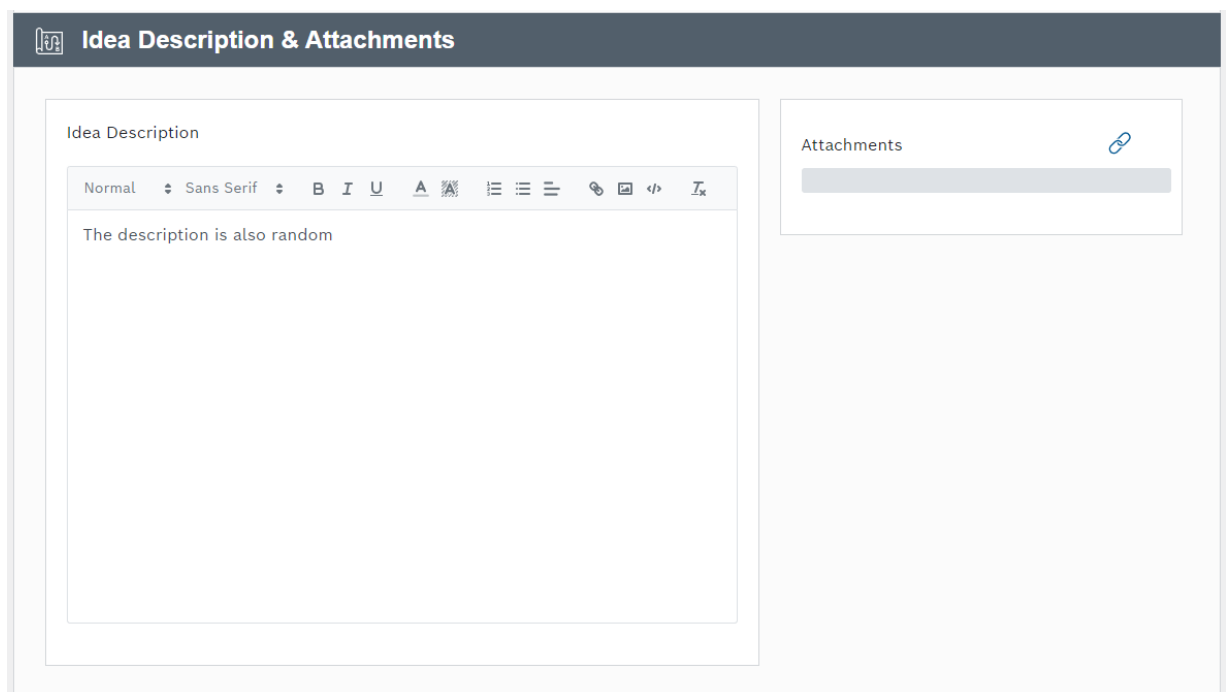
A screenshot of the "Idea Description & Attachments" section of the Add Idea page. The section has a dark header with the title and a small icon. Below the header, there are two main areas. The left area is titled "Idea Description" and contains a rich text editor with a toolbar (Normal, Sans Serif, Bold, Italic, Underline, Text Color, Background Color, Bulleted List, Numbered List, Indent, Link, Image, Code, Undo, Redo) and a text area containing the text "The description is also random". The right area is titled "Attachments" and contains a link icon and an empty list box.

Figure 6.5: Idea Description & Attachments section of the Add Idea page

In Figure 6.6, the File Explorer window is shown as displayed in the tool after pressing the icon button used for adding attachments. The *statistics (1).pdf* file was selected.

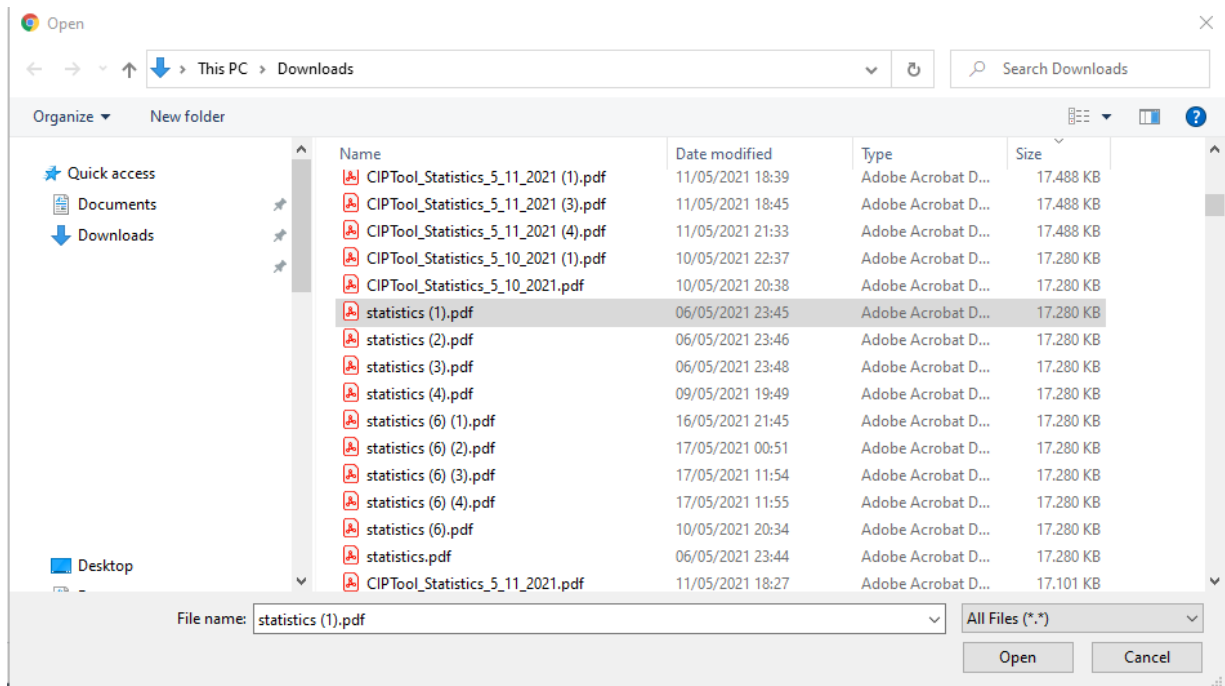


Figure 6.6: File Explorer window displaying the file to be uploaded

Figure 6.7 illustrates the updated list of attachments after uploading the previous file, as well as the completion percentage of the upload.

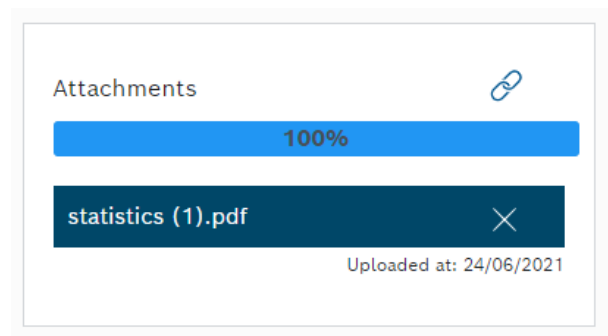


Figure 6.7: The updated attachment list after uploading a file

The next step consists of completing the financial report related to the idea. The Planned Savings and Planned Expenses fields are filled in by the user, then the system computes the Planned Bonus as their difference. The Actual Savings and Actual Expenses fields are filled in by the user, then the system computes the Actual Bonus as their difference and highlights the row in the Bonus table corresponding to the Actual Bonus value so that the bonus computation can be validated.

The figure below shows the Financial Report & Bonus section of the Add Idea page corresponding to the current step.

€

Financial Report & Bonus

Financial Report

Planned Savings

230

€

Planned Expenses

0

€

Planned Balance

230€

Planned Bonus

0€

Actual Savings

4000

€

Actual Expenses

33

€

Actual Balance

3967€

Actual Bonus

210€

Bonus

Lower Bound	Upper Bound	Award
-999999999€	499€	0€
500€	999€	50€
1000€	1999€	90€
2000€	2999€	160€
3000€	5999€	210€
6000€	9999€	300€
10000€	19999€	500€
20000€	29999€	900€
30000€	39999€	1100€
40000€	49999€	1200€
50000€	999999999€	1500€

Figure 6.8: Financial Report & Bonus section of the Add Idea with the completed fields

In the following step, the idea is submitted by pressing the Submit button, as show in Figure 6.9.

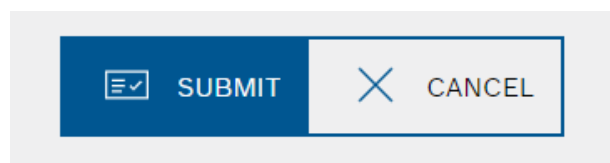


Figure 6.9: Submit and Cancel buttons in the Add Idea page

In the next step, the similarity computation results are displayed to the user in the form of a dialog which contains the titles, similarity percentages and links to the most similar ideas, which are at least 40% similar to the current idea, as shown in Figure 6.10.

Then, in the step after, the user confirms the submission of the idea by pressing the Confirm button, as shown in Figure 6.10 as well.

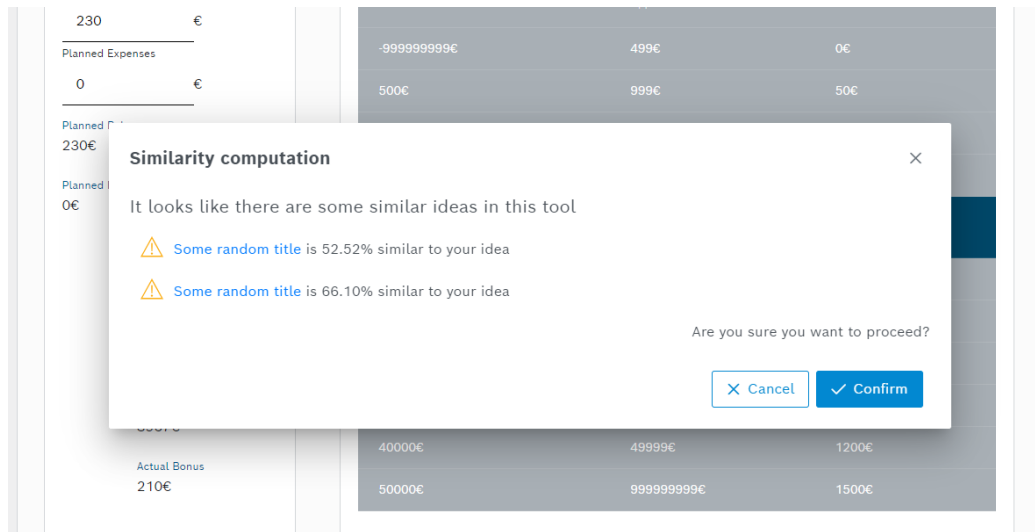


Figure 6.10: Idea similarity computation results display

In the following step, the system sends a confirmation email to the idea owner, as presented in the figure below.

Idea submission in CIP Tool



Figure 6.11: The confirmation email sent to the idea owner after registering the idea

In the end, the system redirects the user to My Ideas page, where the previously registered idea is displayed in the Waiting for approval ideas table, as shown in Figure 6.12.

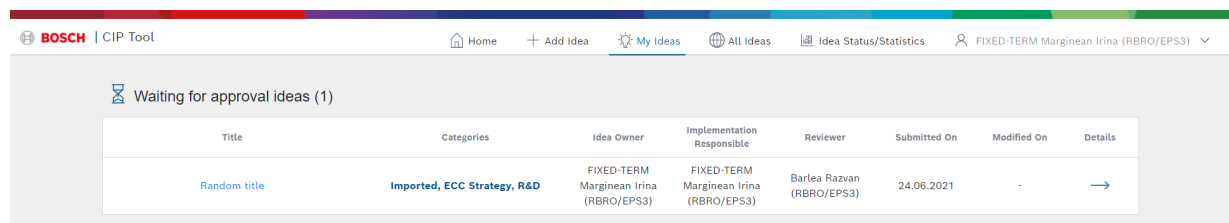


Figure 6.12: My Ideas page where the newly registered idea is displayed

What is more, as a part of the postconditions of the use case, the idea is also displayed in the Waiting for approval table in the Leader Response page of the reviewer of the idea, which is illustrated in the figure below.

The screenshot shows the 'Leader Response' page of the Bosch CIP Tool. At the top, there is a navigation bar with links: Home, Add Idea, My Ideas, All Ideas, Leader Response (active), Idea Status/Statistics, and a user profile for Barlea Razvan (RBRO/EPS3). Below the navigation bar, there is a section titled 'Waiting for approval ideas (1)'. This section contains a table with the following data:

Title	Categories	Idea Owner	Implementation Responsible	Submitted On	Modified On	Details
Random title	Imported, ECC Strategy, R&D	FIXED-TERM Marginean Irina (RBRO/EPS3)	FIXED-TERM Marginean Irina (RBRO/EPS3)	24.06.2021	-	→

Figure 6.13: Leader Response page of the reviewer where the newly registered idea is displayed

6.2 Text similarity algorithm validation

The results generated by the text similarity algorithm presented in section 4.4.3 *Approach #3 - SBERT and Cosine Similarity* have been validated using regression problems evaluation metrics. The results have been obtained based on the same dataset mention in section 5.4 *Machine Learning application*. The outcomes are available in the table below.

No.	Evaluation Metric	Value	Comments
1	Explained Variance Score	0.3349	best & max score is 1.0
2	Max Error	0.3975	the smaller the better
3	Mean Absolute Error	0.1237	the smaller the better
4	Mean Squared Error	0.0226	the smaller the better
5	Root Mean Squared Error	0.1506	the smaller the better
6	Mean Squared Log Error	0.0081	the smaller the better
7	Median Absolute Error	0.1026	the smaller the better
8	R2 Score	0.2895	best & max score is 1.0 (also negative)
9	Mean Poisson Deviance	0.0333	the smaller the better
10	Mean Gamma Deviance	0.0529	the smaller the better
11	Mean Absolute Percentage Error	0.2071	the smaller the better

Table 6.1: Evaluation metrics results run against the SBERT approach

The results present in Table 6.1 have been used to plot the bar chart below in order to visualize the outcomes of the validation of the algorithm in an effortless and straightforward manner.

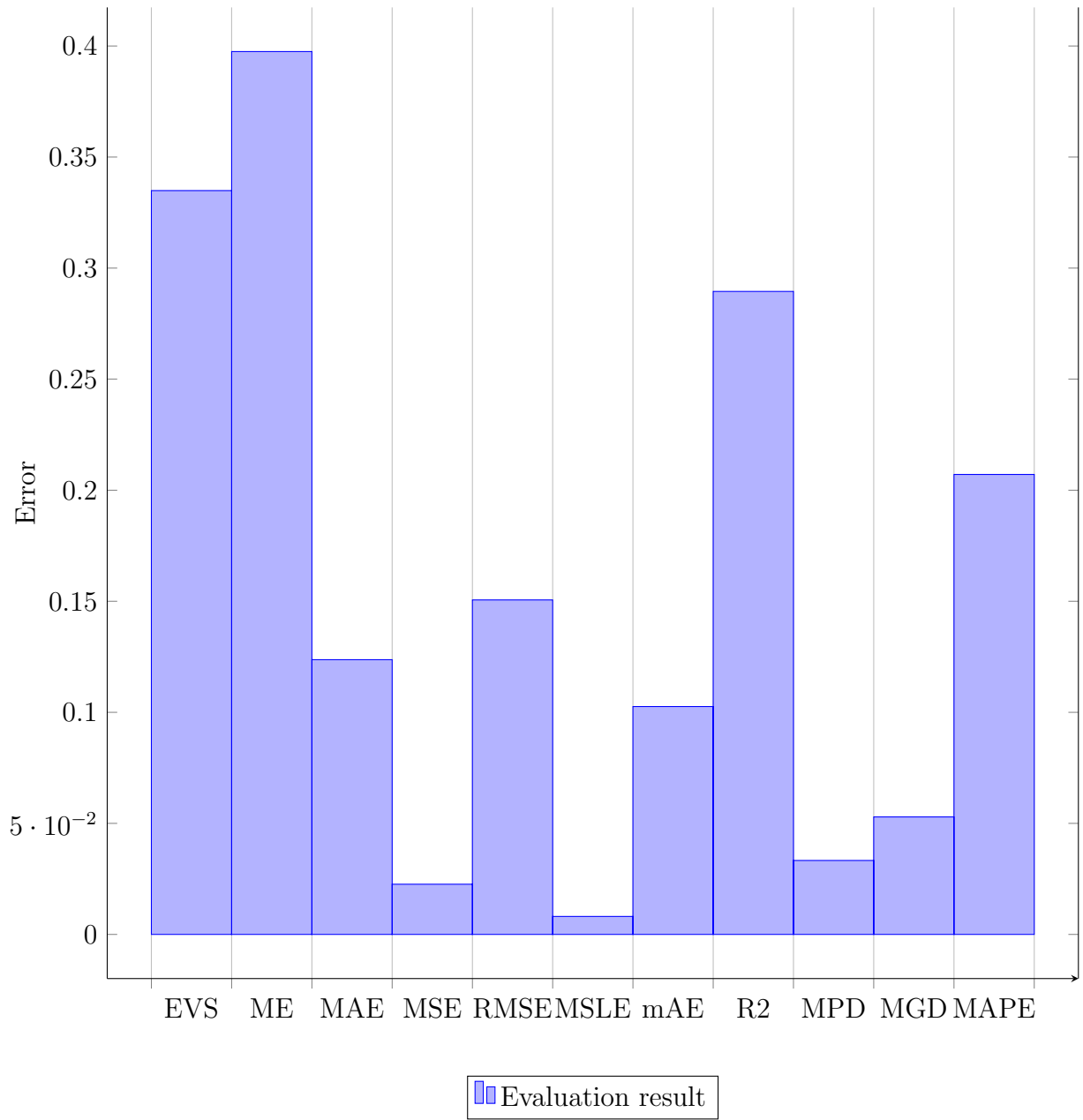


Figure 6.14: Plot of the evaluation metrics results

As it can be observed in Table 6.1 and Figure 6.14, the algorithm's performance is appropriate for its integration in a real system, not just for the use in an academic environment. Although the Explained Variance Score, R2 Score and Maximum Error can definitely be improved, the error regression loss measures offer satisfying results.

Chapter 7

User's manual

This chapter has the purpose of presenting the installation instructions in the form of a guide in order to allow any administrator to perform this operation, as well as providing a demonstration on how to use the tool for any type of user.

7.1 Installation guide

The system consists of four main modules, as presented in the previous chapters: MS SQL database, ASP.NET Core Web API, Python Machine Learning API and Angular frontend application. The Operating System which is best suitable for running the applications is Windows 10. The packages needed in the installation of each module and their integration in the system will be presented in the next sections.

7.1.1 MS SQL database

The applications needed to be installed in order to run and access the MS SQL database are:

1. SQL Server Express, available at [22] - used as the relational database management system to store and alter the data
2. SQL Server Management Studio, available at [23] - used to retrieve and modify the data in the database through a GUI

7.1.2 ASP.NET Core Web API

The applications needed to be installed in order to run the backend API are:

1. .NET Core 3.1, available at [24] - the SDK used in the development of the application
2. Visual Studio Community Edition, available at [25] - the IDE used in the development of the application

To run the application, it needs to be copied to the local computer, then run in Visual Studio in the Debug or Release mode after selecting the only executable project in the solution: *CIPTool*. To run the application, the green Start button must be pressed. In order to generate the database, the *DataAccessLayer* project must be selected in the Package Manager Console inside Visual Studio, then the *Update-Database* command must be run.

7.1.3 Python Machine Learning application

The applications needed to be installed in order to run the Machine Learning API are:

1. Python 3.9, available at [26]
2. PyTorch, available at [27]
3. all the packages in the *requirements.txt* file - run the command *pip install -r requirements.txt* in a Command Prompt
4. PyCharm, available at [28] - the IDE used in the development of the application

To be able to run the application, the source code has to be copied to the local machine and opened into PyCharm. To run the application, the green Start button must be pressed.

7.1.4 Angular frontend application

The applications needed to be installed in order to run the Angular website are:

1. Node.js, available at [29]
2. Angular CLI, available at [30]
3. Visual Studio Code, available at [31] - the code editor used in the development of the application

To be able to run the application, the source code has to be copied to the local machine and opened into Visual Studio Code. Firstly, the packages used by the website must be installed through the Terminal inside VS Code via the *npm install* command. Then, after the dependencies have been installed, the application can be started using the *ng serve* command in the same terminal.

7.2 System utilization

After the system is set up and all modules are running, the user must log into the tool using the Login page, where the username or e-mail address, and the password must be provided by the user.

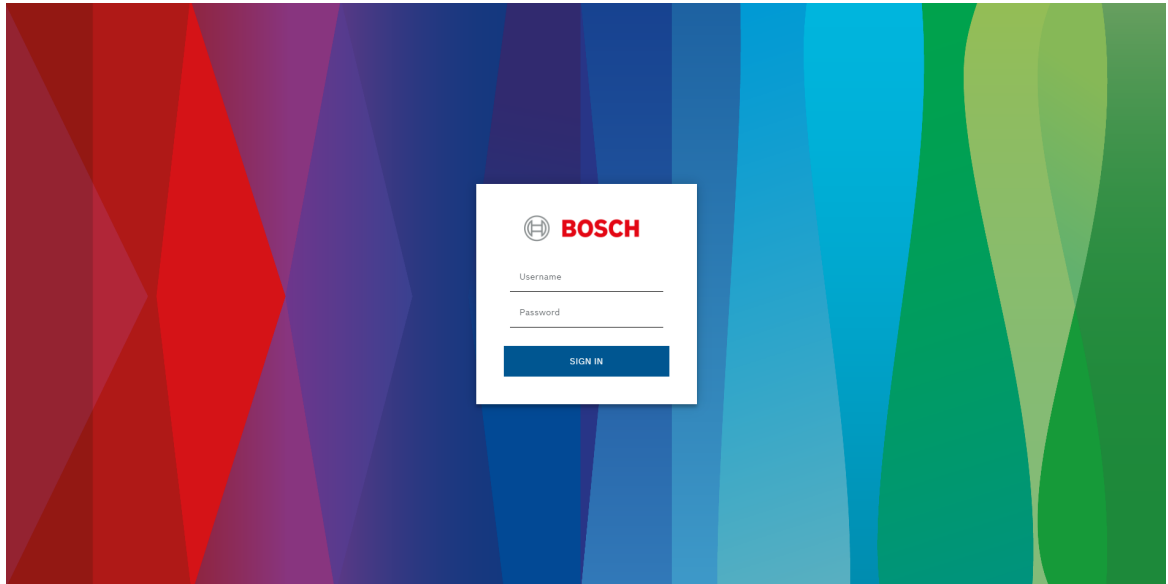


Figure 7.1: Login page

After a successful login, the user is redirected to the Home page.

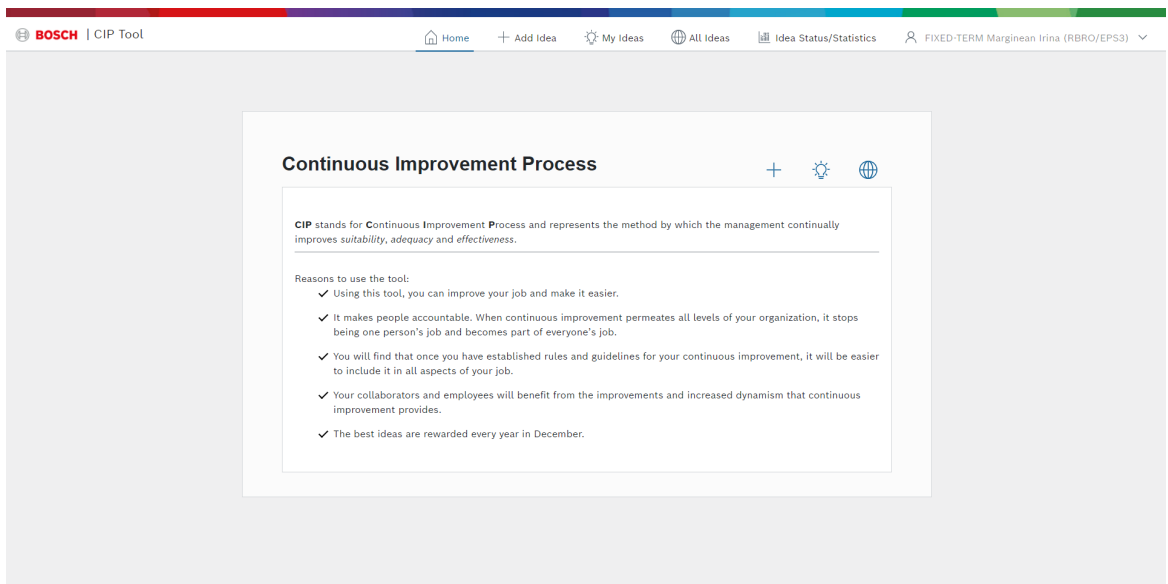


Figure 7.2: Home page

From the Home page, the user can access the Add Idea, My Ideas, All Ideas, Leader Response (if the user is either a leader or an administrator) and Idea Status/Statistics pages.

In the Add Idea page, the user must fill in the idea related fields.

The screenshot shows the 'Idea Owner Information' section of the 'Add Idea' page. The page header includes the BOSCH CIP Tool logo and navigation links: Home, + Add Idea, My Ideas, All Ideas, Idea Status/Statistics, and a user profile dropdown for FIXED-TERM Marginean Irina (RBRO/EP3). The 'Idea Owner Information' section contains the following fields:

- Name of idea owner:** Irina Marginean
- Is responsible for the implementation:** A toggle switch is currently turned off.
- Group:** EPS3
- Group Leader:** Razvan Barlea
- Department:** EPS
- Department Leader:** Felizian Aberham
- Name of the person who is responsible for the implementation:** Domnar Vlad-ilie (RBRO/PJ-PE)

Figure 7.3: Add idea page - Idea Owner Information section

The screenshot shows the 'Idea Summary' section of the 'Add Idea' page. The page header includes the BOSCH CIP Tool logo and navigation links: Home, + Add Idea, My Ideas, All Ideas, Idea Status/Statistics, and a user profile dropdown for FIXED-TERM Marginean Irina (RBRO/EP3). The 'Idea Summary' section contains the following fields:

- Predicted Idea Number:** 2021_RBRO/EP3/7
- Idea title:** A text input field.
- Current context:** A text input field.
- Target state:** A text input field.
- Categories:** A section with a 'Categories' label and a plus icon. It contains several buttons: ECC Strategy, Organization, Process (highlighted in dark blue), R&D, Comfort related, Test IDEA (highlighted in dark blue), and Imported.
- Other:** A text input field with the value 'Internal' and an 'Add' button.

Figure 7.4: Add idea page - Idea Summary section

Planned implementation date *
dd/mm/yyyy

Idea Description & Attachments

Idea Description

Normal
Sans Serif
B
I
U
A

Attachments

Figure 7.5: Add idea page - Idea Description & Attachments section

€ Financial Report & Bonus

Financial Report

Planned Savings
0 €

Planned Expenses
0 €

Planned Balance
0€

Planned Bonus
0€

Actual Savings
0 €

Actual Expenses
0 €

Actual Balance
0€

Actual Bonus
0€

Bonus

Lower Bound	Upper Bound	Award
-99999999€	499€	0€
500€	999€	50€
1000€	1999€	90€
2000€	2999€	160€
3000€	5999€	210€
6000€	9999€	300€
10000€	19999€	500€
20000€	29999€	900€
30000€	39999€	1100€
40000€	49999€	1200€
50000€	99999999€	1500€

✓ SUBMIT

✕ CANCEL

Figure 7.6: Add idea page - Financial Report & Bonus section

In order to register the idea, the Submit button must be pressed. After pressing the button, the results of the idea similarity computation are displayed in a dialog, containing the similar ideas' title and similarity score, as well as a link to them. If the user is sure they want to still submit the idea, the Confirm button must be pressed. Otherwise, the Cancel button can be pressed to cancel the submission.

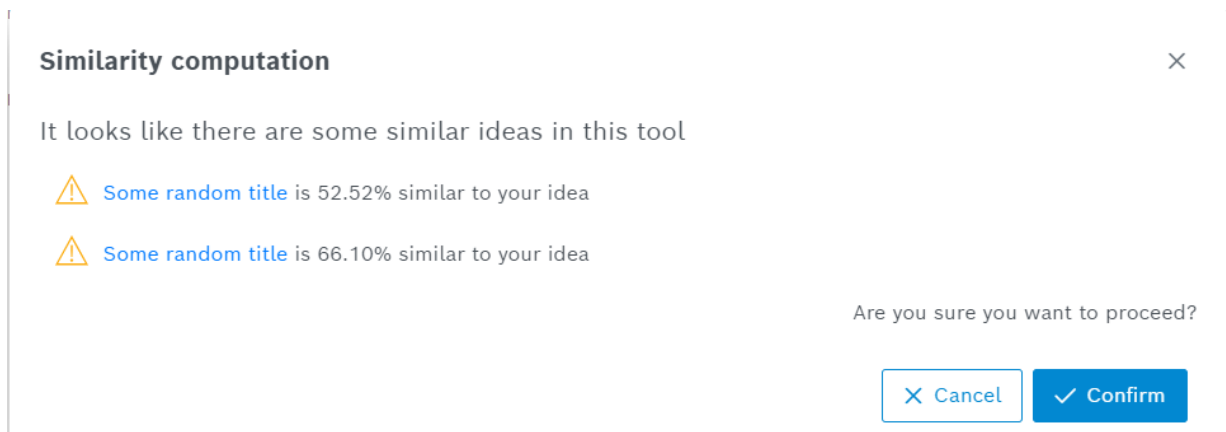


Figure 7.7: Add idea page - Similarity computation dialog

If the idea has been confirmed to be registered, then the user will be redirected to My Ideas page, where an overview of the current user's ideas is displayed.

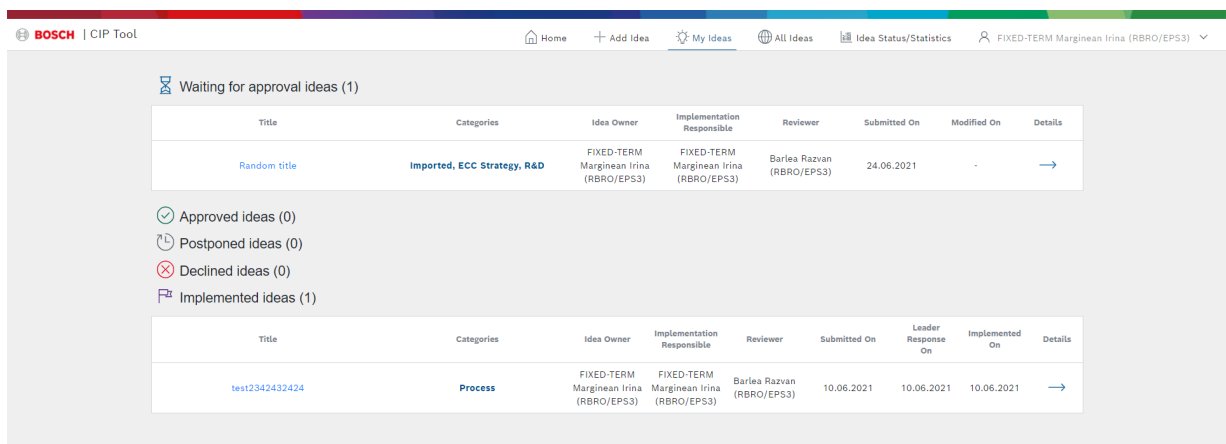


Figure 7.8: My ideas page

Furthermore, an overview of all submitted ideas can be viewed in the All Ideas page.

Waiting for approval ideas (6)

Title	Categories	Idea Owner	Implementation Responsible	Reviewer	Submitted On	Modified On	Details
Test for file upload	ECC Strategy, Test IDEA	Barlea Razvan (RBRO/EP3)	Barlea Razvan (RBRO/EP3)	Aberham Felizian (RBRO/EP3)	08.06.2021	08.06.2021	→
Some random title	Imported, Internal	Barlea Razvan (RBRO/EP3)	Barlea Razvan (RBRO/EP3)	Aberham Felizian (RBRO/EP3)	08.06.2021	08.06.2021	→
c	ECC Strategy	Barlea Razvan (RBRO/EP3)	Barlea Razvan (RBRO/EP3)	Aberham Felizian (RBRO/EP3)	08.06.2021	-	→
Random title	Imported, ECC Strategy, R&D	FIXED-TERM Marginean Irina (RBRO/EP3)	FIXED-TERM Marginean Irina (RBRO/EP3)	Barlea Razvan (RBRO/EP3)	24.06.2021	-	→
test	ECC Strategy, Organization	Domnar Vlad-Ilie (RBRO/PJ-PE)	Domnar Vlad-Ilie (RBRO/PJ-PE)	Greiner Rinaldo (RBRO/ESA RBRO/PJ-PE)	16.06.2021	-	→
Some random title	Test IDEA	Barlea Razvan (RBRO/EP3)	Barlea Razvan (RBRO/EP3)	Aberham Felizian (RBRO/EP3)	08.06.2021	-	→

✓ Approved ideas (0)
 ⌚ Postponed ideas (0)
 ✗ Declined ideas (0)
 📋 Implemented ideas (1)

Title	Categories	Idea Owner	Implementation Responsible	Reviewer	Submitted On	Leader Response On	Implemented On	Details
test2342432424	Process	FIXED-TERM Marginean Irina (RBRO/EP3)	FIXED-TERM Marginean Irina (RBRO/EP3)	Barlea Razvan (RBRO/EP3)	10.06.2021	10.06.2021	10.06.2021	→

Figure 7.9: All ideas page

The details of a specific idea can be viewed inside the Idea Details page, which can be accessed from either My Ideas or All Ideas page on the Details column in all tables. The details page looks similar to the Add Idea page, except for the leader response which can be viewed inside this page.

test2342432424 Implemented

2021_RBRO/EP3/5

Plan date on	Planned implementation date	Leader response on	Actual start implementation date	Actual implementation date
10.06.2021	10.06.2021	10.06.2021	22.06.2021	10.06.2021

Name of idea owner Irina Marginean	Group EPS3	Group Leader Razvan Barlea
is responsible for the implementation	Department EPS	Department Leader Felizian Aberham

Current context test12	
Target state test123	

Figure 7.10: Idea details page

Categories

ECC Strategy

Organization

Process

R&D

Comfort related

Test IDEA

Imported

Money saver

Idea Description

test cu irina

Attachments

CIP Award.pptx

Uploaded on: 10.06.2021

Financial Report

Planned Savings

1234€

Planned Expenses

23€

Planned Balance

1211€

Planned Bonus

90€

Actual Savings

0€

Actual Expenses

0€

Actual Balance

0€

Bonus

Lower Bound	Upper Bound	Award
-99999999€	499€	0€
500€	999€	50€
1000€	1999€	90€
2000€	2999€	160€
3000€	5999€	210€
6000€	9999€	300€
10000€	19999€	500€
20000€	29999€	900€
30000€	39999€	1100€
40000€	49999€	1200€

Figure 7.11: Idea details page

Leader responses

✓

Barlea Razvan (RBRO/EPG3) has approved the idea

aprob

10.06.2021

Figure 7.12: Idea details page - leader response

The idea can be edited by pressing the pencil icon next to the title of the idea in the Idea Details page. The idea can be edited on by the idea owner, implementation responsible,

owner and the administrator. In the Edit page, all fields can be modified, attachments can be removed or added, and the Actual start implementation date and the Actual implementation date can be filled in.

Figure 7.13: Edit page

The Leader Response overview page is a page which can be accessed only by leaders and administrators. It is very similar to the All Ideas and My Ideas page.

Title	Categories	Idea Owner	Implementation Responsible	Leader Response	Submitted On	Leader Response On	Implemented On	Details
Random title	Imported, ECC Strategy, R&D	FIXED-TERM Marginean Irina (RBRO/EP3)	FIXED-TERM Marginean Irina (RBRO/EP3)	Approved	24.06.2021	27.06.2021	-	→
test2342432424	Process	FIXED-TERM Marginean Irina (RBRO/EP3)	FIXED-TERM Marginean Irina (RBRO/EP3)	Implemented	10.06.2021	10.06.2021	10.06.2021	→

Figure 7.14: Leader response page - overview

The actual leader response can be added by clicking the right arrow inside the Details column. A page similar to the Idea Details page is displayed, along with a new section which allows giving one of the three types of responses, along with a reason.

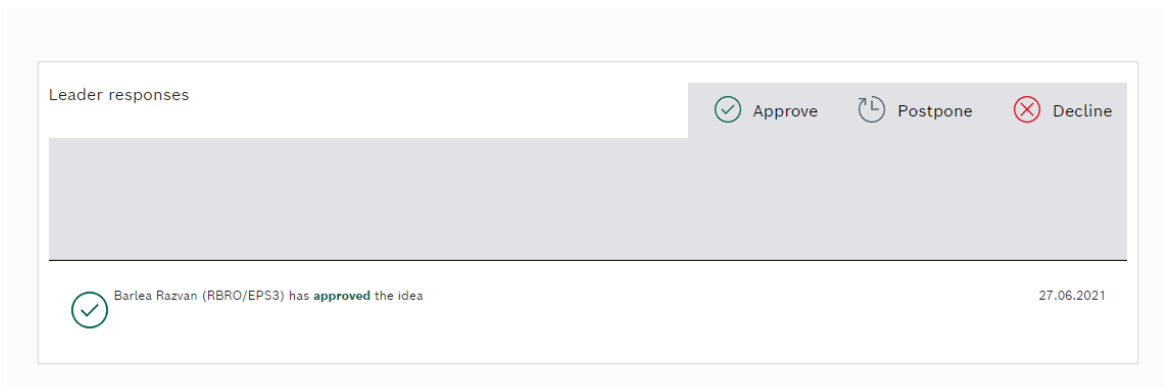


Figure 7.15: Leader response page - response section

The last page currently available in the system is the Idea Status/Statistics page. It displays various types of statistics based on the classification of registered ideas.

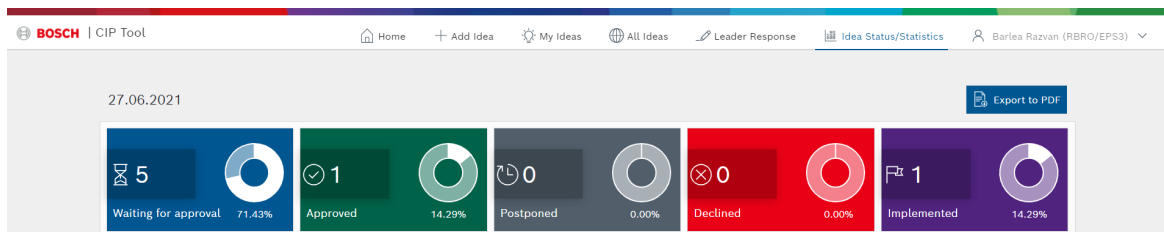


Figure 7.16: Statistics page

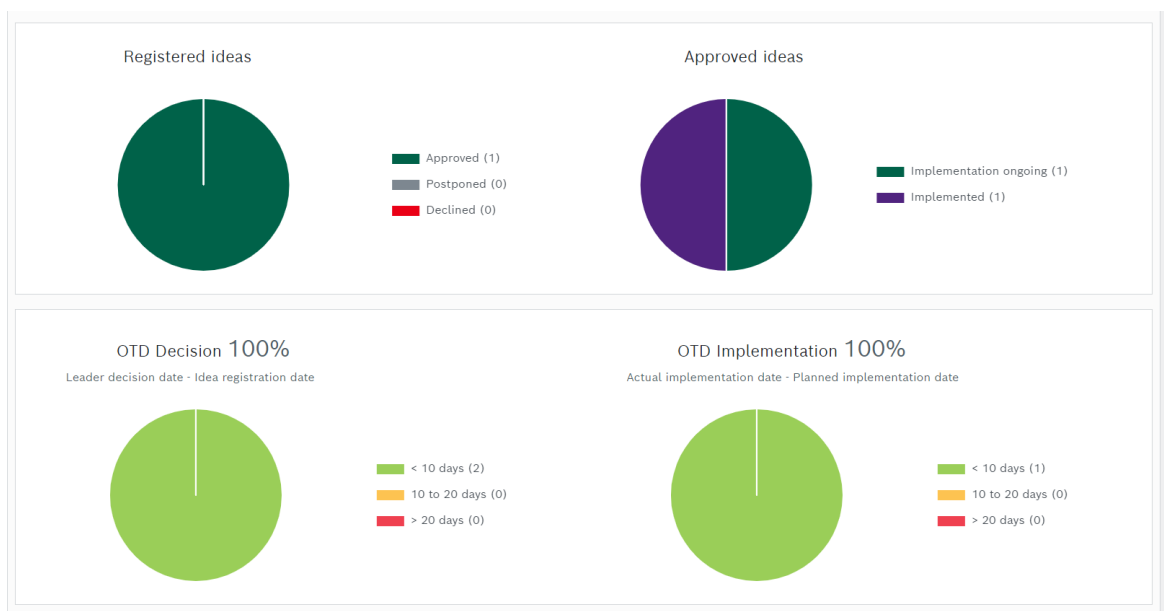


Figure 7.17: Statistics page

Furthermore, statistics related to the financial report and bonus of ideas can be viewed in the bottom part of the page.

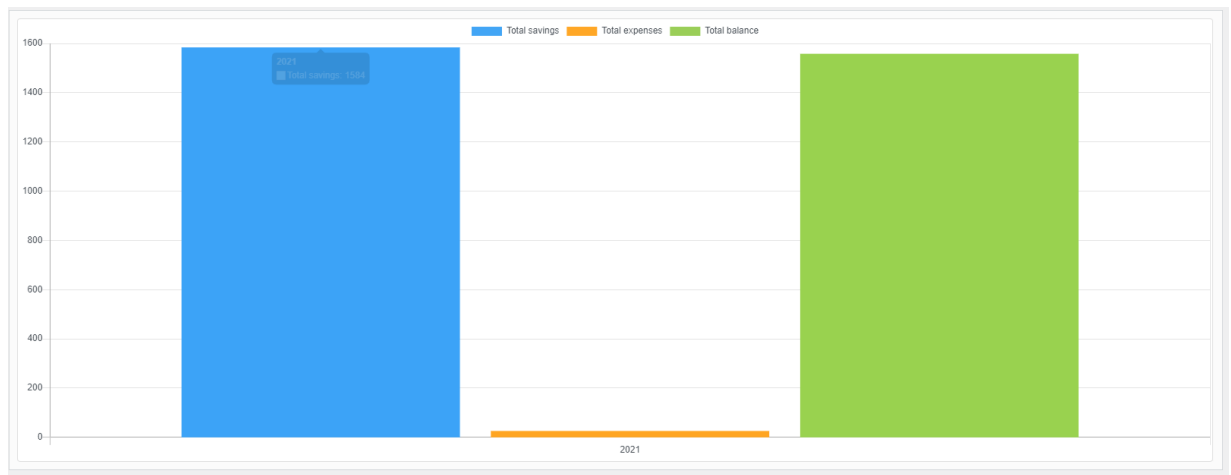


Figure 7.18: Statistics page

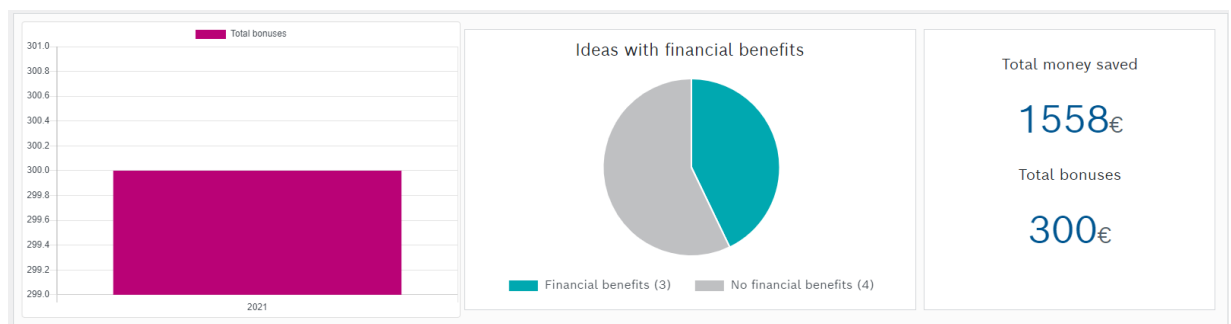


Figure 7.19: Statistics page

The statistics can be downloaded for further use as a PDF through the Export to PDF button at the right top of the page.

Chapter 8

Conclusions

This chapter has the purpose to present the summary and personal contributions of this project and paper, as well as the critical analysis of the obtained results and the future development and improvements.

8.1 Summary and contributions

The idea of implementing a system which aids in the Continuous Improvement Process employed in companies and organizations all over the world came after observing that businesses nowadays use methods such as e-mails, forms and spreadsheets over existing integrated systems, mainly due to their cost effectiveness. CIP Tool is an inexpensive, simple to use application which offers all the features necessary for improving processes inside companies and more, including the semantic similarity verification feature.

The development of the tool has reached all the objectives and functional specifications previously established in the design phase. These requirements have been requested and discussed with a real-life CIP administrator from Bosch Romania, namely Vlad Ilie Domnar. The requirements and specifications, as well as the implementation details related to the Machine Learning feature and some other general features offered by the system were thoroughly discussed with Zoltan Czako, the scientific coordinator. After the implementation phase, a fully functional Web tool has been created, composed of four different modules: a backend API, a Machine Learning API, a database and a frontend webiste.

The personal contributions consist of the design and the implementation of the system which allows employees from companies which follow the CIP principles to register ideas, view other ideas, edit ideas, review ideas and analyze statistics based on the ideas and their financial information, as well as the integration of the semantic text similarity verification into the tool. The semantic text similarity which is applied between the contents of ideas is a significant improvement compared to other tools on the grounds that it removes the necessity of sharing the knowledge base between former, current or future employees, and makes the reviewer's job substantially easier because they can instantly

reject ideas which have already been implemented or registered into the tool.

8.2 Critical analysis of the results

Based on the requirements and specifications presented in Chapter 2 and Chapter 4 of the application, the system can be considered complete, having implemented all base features and some nice-to-have ones needed for putting the tool into production so that it could be brought into play by real users.

The application can be accessed by multiple users at the same time without interference from one to the other, thanks to the asynchronous operations employed throughout the system. Moreover, as a result of the three-tier and MVC architectures used in the design, the system is flexible and scalable, which allows it to be easily extended or modified.

The complexity of the system comes from the varied technology stack used in the development of the Web applications, as well as the Machine Learning algorithm which computes the semantic similarity between ideas. The technologies used consist of C#, ASP.NET Core, Angular, TypeScript, PrimeNG, Bootstrap, HTML, CSS, Python, PyTorch and Flask.

8.3 Future improvements

Just like the CIP principles state, there is always room for improvement and further developments. In the case of CIP Tool, these future improvements include:

- researching and integrating/developing a text similarity algorithm which yields better results when evaluated with the regression metrics in section *6.2 Text similarity algorithm validation*
- personalizing the confirmation e-mails and adding more information inside them
- implementing the domain-based Windows authentication
- gathering user information from the Active Directory inside companies
- implementing a feature which allows leaders to change the reviewers of an idea
- improving the security on the Machine Learning API
- implementing a feature which allows viewing the edit history of an idea

Bibliography

- [1] E. Lodgaard, I. Gamme, and K. Aasland, “Success factors for pdca as continuous improvement method in product development,” *IFIP Advances in Information and Communication Technology*, vol. 397, pp. 645–652, 01 2013.
- [2] Continuous improvement models: four great options for you. [Online]. Available: <https://www.investorsinpeople.com/knowledge/continuous-improvement-models-four-great-options-for-you/>
- [3] P. Kosky and G. Wise. Lean manufacturing. [Online]. Available: <https://www.sciencedirect.com/topics/engineering/lean-manufacturing>
- [4] M. Sokovic, J. Šaković Jovanović, Z. Krivokapic, and A. Vujovic, “Basic quality tools in continuous improvement process,” *Strojniski Vestnik*, vol. 55, pp. 333–341, 05 2009.
- [5] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, “Natural language processing: an introduction,” *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, 09 2011.
- [6] W. Gomaa and A. Fahmy, “A survey of text similarity approaches,” *international journal of Computer Applications*, vol. 68, 04 2013.
- [7] D. Prasetya, A. Wibawa, and T. Hirashima, “The performance of text similarity algorithms,” *International Journal of Advances in Intelligent Informatics*, vol. 4, 05 2018.
- [8] S. Smith. (2020, Dec.) Overview of asp.net core mvc. [Online]. Available: <https://docs.microsoft.com/ro-ro/aspnet/core/mvc/overview?view=aspnetcore-5.0>
- [9] J. Rabelo. (2021, Jan.) Three-tier architecture. [Online]. Available: <https://www.techopedia.com/definition/24649/three-tier-architecture>
- [10] What is http? [Online]. Available: <https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/>
- [11] What is rest. [Online]. Available: <https://restfulapi.net/>

- [12] Semeval sts task. [Online]. Available: <https://github.com/brmson/dataset-sts/tree/master/data/sts/semeval-sts>
- [13] Mean squared error: Definition and example. [Online]. Available: <https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error/>
- [14] Cosine similarity. [Online]. Available: <https://deepai.org/machine-learning-glossary-and-terms/cosine-similarity>
- [15] C. Van den Rul. (2019, Sep.) Understanding word embeddings with tf-idf and glove. [Online]. Available: <https://towardsdatascience.com/understanding-word-embeddings-with-tf-idf-and-glove-8acb63892032>
- [16] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. [Online]. Available: <https://nlp.stanford.edu/projects/glove/>
- [17] J. Pennington, R. Socher, and C. Manning, “GloVe: Global vectors for word representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: <https://www.aclweb.org/anthology/D14-1162>
- [18] coderasha. (2019, Sep.) Compare documents similarity using python | nlp. [Online]. Available: <https://dev.to/coderasha/compare-documents-similarity-using-python-nlp-4odp>
- [19] J. Pennington, R. Socher, and C. D. Manning. Gensim: Core concepts. [Online]. Available: https://radimrehurek.com/gensim/auto_examples/core/run_core_concepts.html#sphx-glr-auto-examples-core-run-core-concepts-py
- [20] B. Lutkevich. Bert language model. [Online]. Available: <https://searchenterpriseai.techtarget.com/definition/BERT-language-model>
- [21] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” 01 2019, pp. 3973–3983.
- [22] Sql server express. [Online]. Available: <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>
- [23] Sql server management studio. [Online]. Available: <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>
- [24] .net core 3.1. [Online]. Available: <https://dotnet.microsoft.com/download/dotnet/3.1>

- [25] Visual studio community edition. [Online]. Available: <https://visualstudio.microsoft.com/vs/>
- [26] Python 3.9. [Online]. Available: <https://visualstudio.microsoft.com/vs/>
- [27] Pytorch. [Online]. Available: <https://pytorch.org/>
- [28] Pycharm. [Online]. Available: <https://www.jetbrains.com/pycharm/download/#section=windows>
- [29] Node.js. [Online]. Available: <https://nodejs.org/en/>
- [30] Angular cli. [Online]. Available: <https://nodejs.org/en/>
- [31] Visual studio code. [Online]. Available: <https://code.visualstudio.com/download>

Appendix A

GUI Mockups

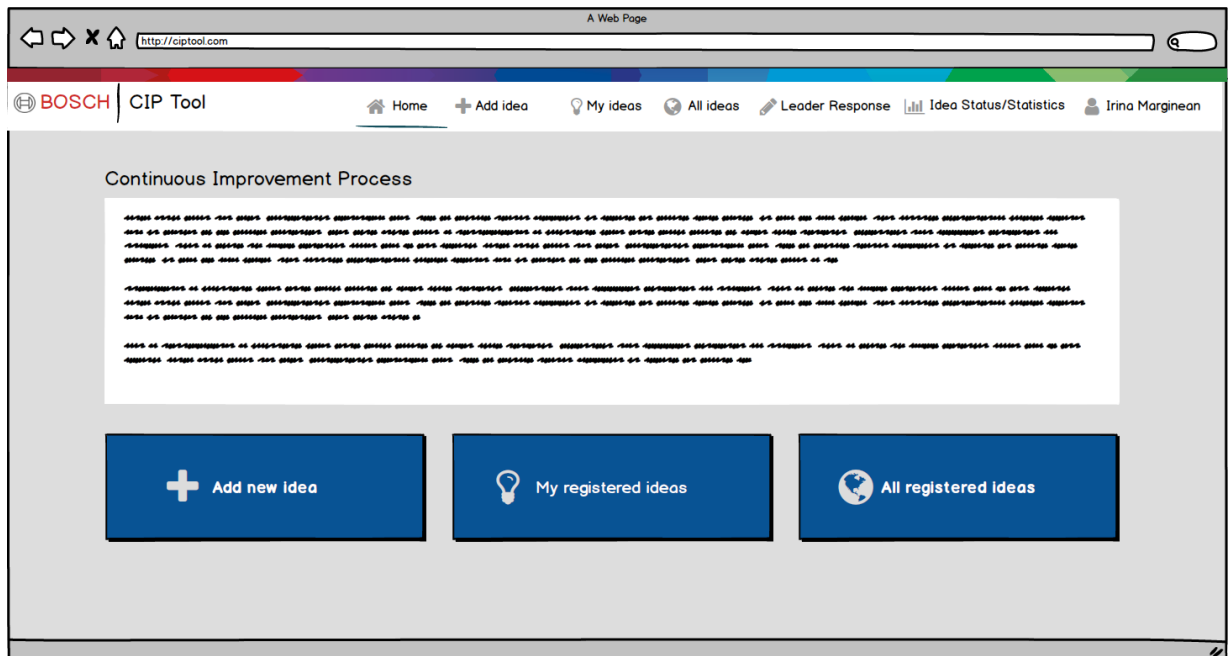


Figure A.1: Home page

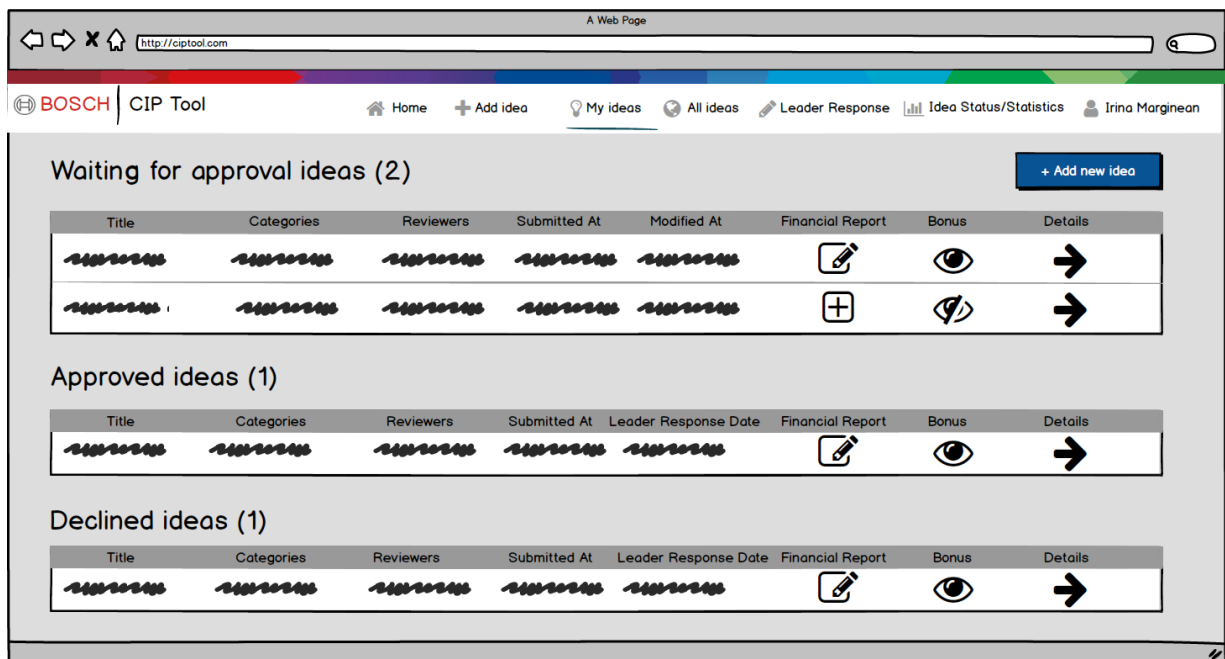


Figure A.2: My ideas overview page

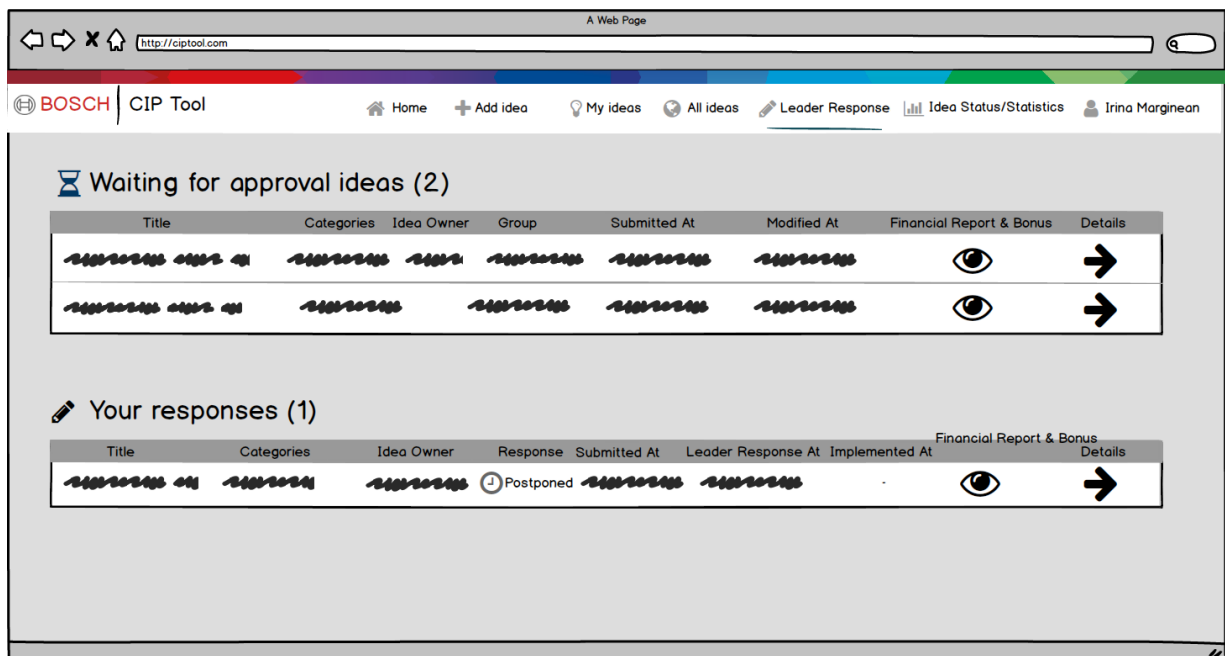


Figure A.3: Leader Response overview page

↑

91



92

A Web Page
http://capital.com

BOSCH
CIP Tool
Home
Add idea
My ideas
All ideas
Leader Response
Idea Status/Statistics
Inna Marginean

Idea title

2021_2801/EP3012
Waiting for approval

02.03.2021

Name of idea creator
Inna Marginean

Department
Department leader

Is responsible for the implementation

Group
Group leader

Current state

Target state

Categories
☒ Money saver

Idea Description

Attachments

photo1.png
Uploaded at: 2021-01-10

photo3.png
Uploaded at: 2021-01-10

photo2.png
Uploaded at: 2021-01-10

Financial Report

Planned values

Actual values

Savings

Expenses

Balance

Bonus

Bonus

benefits/savings (estimated or calculated annual net saving in EUR)	Awards (Net)	Solution has been...	Correction factor
500,00	299,00	x never discussed in the organization	1
1000,00	199,00		
2000,00	299,00		
3000,00	599,00		
6000,00	999,00	previously discussed in the organization	0,50
10000,00	1999,00		
20000,00	2999,00		
30000,00	3999,00		
X 40000,00	4999,00		
Above	5000,00		

Leader Response

Reason

Figure A.6: Leader Response page

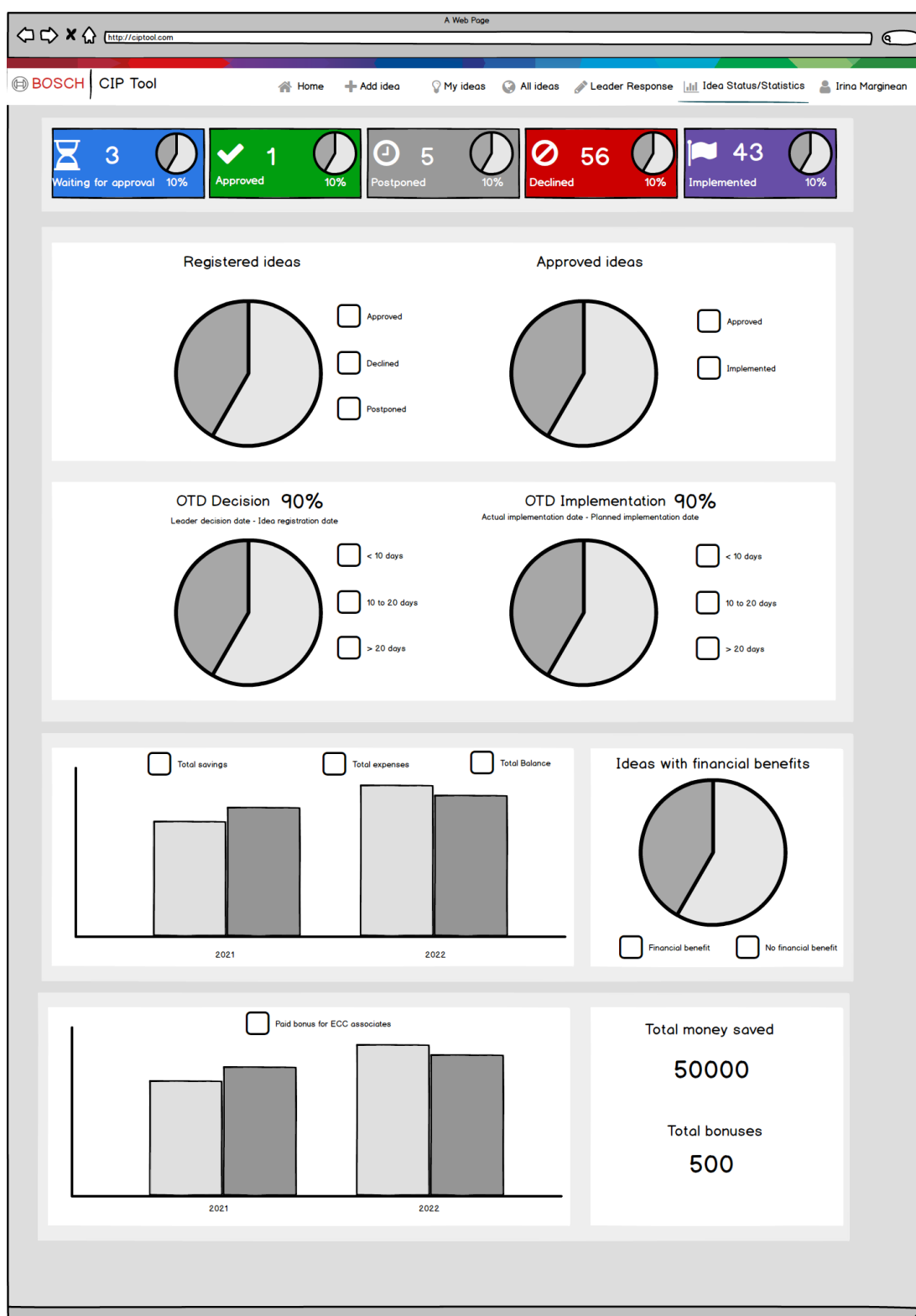


Figure A.7: Statistics page