Journal of Universal Computer Science, vol. 30, no. 11 (2024), 1455-1483
submitted: 3/7/2023, accepted: 11/5/2024, appeared: 28/10/2024

Microservices Patterns Recommendation based on Information Retrieval

Álex dos Santos Moura
(Federal University of Sergipe, Aracaju, Sergipe, Brazil
https://orcid.org/0000-0003-0906-7423, alexmoura658@gmail.com)

Fábio Gomes Rocha
(Federal University of Sergipe, Aracaju, Sergipe, Brazil
https://orcid.org/0000-0002-0512-5406, gomesrocha@gmail.com)

Michel S. Soares
(Federal University of Sergipe, Aracaju, Sergipe, Brazil
https://orcid.org/0000-0002-7193-5087, michel@dcomp.ufs.br)

Abstract: Software developers encounter recurring problems during software development, which can be solved using proven solutions known as design patterns. Microservices architecture, a Service-Oriented Architecture (SOA) variant, presents common communication, deployment, and service definition challenges. However, selecting the appropriate design pattern from a vast pool of patterns to solve a problem is difficult for novice and experienced developers. This paper proposes a recommendation tool based on Information Retrieval (IR) to assist developers in choosing the suitable microservices pattern to solve a given problem. The tool leverages textual descriptions given by developers to provide relevant indications of microservices patterns. It has been evaluated using both toy and industrial problems, demonstrating promising results. The results showed that the tool was able to solve 60% of toy design problems, indicating that it can provide valuable and accurate recommendations. Furthermore, tests with industrial problems revealed that over 70% of the recommended patterns helped to address the problems at hand. Interviews with developers who work in the software industry corroborated the recommendation tool's effectiveness and practicality.

Keywords: Microservices, Recommendation, Design Patterns, Information Retrieval, Microservices Patterns
Categories: H.3.1, H.3.2, H.3.3, H.3.7, H.5.1
DOI: 10.3897/jucs.108974

1 Introduction

Software developers have to deal with a variety of problems when they are developing or maintaining software products. Fortunately, some problems are already known to arise recurrently during software development. To solve these known problems, developers use design patterns that are proven solutions designed by other developers from their experiences [Hussain et al. 2019].

The notion of using common patterns for solving recurrent software problems has been known for decades. Design patterns were proposed in the literature for object-oriented design [Gamma et al. 1995]. Later on, the notion of a service was introduced

and defined as autonomous, platform-independent computational entities that can be used to create large, complex applications. Then, Service-Oriented Architecture (SOA) is proposed as an architectural paradigm that uses services as a fundamental element for developing software applications.

The notion of microservice was then introduced as a cohesive, independent process interacting via messages. The term "cohesive" indicates that a service implements only functionalities strongly related to the concern that it is meant to model [Dragoni et al. 2017].

The microservice architecture is a distributed application where all its modules are microservices. There are advantages of using microservices-based architectures for software development, such as small code base, gradual transitions to new versions of a microservice, leading to continuous integration and easier software maintenance, and freedom to choose resources for the implementation of each microservice.

Developers working on software based on the microservices architecture face recurring problems that are related, for example, to communication, deployment and definition of services [Balalaie et al. 2016]. Microservices architecture is a variant of SOA (Service-Oriented Architecture) [Soares and França 2016, Niknejad et al. 2018], which proposes the development of systems based on independent services, small and responsible for a single business domain.

To solve common problems in such software systems, developers can use design patterns that are described, for instance, in the popular book named "Microservices Patterns: With examples in Java" [Richardson 2018]. Design patterns applied to these systems are also known as microservices patterns.

Systems with many design problems are subject to being discontinued or re-engineered [Sousa et al. 2018], as these problems negatively impact their quality attributes [Sousa et al. 2017]. This shows that it is important to solve design problems, and as explained, these problems can be solved through design patterns.

Thus, solving design problems is theoretically simple, the developer needs to select design patterns and apply them. However, in practice, beginner and experienced developers may face difficulties when selecting design patterns, given the substantial number of design patterns described in the past decades and limited knowledge [Hamdy and Elsayed 2018, Sanyawong and Nantajeewarawat 2015]. Therefore, this work seeks to offer a solution to this difficulty, which is important because it can help to prevent microservices-based systems from being discontinued or re-engineered.

Thus, the objective of this work is to provide a way to help software developers choose the appropriate microservices pattern to solve a problem. Thus, this study proposes a recommendation tool based on IR (Information Retrieval), where a developer can textually describe the problem he/she needs to solve and then obtain indications of microservices patterns. Parts of this article have been published in a Master Thesis [Moura 2023].

The remainder of the paper is organized as follows. Section 2, where important concepts for understanding the work are presented, Section 3, where related works are presented, Section 4, which shows how the tool was developed and how it works, Section 5, which presents results from the tests performed, and Section 6 which describes threats to validity. The paper ends with Section 7 about discussion and Section 8, where a conclusion about the study is reported and future work is also pointed out.

2 Theoretical Foundation

2.1 Microservices

The microservices architecture consists of a set of best practices involving, for example, DevOps and containers [Thönes 2015, Koschel et al. 2017, Yugopuspito et al. 2017]. The main components of this architecture are the microservices which, according to [Dragoni et al. 2017], are small, decoupled services specialized in tasks of a single context. Communication between microservices takes place over the network, as they are embedded in different processes when they are running [Mazlami et al. 2017, Balalaie et al. 2018]. Common communication mechanisms are REST (Representational State Transfer), API (Application Programming Interface) and messaging [Steinmetz et. al., 2018].

The benefits offered by the microservices architecture are diverse [Chen et al. 2017], because of the characteristics of microservices, systems that are based on this architecture are easier to maintain, scale and deploy. Thus, this architecture style has been the target of scientific research and also projects in industry [Mazlami et al. 2017, Cerny et al. 2018, Abgaz et al,. 2023].

Within the monolithic architecture [Baresi et al. 2017], all functionalities of any context are in a single software system [Taibi et al. 2017]. The communication between the modules of a monolith happens through the invocation of methods since it is inserted in only one process when it is running.

Monolithic architectures present problems that are visible in monoliths during their evolution process [Fritzsch et al. 2019]. According to [Dragoni et al. 2017], the microservices architecture emerged to deal with the problems present in monoliths, some of which are:

- It is difficult to evolve and maintain large monoliths, as modules are coupled and there are many complex lines of code;
- Large monoliths may have to deal with a term known as “Dependency Hell”, where adding or updating some libraries can compromise the system, and make it difficult, for example, to compile it;
- To release new features or bug fixes in large monoliths, takes a substantial period, as any modification to the system requires a total re-deployment of it.

2.2 Design Patterns

The concept of patterns in Software Engineering was adopted from Alexander’s writings [Richardson 2018]. In Software Engineering, a pattern is a proven solution to a design problem common in software projects [Sanyawong and Nantajeewarawat 2015].

A pattern, generally speaking, is embedded in a collection of related patterns that solve problems in a specific domain. In the context of microservices, one example is a collection with 44 patterns that are distributed into 16 categories [Richardson 2018], as can be seen in Table 1.

Generally, a pattern is described through two sections, which are the problem domain and the solution domain. The problem domain exposes the context in which a pattern is applied, while the solution domain presents the structure to solve a problem [Hussain et al. 2019]. Table 4 shows the information used to describe a pattern [Richardson 2018], except for id and link.

Category	Patterns
Application Architecture Patterns	Monolithic Architecture and Microservice Architecture
Decomposition Patterns	Decompose by Business Capability and Decompose by Subdomain
Messaging Style Patterns	Messaging and Remote Procedure Invocation
Reliable Communications Patterns	Circuit Breaker
Service Discovery Patterns	3rd Party Registration, Client-Side Discovery, Self-Registration and Server-Side Discovery
Transactional Messaging Patterns	Polling Publisher, Transaction Log Tailing and Transactional Outbox
Data Consistency Patterns	Saga
Business Logic Design Patterns	Aggregate, Domain Event, Domain Model, Event Sourcing and Transaction Script
Querying Patterns	API Composition and Command Query Responsibility Segregation
External API Patterns	API Gateway and Backend for Frontend
Testing Patterns	Consumer-Driven Contract Test, Consumer-Side Contract Test and Service Component Test
Security Patterns	Access Token
Cross-Cutting Concerns Patterns	Externalized Configuration and Microservice Chassis
Observability Patterns	Application Metrics, Audit Logging, Distributed Tracing, Exception Tracking, Health Check API and Log Aggregation
Deployment Patterns	Deploy a Service as a Container, Deploy a Service as a VM, Language-Specific Packaging Format, Service Mesh, Serverless Deployment and Sidecar
Refactoring to Microservices Patterns	Anti-Corruption Layer and Strangler Application

Table 1: Microservices Patterns

2.3 Information Retrieval

Information Retrieval (IR) is the process of retrieving documents that match a query [Hambarde and Proenca 2023]. A Document (D) is an information unit that can be, for example, a text document (Web page, report, email and plain text) or even a multimedia file (image, video and audio). An IR-based system is called an Information Retrieval System (IRS). Google, a search engine, is an example of IRS, where a user submits a query and receives a set of matching Web pages as a result [Ibrihich et al. 2022].

An IRS that deals with text documents performs 3 steps to retrieve them: 1) Preprocess Documents, 2) Represent Documents and Query, and 3) Match [Roshdi and Roohparvar 2015]. Figure 1 shows these steps, including their inputs and outputs.

2.3.1 Step 1: Preprocess Documents

In this step, the text documents and the query are subjected to a set of treatments. It is necessary to apply these treatments so that the IRS has quality. There are 3 main treatments, the first is called “Tokenization”, the second is “Stop Word Removal” and the third is “Stemming”.

Tokenization consists of dividing the text into tokens. A token can be a word, phrase or symbol [Ceri and et al. 2013]. To exemplify, one can consider this text “Tokenization is important”, if a token is considered as a word, then in this text there are 3 tokens: “Tokenization”, “is” and “important”. If a token is considered as a phrase, then in this text there is only 1 token which is “Tokenization is important”. When a token is composed of just 1 word, it is called “Unigram”, when it is composed of 2 words it is called “Bigram” and so on.

After obtaining the tokens from a text, the Stop Word Removal treatment is applied. Words that are not relevant to the retrieval process, such as prepositions and articles,

are removed from the tokens [Ibrihich et al. 2022]. To exemplify, one can consider these 3 tokens: “Tokenization”, “is” and “important”. Applying the Stop Word Removal treatment, the “is” token is removed, after applying this treatment, there are only 2 tokens which are “Tokenization” and “important”.

Stemming treatment is applied after the Stop Word Removal treatment. The goal of Stemming treatment is to reduce inflected words – words that have different grammatical forms or conjugations – to their stem, minimize the number of words, ensure that stems are matched with precision and save memory [Ibrihich et al. 2022]. To exemplify, one can consider the tokens “Tokenization” and “important”, applying the Stemming treatment, the result is “Token” and “import”.

After being subjected to these treatments, the text documents and the query are called preprocessed text documents and a preprocessed query. As Figure 1 shows, the preprocessed text documents and the preprocessed query are the outputs of this step. In addition to these treatments, there are others that are more specific, as can be seen in Subsubsection 4.2.1.

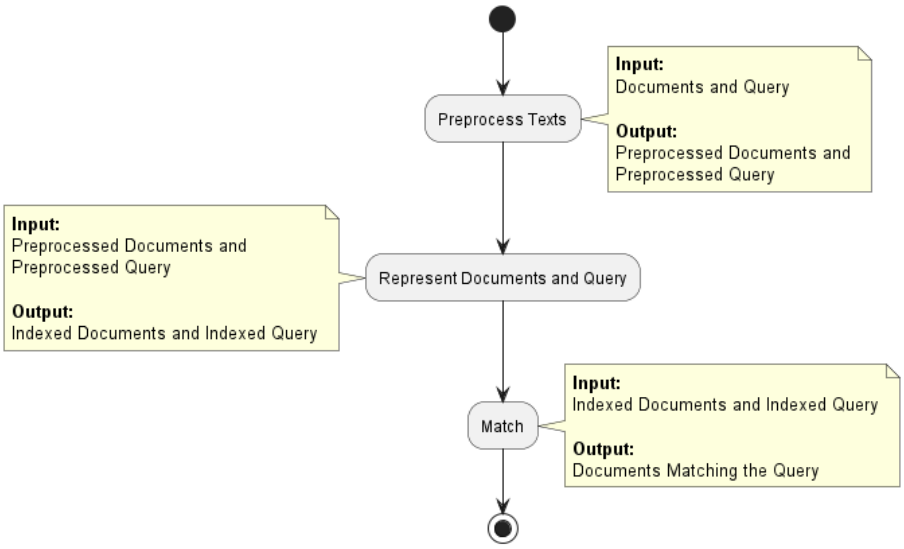

Figure 1: Steps Performed by an Information Retrieval System.

2.3.2 Step 2: Represent Documents and Query

In this step, the preprocessed text documents and the preprocessed query are represented using models so that it is possible to compare them, aiming to retrieve the matching text documents. There are two classic models, the Boolean model and the Vector Space Model (VSM) [Ibrihich et al. 2022].

Initially, the two models form a vocabulary from preprocessed text documents. In the Boolean model, when a preprocessed text document or a preprocessed query contains

a vocabulary word, this word receives the value of "true" or "1", otherwise, this word receives the value of "false" or "0". Table 2 shows an example, where this "Token" preprocessed query is represented and the following preprocessed text documents are represented:

- **Document 1:** Token import;
- **Document 2:** Token Inform Retriev;
- **Document 3:** Master dissert.

Words	Query	Document 1	Document 2	Document 3
Token	1	1	1	0
import	0	1	0	0
Inform	0	0	1	0
Retriev	0	0	1	0
Master	0	0	0	1
dissert	0	0	0	1

Table 2: Boolean Model: Documents Represented.

In the VSM, the preprocessed text documents and the preprocessed query are represented by vectors [Ibrihich et al. 2022]. Subsubsections 4.2.2 and 4.2.3 explain how to measure how relevant a word is to a document, considering all documents, using the VSM. In the Boolean model, it is not possible to do this. As Figure 1 shows, the outputs of this step are the represented documents and the represented query.

2.3.3 Step 3: Match

This step is where the documents that match the query are retrieved. Considering Table 2, where the documents and the query were represented using the Boolean model, documents 1 and 2 would be retrieved, as they are the only ones that have the "Token" word.

In the Boolean model, logical operators (AND, OR, and NOT) can be used [Ibrihich et al. 2022]. Therefore, it is possible to have, for example, a query such as "Master AND dissert". In this case, only document 3 would be retrieved, as it is the only document that has the words "Master" and "dissert".

In the case of the VSM, in this step, a measure of similarity is to check how similar a document is to a query. Documents that have a considerable degree of similarity are retrieved. In Subsubsection 4.2.3, this process is explained in detail. This step is the last one, its outputs are the recovered documents.

2.3.4 Evaluation Metrics

It is possible to measure the quality of an IRS through metrics such as Precision, Recall, F1-Score and Coverage. To explain these metrics, one can consider a collection/set of documents called C, where $C = \{D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8\}$, and Table 3 that exposes query examples.

Query (Q)	Important Documents (IDs)	Retrieved Documents (RDs)
Q1	D_2 and D_8	D_2, D_5, D_6 and D_8
Q2	D_2, D_6 and D_8	D_1, D_3, D_7 and D_8
Q3	D_5	D_2, D_5, D_6 and D_7
Q4	D_2, D_4, D_6 and D_8	D_2, D_4, D_6 and D_8

Table 3: Query Examples.

The Precision metric indicates the relationship between the number of Important Documents Retrieved (IDR) and RDs. This metric is given by Equation 1. Considering Table 3, the Precision of Q1 is 0.5 or, multiplying by 100, 50%. This means that 50% of the RDs are IDs. For other queries, Precision is Q2 (25%), Q3 (25%) and Q4 (100%). The higher the Precision, the better.

$$Precision = \frac{|important\ documents \cap retrieved\ documents|}{|retrieved\ documents|} \tag{1}$$

The Recall metric indicates the relationship between the number of IDR and IDs. This metric is given by Equation 2. Considering Table 3, the Recall of Q1 is 1.0 or, multiplying by 100, 100%. This means that 100% of the IDs were retrieved. For the other queries, the Recall is: Q2 (≈35%), Q3 (100%) and Q4 (100%). The higher the Recall, the better.

$$Recall = \frac{|important\ documents \cap retrieved\ documents|}{|important\ documents|} \tag{2}$$

The Precision and Recall metrics are frequently used to measure the quality of an IRS [Saini et al. 2014]. From these two metrics, it is possible to calculate the F1-Score metric which is especially useful when the aim is to balance the need to retrieve IDs (Recall) and the need to ensure that the documents retrieved are really important (Precision). The F1-Score metric is given by Equation 3.

$$F1\text{-}Score = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall} \tag{3}$$

Considering Table 3, the F1-Score of Q1 is ≈0.70 or, multiplying by 100, ≈70%. This means that for Q1, the IRS had a performance of ≈70%, combining Precision and Recall. For the other queries, the F1-Score is: Q2 (≈30%), Q3 (40%) and Q4 (100%). The higher the Recall, the better.

The Coverage metric indicates the proportion of IDs that the IRS was able to retrieve. This metric is given by Equation 4. Considering Table 3, there are 5 IDs available (D_2, D_4, D_5, D_6 and D_8), all these 5 documents were retrieved. Thus, the IRS had a Coverage of 1.0 or, multiplied by 100, 100%.

$$Coverage = \frac{total\ important\ documents\ retrieved}{total\ number\ of\ important\ documents\ available} \tag{4}$$

3 Related Works

In this Section, some work related to the recommendation of design patterns is discussed as well as the differences of this work concerning them. Each work presented suggests a different recommendation approach, using Information Retrieval, Text Classification or Mix. These works are grouped by type of approach and presented in specific Subsections.

3.1 Approach: Information Retrieval

Hamdy and Elsayed [Hamdy and Elsayed 2018] propose a GoF patterns recommendation approach that is based on information retrieval plus LDA (Latent Dirichlet Allocation). For the recommendation of these patterns, this approach proved to be superior to traditional information retrieval with unigram vector space. The authors stated this after carrying out tests with 29 problems, obtaining a precision rate equal to 72%.

Rahmati and the other authors [Rahmati et al. 2019] also deal with the recommendation of GoF patterns through an approach, suggested by them, which is based on information retrieval. This approach makes use of an improved weighting algorithm that proved to be superior to the original algorithm after the authors ran tests with 29 problems and obtained superior results in terms of precision (8.5%), recall (1.2%) and accuracy (5.2%). It is worth mentioning that out of 29 problems used in the tests, 9 are industrial problems and 20 are toy problems.

Hussain and other authors [Hussain et al. 2019], in addition to dealing with recommending GoF patterns, also deal with recommending security patterns [Schumacher et al. 2013], patterns mentioned by Landay and Hong [Landay and Hong 2003], and Douglass [Douglass 2003]. The authors propose a recommendation approach based on information retrieval and "Learning to Rank". Their approach uses 3 algorithms called Coordinate Ascent, AdaRank and LambdaMART. After the authors test 47 industrial problems and observe 2 metrics called MAP (Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain), they conclude that LambdaMART is superior to other algorithms and that the approach they propose is promising.

3.2 Approach: Text Classification

Sanyawong and Nantajeewarawat [Sanyawong and Nantajeewarawat 2015] deal with recommending GoF design patterns [Gamma et al. 1995]. Their goal is to improve on part of a hierarchical recommendation approach they developed in a previous work [Sanyawong and Nantajeewarawat 2014]. With this in mind, the authors use textual classification algorithms, such as Naive Bayes, J48, K-NN and SVM, to relate a given design problem to one of the 3 categories of GoF patterns: creational, structural and behavioural. As a form of evaluation, the authors test 26 industrial problems and use 3 metrics, precision, recall and F-measure. After analyzing the results, the authors conclude that with this improvement, developers can select a GoF pattern more quickly and that the software industry can use the approach they present.

Silva-Rodríguez and the other authors [Silva-Rodríguez et al. 2020] deal with recommending interaction patterns to help designers develop better user interfaces. The recommendation approach they propose is based on textual classification, where 4 algorithms are used: Logistic Regression, Multinomial Naive Bayes, Linear SVM and Random Forest. After the authors tested the approach, Linear SVM stood out regarding the other algorithms, the authors observed this after checking 4 metrics: accuracy, precision, recall and F1-score.

3.3 Approach: Mix

Celikkan and Bozoklar [Celikkan and Bozoklar 2019] also propose a GoF pattern recommendation approach. Their approach is also based on information retrieval, it makes recommendations from texts, cases and questions. To evaluate the approach, the authors performed tests with 120 industrial problems. Initially, recommendations were made without considering questions, only texts and cases. According to the authors, the results were as follows: for 65% of the problems, the expected pattern was one of the first 3 recommended patterns; for 76% of the problems, the expected pattern was one of the top 5 recommended patterns; for 86% of the problems, the expected pattern was one of the top 7 recommended patterns; and finally, when considering questions, the answers of more experienced developers improved the recommendations considerably.

Zhang et al. [Zhang et al. 2023] put together a catalogue based on a systematic review of software patterns for decentralized blockchain applications. The authors presented 40 patterns and highlighted the importance of using software patterns to improve the development process of decentralized applications, as they help developers reuse proven solutions and avoid common pitfalls. In another recent work, the authors describe a grey literature review on standards adopted in microservices [Rocha et al. 2023]. The authors present a catalogue of 18 standards on what has been applied from a technical point of view. However, catalogues serve as a guide but require knowledge of the problem and which standards meet the developer's needs.

This work has the distinction of recommending standards, not utilizing a catalogue, but adopting information retrieval to facilitate developers' work, reducing the impact of adopting a microservices architecture.

4 Development of a Microservices Patterns Recommender

4.1 Microservices Patterns Corpus

Before developing the MPR (Microservices Patterns Recommender), it was necessary to build the MPC (Microservices Patterns Corpus) which is essential for the MPR to make recommendations, as will be presented in detail in Subsection 4.2. With that in mind, to build the MPC, 2 sources of information were used, Richardson's popular book named "Microservices Patterns: With examples in Java" [Richardson 2018] and his website [Microservices 2022].

From these sources, 10 patterns were randomly selected to compose the MPC, namely: Decompose by Capability, Decompose by Subdomain, Remote Procedure Invocation, Circuit Breaker, Self Registration, Client-Side Discovery, 3rd Party Registration, Server-Side Discovery, Messaging, and Transactional Outbox. For each of these patterns, 13 pieces of information were collected, which are presented in Table 4. Thus, the MPC was built.

4.2 Microservices Patterns Recommender

MPR is a Web Service capable of receiving one design problem and recommending three microservices patterns that can solve it. The Endpoint */recommendations* is responsible for making recommendations and requires a Query Parameter *design_problem*. It is possible to consume the Endpoint through HTTP requests and its responses are returned in JSON. All these details can be seen in an OpenAPI Specification available in */docs*.

Information	Description
Id	Unique number used to identify the pattern
Link	Web address for viewing pattern information
Name	Word used to refer to the pattern
Group	Broad term that indicates the category of the pattern
Subgroup	Specific term indicating the subcategory of the pattern
Layer	Layer where the pattern applies
Context	Circumstance in which the pattern applies
Problem	Question that the pattern solves
Forces	Issues that must be addressed when solving a problem in a given context
Solution	How the pattern solves the problem
Resulting Context	Situation resulting from the application of the pattern that involves benefits, drawbacks, and issues
Issues	Important points that must be analyzed to use the pattern
Related Patterns	Other patterns that have some kind of relationship with the pattern

Table 4: Information collected for each pattern

Figure 2 provides an example illustrating the communication between a client and the MPR. In the example, the client requests the endpoint that recommends microservices patterns, indicating the HTTP verb "GET" and the design problem "Microservices A and B need to communicate.". In response, the client receives 3 recommendations, two of which are omitted to simplify the example response.

The technologies used to develop the MPR were: Python [Python 2022], FastAPI [FastAPI 2023], Pandas [Pandas 2023], NLTK (Natural Language Toolkit) [NLTK 2022], Scikit-Learn [Scikit-Learn 2022] and BeautifulSoup [Beautiful Soup 2023]. The Web Service contains seven classes as depicted in Figure 3 with their main attributes and methods. The responsibilities of these classes will be referred to in Sub-subsections 4.2.1, 4.2.2 and 4.2.3 which explain what steps are performed by the MPR to make recommendations.

It should be noted that the MPR makes recommendations through IR (Information Retrieval). Thus, the MPC is substantial for recommendations to be made, as the MPR will use it to retrieve the patterns that have the most similarities with a design problem. For MPR to make recommendations, three steps are performed by it: 1) Preprocess Documents, 2) Represent Documents and Query and 3) Match. Figure 4 exposes these steps.

4.2.1 Step 1: Preprocess Documents

The *Text* and *Dataset* classes are responsible for performing this Step. They pre-process the design problem and the MPC to prepare them for other steps and ensure more efficient recommendations. Preprocessing means performing the following sequential treatments: 1) Remove Null Values, 2) Change Letters to Lowercase, 3) Remove NaN (Not a Number) Values, 4) Remove Newlines, 5) Remove Tabs, 6) Remove Returns, 7) Remove HTML Tags, 8) Remove Punctuation, 9) Remove Extra Spaces, 10) Get Uncontracted Words,

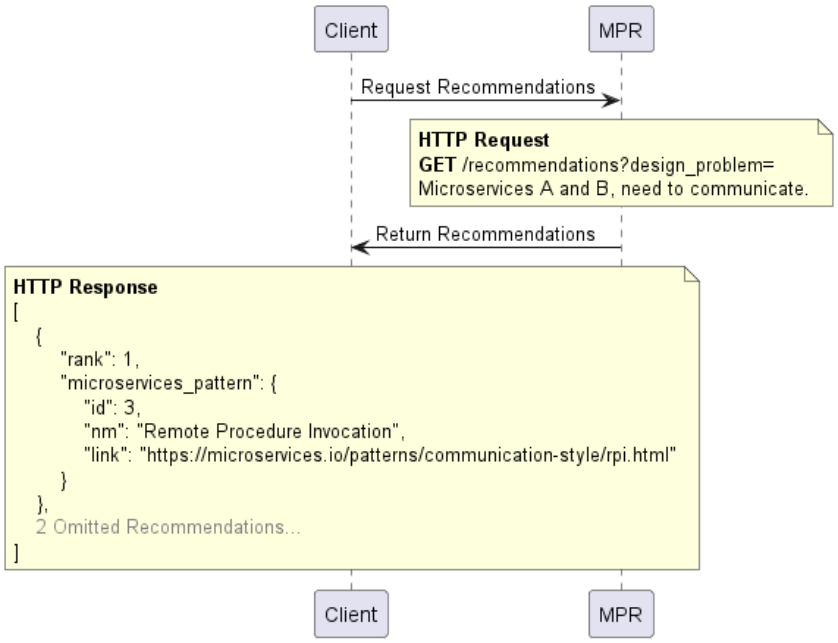

Figure 2: A Client communicating with MPR

11) Remove Stop Words, 12) Get Stems (Porter's Algorithm [Jones et al. 1997]). The *Text* class is responsible for performing these treatments with the *clean* method which makes multiple invocations of other methods. Table 5 exposes these treatments and the responsible methods.

Therefore, to pre-process the design problem, the *Text* class is used. To pre-process the MPC that is represented by the *Dataset* class, the *clean* method it contains is used. It should be noted that the *clean* method of *Dataset* performs the treatments through *Text*. Finally, to finalize this Step, the *Dataset* method *genr_soup* is executed to concatenate the preprocessed information from the microservices patterns (Context, Problem, Forces, Solution, Resulting Context and Related Patterns) and form an IS (Information Soup) for each pattern.

4.2.2 Step 2: Represent Documents and Query

With the design problem pre-processed and the information soups prepared, in this Step, MPR will weigh their terms, where for each term a weight will be assigned. This weight represents how important a *t* term is for a document *d* considering the other existing documents in MPC. It should be noted that, in this work, a term is a word and a document is a microservices pattern or a design problem. To calculate the value of this weight, MPR uses equation 5 which is called TF-IDF (Term Frequency-Inverse Document Frequency).

$$TF - IDF(t, d) = TF(t, d) \cdot IDF(t) \tag{5}$$

The value of TF(t,d) is the number of times *t* appears in *d* divided by the number of words *d* contains. The value of IDF(t) is calculated by equation 6, where *n* is the

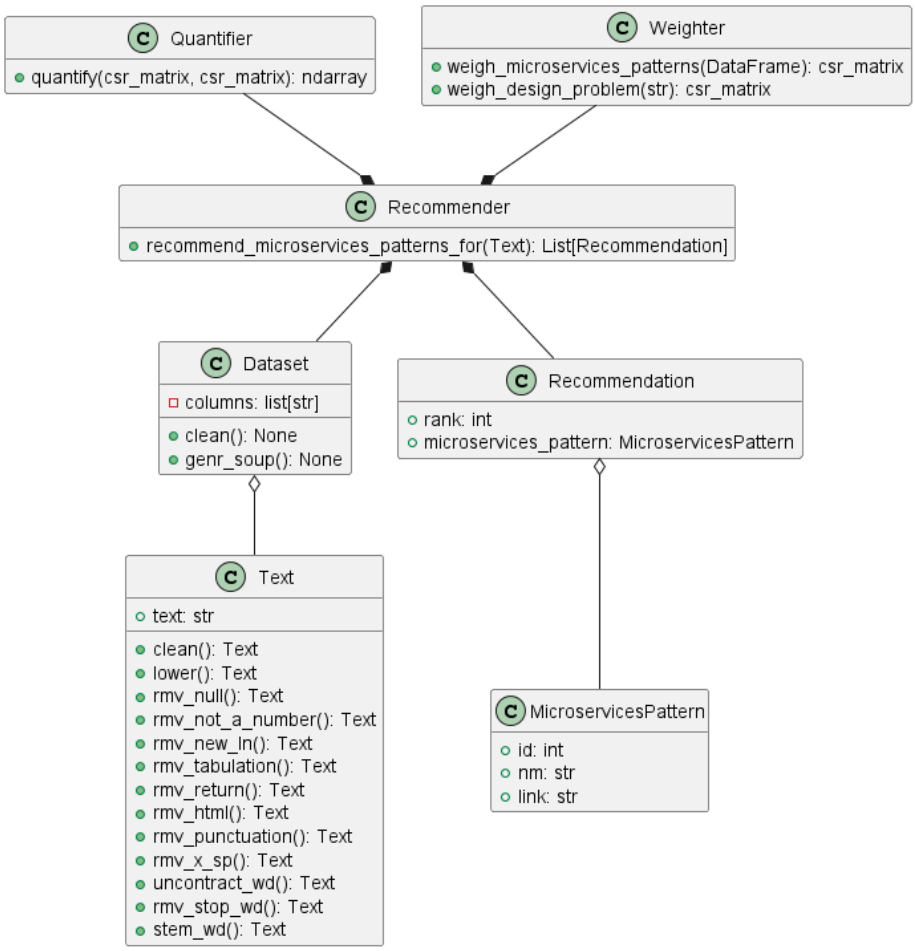

Figure 3: Class Diagram - MPR

number of documents that the MPC contains and DF(t) is the number of documents that contain *t*. The equations presented are used by Scikit-Learn to perform this Step. The class called *Weighter* uses Scikit-Learn in its two methods with meaningful names: *weigh_microservices_patterns* and *weigh_design_problem*.

$$IDF(t) = \log \frac{1+n}{1+DF(t)} + 1 \tag{6}$$

Before explaining the next step, it is important to exemplify this Step. Thus, assuming that MPC has 2 patterns with word communication and MPR received a design problem with 100 words, where word communication appears 3 times, given that the TF value is 0.03 and IDF is 1.56, the weight of word communication for the design problem is 0.04.

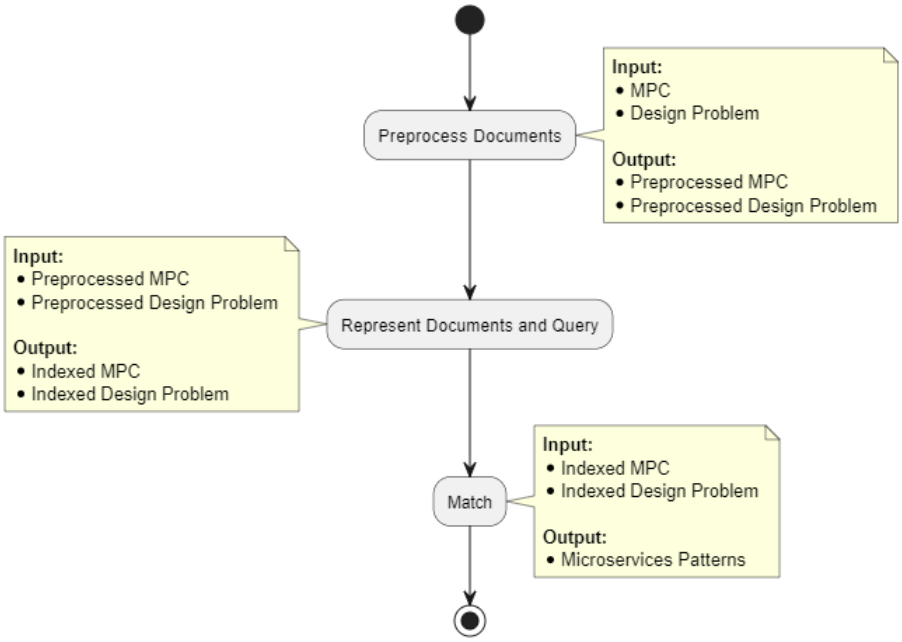

Figure 4: Steps performed for MPR to make recommendations

4.2.3 Step 3: Match

This is the last Step, in it the MPR will calculate how similar the design problem is to each microservices pattern, using a CS (Cosine Similarity) that is given by Equation 3, where V is the microservices pattern vector, W is the design problem vector and n is the size of the vectors. It should be noted that the vectors used in this Step are the outputs of the previous Step and are composed of the weights calculated with TF-IDF. Thus, the first 3 patterns, ordered in descending order by CS value, are those indicated by the MPR to solve the design problem. To perform this Step, MPR uses a class named *Quantifier* and its method *quantify*.

$$CS(V,W) = \frac{\sum_{i=1}^{n} V_i W_i}{\sqrt{\sum_{i=1}^{n} V_i^2}\sqrt{\sum_{i=1}^{n} W_i^2}} \tag{7}$$

4.3 Floc

Floc is a microservices patterns recommendation tool that provides a friendly UI (User Interface) for developers and students to get microservices pattern recommendations that help them solve design problems. The tool was developed in Java [Java 2023], using Spring [Spring 2023], Thymeleaf [Thymeleaf 2023] and PostgreSQL [PostgreSQL 2023] which are popular technologies. It allows a user to sign up, log in, register a new design problem, edit an existing design problem, delete an existing design problem and view details of an existing design problem.

Treatment	Method	Input (E.g.)	Output (E.g.)
Remove Null Values	rmv_null	Null	Empty String
Change Letters to Lowercase	lower	This is a TEXT	this is a text
Remove NaN (Not a Number)	rmv_not_a_number	NaN	Empty String
Remove Newlines	rmv_new_ln	This is a \n Text	This is a Text
Remove Tabs	rmv_tabulation	This is a \t Text	This is a Text
Remove Returns	rmv_return	This is a \r Text	This is a Text
Remove HTML Tags	rmv_html	This is a Text	This is a Text
Remove Punctuation	rmv_punctuation	This is a Text!!!	This is a Text
Remove Extra Spaces	rmv_x_sp	This is a Text	This is a Text
Get Uncontracted Words	uncontract_wd	It's hard	It is hard
Remove Stop Words	rmv_stop_wd	This is a communication pattern	communication pattern
Get Stems	stem_wd	Communication	Commun

Table 5: Treatments for Pre-Processing Texts

To store data handled by Floc, four tables were created, as shown in Figure 5. Table *microservices_pattern* receives a data preload for normalization and performance purposes. Thus, the existing microservices patterns in MPC are previously inserted in this table. It is worth mentioning that the tool interacts with MPR to obtain recommendations, as explained in Subsection 4.2.

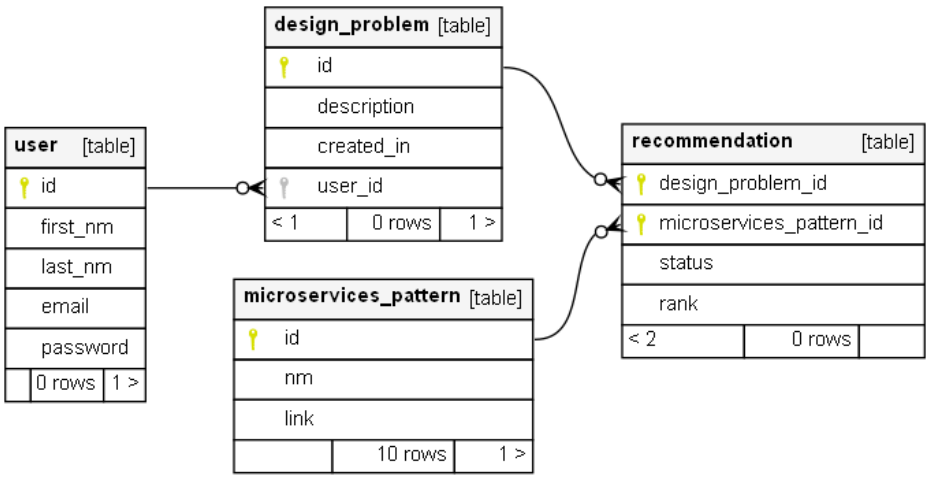

Figure 5: ERD (Entity Relationship Diagram) - Floc Database

When accessing the tool, the user can use two features, login or sign up. To sign up, the user must inform his/her first name, last name, email and password. To log in, it is necessary to provide an email and password. When registering, the user can log in.

When logging in, the user is redirected to the My Design Problems page, where

he/she can view his/her registered design problems or click on New Design Problem to go to the New Design Problem page, where it is possible to describe a new design problem. Figures 6 and 7 show, respectively, the mentioned pages. The New Design Problem page contains a single field form that receives a design problem. After reporting a design problem, the user can click Solve to register the reported design problem and obtain recommendations.

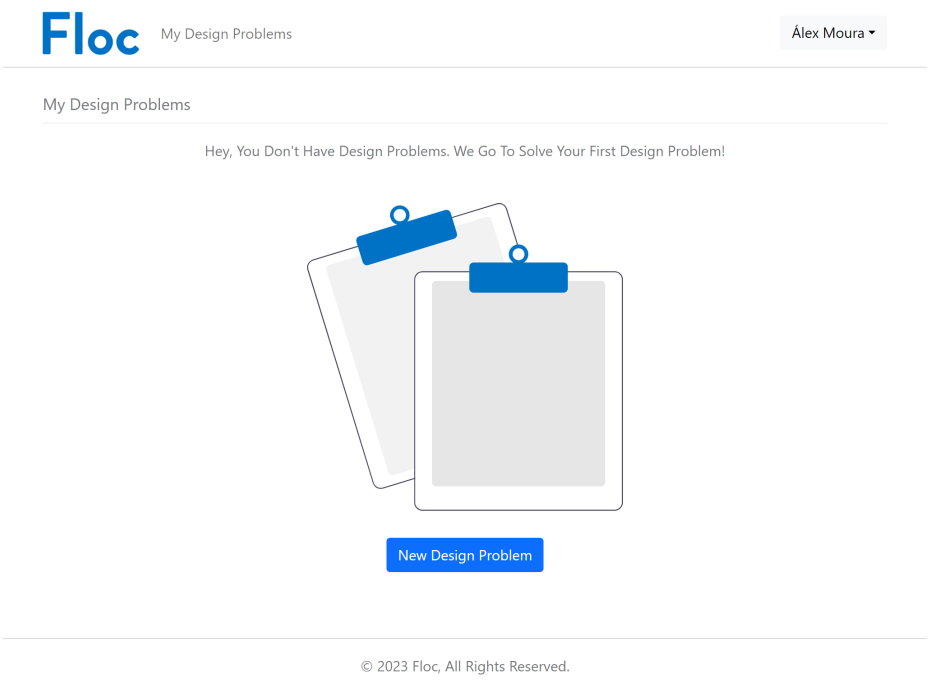

Figure 6: My Design Problems Page

After the user clicks Solve, he/she is redirected to the Design Problem Details page that presents data regarding the registered design problem: Description, Created In and Recommended Microservices Patterns. It is possible to view detailed information about a pattern by clicking the button which has a link icon, it redirects to the exact page on Richardson's Website which explains the pattern. The user can also inform if a recommended pattern solves the design problem, partially solves the design problem or does not solve the design problem. This is important to build a dataset that can be used in future research to make recommendations with classification algorithms, for example.

The Design Problem Details page also allows editing the design problem, by clicking the Edit button which redirects to the Edit Design Problem page which contains a single field form, where the design problem can be changed. If the user wants to delete the design problem, then he/she can simply click the Delete button in Design Problem Details and confirm the action. Figure 8 shows the Design Problem Details page.

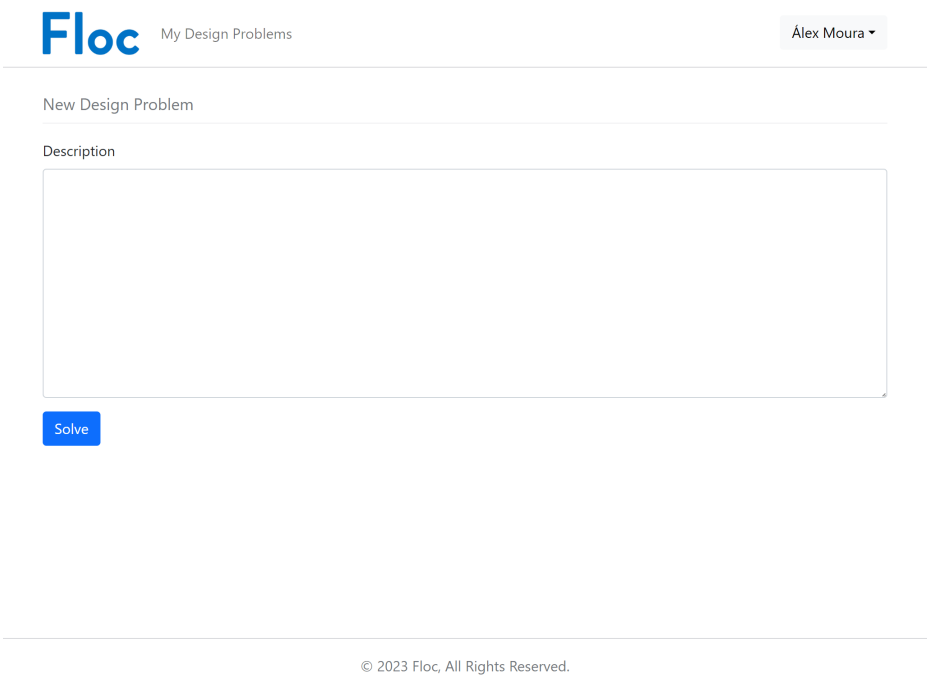

Figure 7: New Design Problem Page

4.4 Evaluation: Microservices Patterns Recommender

With the intention of testing and evaluating the MPR, a Collection of Toy Design Problems (CTDP) was created, where 10 problems were defined. These 10 problems are presented as follows:

- **Toy Design Problem 1:** A company that develops software systems, received a new demand, where it is necessary to develop a clinical management system. Because of this, the requirements analysts performed the requirements survey, and from the requirements, the software architects concluded that the microservices architecture would be the most suitable for the system. Then, systems analysts created a top-level domain model, identified command and query operations, and finally mapped business capabilities, with the help of requirements analysts. Now systems analysts want to define system services. These services must be independent, small and contextually limited;
- **Toy Design Problem 2:** OnShop is a monolithic software system that will be migrated to microservices. Professionals participating in this migration want to use Domain-Driven Design (DDD) concepts to decompose the system into services, as they are familiar with DDD;
- **Toy Design Problem 3:** The fictional company, XGO, is designing a microservices architecture for its main system and it needs to make the microservices communicate. This communication must be request/response-based and synchronous;

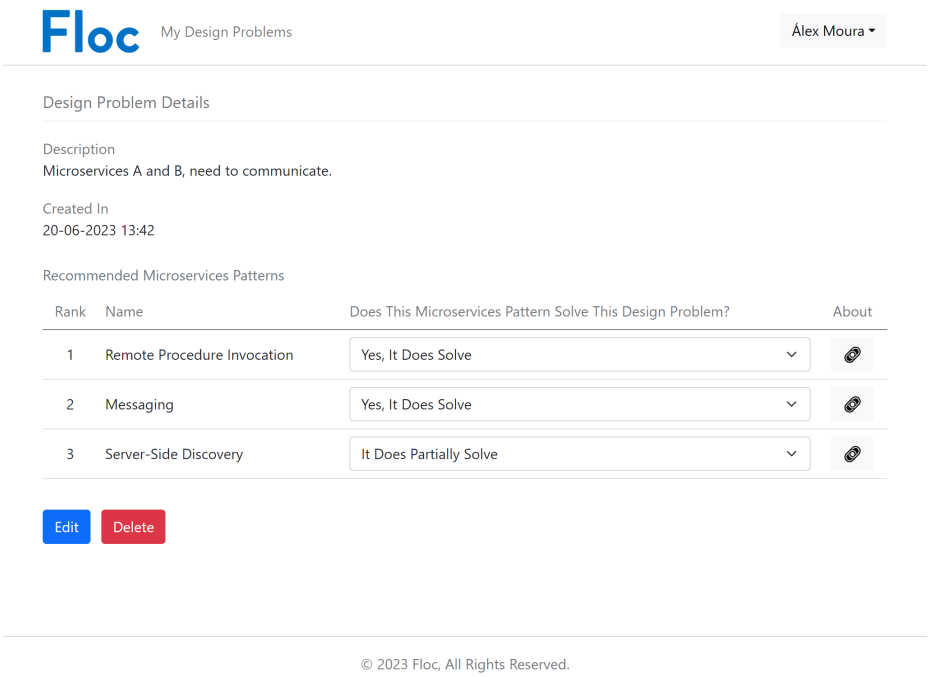

Figure 8: Design Problem Details Page

- **Toy Design Problem 4:** After making the services communicate through REST, it was observed that when the mobile app makes multiple requests to the order service, through an API gateway, and the order service is down or responding to the requests extremely slowly, the API gateway is unable to service new requests;
- **Toy Design Problem 5:** Instances of ordering, payment and other services need to make their network address available in some way, as customers need this information to make requests;
- **Toy Design Problem 6:** The ordering service is a client of the ticketing service, so there needs to be communication between them, but for that, the client needs to find out which instance of the ticket service should be called;
- **Toy Design Problem 7:** We want instances of the services not to be responsible for registering through the registry service as we do not want to hook them up. Therefore, we would like to transfer this task to a third party;
- **Toy Design Problem 8:** Clients need to communicate with the services, but there must be a server-side mechanism that figures out which instance of the services should serve the request;
- **Toy Design Problem 9:** The ordering service is responsible for creating a new order, for this it must communicate with the payment service, if the payment is approved, the ordering service must notify the ticket service that there is a new meal to prepare;

- **Toy Design Problem 10:** The order service must send a message to the payment service if the order is created successfully, as the message must be part of the transaction, in case of failure, the message must not be sent.

Each of these problems has been labelled with the microservices pattern that can solve it. Table 6 exposes each problem with its label. One can note that all microservices patterns that belong to the CMP were used as labels. An important point is that only one microservices' pattern was considered to solve a problem, in this case, the most appropriate pattern, however, it is worth highlighting that a problem can be solved by more than one microservices' pattern.

Design Problem	Microservices Pattern
1	Decompose by Business Capability
2	Decompose by Subdomain
3	Remote Procedure Invocation
4	Circuit Breaker
5	Self Registration
6	Client-Side Discovery
7	3rd Party Registration
8	Server-Side Discovery
9	Messaging
10	Transactional Outbox

Table 6: Each Toy Design Problem with the Microservices Pattern that Can Solve it.

The recommendation approach was evaluated through tests carried out with the problems present in the CTDP, where recommendations were requested for each of these problems. After obtaining the recommendations, they were analyzed considering Table 6 to verify the quality of the recommendation approach through the metrics presented in Subsection 2.3.4.

4.5 Evaluating Floc

To evaluate the tool, it was published so that tests could be carried out in 3 different companies. One of these companies offers integrated and innovative cyber security and infrastructure solutions. In 2022, this company was classified, by MSSP Alert [MSSP Alert 2023], as one of the 40 best security companies in the world.

Five experienced developers ran tests in which they asked for recommendations for microservices patterns to solve industrial design problems. Then, these developers evaluated, in the tool itself, each recommendation with one of these 3 response options: 1) Yes, It Does Solve; 2) It Does Partially Solve; or 3) It Does Not Solve. Table 7 shows the time of experience, in years, that each developer has with software development and systems based on microservices.

After testing, the developers involved were individually interviewed. A total of 9 questions were asked, which can be found in Table 8. The interviews were essential to collect feedback from each developer and information that could contribute to improving the tool.

ID	Experience in Software Development (Years)	Experience Microservices-Based Systems (Years)
D1	29	7
D2	23	3
D3	12	1
D4	12	10
D5	7	2

Table 7: Time of Experience of Developers Involved in the Tests.

Question	Description
Q1	In which contexts can the tool help?
Q2	How can the tool help developers?
Q3	What advantages can the tool provide in a software project?
Q4	What disadvantages can the tool provide in a software project?
Q5	How can a company benefit from adopting the tool?
Q6	What limitations does the tool present?
Q7	What can be improved in the tool?
Q8	Would you use the tool in a software project?
Q9	What difficulties did you find with the user interface when using the tool?

Table 8: Questions Asked in the Interviews.

5 Results

This Section presents the results obtained in tests carried out with the CTDP, industrial design problems and interviews carried out with developers who participated in the tests. Subsections 5.1, 5.2 and 5.3 present these results, respectively.

5.1 Tests with Toy Design Problems

The microservices patterns recommended for the design problems of the CTDP can be seen in Table 9. In total 30 microservices patterns were recommended, and 3 microservices patterns were recommended per design problem. The recommended microservices patterns are organized into 3 columns, “Recommendation 1”, “Recommendation 2” and “Recommendation 3”.

When checking if each design problem received for recommendation the microservices pattern that is capable of solving it, as shown in Table 9, 6 out of the 10 design problems received and only 4 did not receive. The 6 design problems that received were: 1, 2, 3, 6, 7 and 10. The 4 design problems that did not receive were: 4, 5, 8 and 9. The microservices pattern that can solve each design problem can be seen in Table 6.

Table 10 exposes, for each design problem, the following metrics: Precision, Recall and F1-Score. One can note that for the design problems solved (1, 2, 3, 6, 7 and 10), the values of these metrics are the same, as only 1 microservice pattern was considered as a label, as explained in Subsection 4.4. For the unsolved design problems (4, 5, 8 and 9), the values are also the same, as none of them received the label how recommendation.

DP	Recommendation 1	Recommendation 2	Recommendation 3
1	Decompose by Business Capability	Decompose by Subdomain	Server-Side Discovery
2	Decompose by Subdomain	Decompose by Business Capability	Remote Procedure Invocation
3	Remote Procedure Invocation	Messaging	Decompose by Subdomain
4	Remote Procedure Invocation	Client-Side Discovery	Server-Side Discovery
5	Server-Side Discovery	Remote Procedure Invocation	Client-Side Discovery
6	Remote Procedure Invocation	Client-Side Discovery	Server-Side Discovery
7	Self Registration	3rd Party Registration	Client-Side Discovery
8	Remote Procedure Invocation	Client-Side Discovery	Self Registration
9	Remote Procedure Invocation	Self Registration	3rd Party Registration
10	Transactional Outbox	Messaging	Circuit Breaker

Table 9: Recommendations per Design Problem (DP).

For the design problems solved, the values were Precision ($\approx$ 35%), Recall (100%) and F1-Score ($\approx$ 50%), while for the unsolved design problems, the values were Precision (0%), Recall (0%) and F1-Score not calculated.

Design Problem	Precision	Recall	F1-Score
1	$\approx$ 35%	100%	$\approx$ 50%
2	$\approx$ 35%	100%	$\approx$ 50%
3	$\approx$ 35%	100%	$\approx$ 50%
4	0%	0%	-
5	0%	0%	-
6	$\approx$ 35%	100%	$\approx$ 50%
7	$\approx$ 35%	100%	$\approx$ 50%
8	0%	0%	-
9	0%	0%	-
10	$\approx$ 35%	100%	$\approx$ 50%

Table 10: Metrics per Design Problem.

In this case, Precision is not that important, as only 1 microservices' pattern was considered as label for each design problem. In testing, the important thing is that at least 1 of the microservices patterns recommended for each design problem is the label shown in Table 6. Therefore, in this case, Recall is more important. The metrics

values for each design problem solved were good: Precision indicating that among the 3 recommended microservices patterns, there is 1 important; Recall indicating that all important microservices patterns have been recommended; and F1-Score indicating a reasonable performance.

One can note that 4 out of the 6 important microservices patterns were recommended first and 2 in second. In other words, this means that 40% of these patterns were recommended first and 20% second.

Out of 10 microservices patterns used as labels, 6 were recommended and 4 were not. In other words, the MPR covered/recommended 60% of these patterns. This indicates that the MPR was able to address 60% of the design problems and that there is room for improvement, as 40% of the design problems were not addressed. In general, the MPR presented good results.

5.2 Tests with Industrial Design Problems

As explained in Subsection 4.5, tests were performed in 3 different companies with industrial design problems, where 5 developers have requested recommendations for existing design problems, in certain software products, using Floc. After getting the recommendations, they labelled each recommended pattern with one of the following: Yes, It Does Solve; It Does Partially Solve; or It Does Not Solve. This was done on the page called Design Problem Details.

In total, eleven design problems were reported by the developers. Thus, 33 microservices pattern recommendations were made, given that for a design problem, 3 patterns are recommended, as explained in Subsection 4.2. Considering the labels already mentioned, 16 recommended patterns were labelled "Yes, It Does Solve", 8 were labelled "It Does Partially Solve" and 9 were labelled "It Does Not Solve".

Intending to present some examples of industrial design problems reported during the tests performed and provide an idea of how these problems were described, it is possible to view three of the eleven industrial design problems reported by the developers:

- **Industrial Design Problem 1**: In a microservices architecture, it is common for services to need to communicate with each other to fulfil requests from clients. However, if the communication between microservices is not optimized, it can lead to performance issues and slow down the entire system. For example, if each microservice makes separate HTTP requests to other microservices, it can result in a large number of network calls and slow down the overall response time;
- **Industrial Design Problem 2**: How to migrate a monolithic system that has only one database to microservices systems, with communication between the data;
- **Industrial Design Problem 3**: How to make the front-end communicate with several microservices and ensure security.

As depicted in Figure 9, only one pattern was not recommended, Circuit Breaker. All other patterns were recommended and it is important to highlight that the three most recommended patterns were, in descending order, Remote Procedure Invocation, Messaging and Decompose by Subdomain.

Data presented in this Subsection indicate that more than 70% of the recommended patterns helped deal with the problems. Although nine recommended patterns were labelled "It Does Not Solve", 90% of the problems received recommendations for

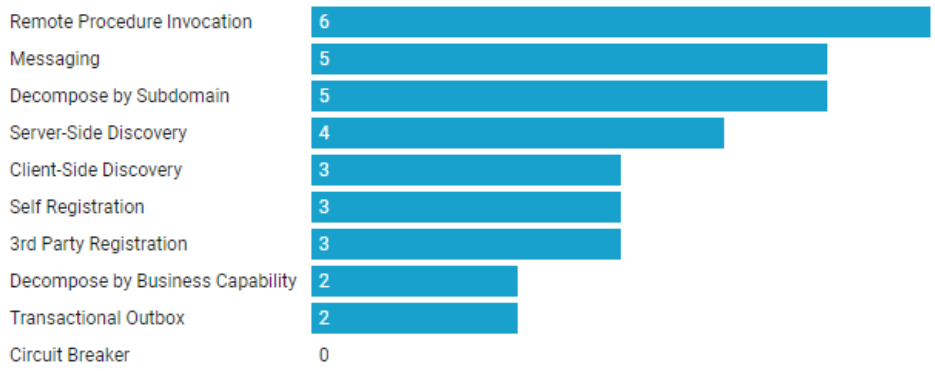

Figure 9: Distribution of Recommendations per Microservices Pattern.

important patterns that solve them. This means that the tool developed in this work is useful. Several different patterns were recommended, demonstrating that MPR is not addicted to specific patterns.

5.3 Interviews with Developers

After running the tests, the five developers who participated in the tests were individually interviewed. During the interviews, they answered nine questions related to the tool. The answers to these questions are presented throughout this Subsection. In general, the responses were positive and revealed improvements for future research.

About **Q1**, it was pointed out by D1 and D4 that the tool is useful in a context where it is necessary to develop new functionalities, as well as maintain existing functionalities. D1 also mentioned that the tool can help in contexts involving less experienced developers. D3, on the other hand, stated that the tool can be better used by more experienced developers. As for D2, he explained that the tool can help in contexts where problems need to be dealt with, including those involving several teams, where support is needed to decide which solution to use. D5 pointed out that the tool can help in planning the solutions that will be used in development.

Regarding **Q2**, the five interviewees stated that the tool can help developers find microservices patterns to solve problems reported by them. According to D1, this improves the development process of systems based on microservices architecture. In addition, D1, D2 and D4 pointed out that the tool can also help developers in continuous learning. Increased productivity was also mentioned by D1, D2 and D3.

About **Q3**, the five interviewees reported that the tool can offer agility in the search for patterns to solve problems present in software projects. According to D1 and D2, other advantages are the standardization of a source of knowledge, as well as the standardization of solutions.

Regarding **Q4**, D1 reported that given the ease provided by the tool in finding patterns to solve certain problems, developers, especially the less experienced ones, can become enchanted and become dependent on the use of patterns, even to solve problems that do not require patterns. Therefore, D1 concluded that it is important for developers to assess any problems and the real need to use patterns to solve them. D2 pointed out that if some pattern recommendations have a considerably long learning curve, it can be a

disadvantage, considering the existence of alternative solutions in other sources, with a smaller learning curve, and the short deadlines in software projects. D4 pointed out that if the software is already under development it can bring complexity to the developers' workflow. D3 and D5 did not mention disadvantages.

About **Q5**, D1 stated that the tool can be used to train developers in a shorter time, providing ease of understanding regarding microservices patterns. D1 also mentioned that the application of the correct standards generates agility and ensures that the microservices that are part of a software product meet the needs of the stakeholders, guaranteeing the quality of the product. D2 and D3 pointed out that the tool can be used to standardize solutions and support decision-making. D2 also stated that not having standardized solutions is a problem, taking into account the turnover of developers in a company. Because of this, D2 stated that the tool can improve developers' productivity and communication between them. D4 pointed out that the tool can help with architectural standardization, reducing risks, increasing development agility and improving product quality. D5 stated that the tool can help reduce planning time for software development.

Regarding **Q6**, D1 and D5 pointed out that the number of patterns that the tool knows is a limitation. D2 stated that not demonstrating practical examples and not indicating other developers' preferences for solving a certain type of problem are limitations. D3 pointed out, as a limitation, the absence of suggestions regarding how to organize folders and files related to a pattern. D4 pointed out that not explaining why each pattern was recommended is a limitation.

About **Q7**, D1 and D5 reported an improvement in the expansion of the MPC. D2 pointed, out as an improvement, the display of statistics to help the developer who seeks to solve a problem, to understand if the recommended pattern that he thinks of choosing, was the choice of other developers to solve problems similar to his. D3 declared, as an improvement, the presentation of suggestions regarding how to organize folders and files related to a pattern. D4 and D5 suggested that the tool explain the reason for the recommendations.

Regarding **Q8**, D1, D4 and D5 pointed out that they would use the tool in a software project. D2 also stated that he would use it, but initially on a small project, without involving multiple teams. D3 said he would also use it, including academically to teach students and develop scientific research.

About **Q9**, the last question, D1, D2, D4 and D5 stated that they had no difficulties with the user interface. D1 found it simple and D2 found it intuitive. D3 reported that he had difficulties understanding the information presented and suggested the inclusion of explanations.

6 Threats to validity

The validity of research is a crucial aspect of ensuring that the results are reliable. Thus, seeking to ensure the reliability of the study, it was necessary to validate factors related to the validity of the research. These are explained and presented in detail how we ensure that this did not suffer threats to its quality and integrity [Host et al. 2012] [Ampatzoglou et al. 2019].

6.1 Internal threat

Among the internal threats are the risks related to the reliability of the results within the study itself. Thus, seeking to guarantee that there is no bias, the execution was evaluated

by three researchers to ensure that there were no problems in its execution. Besides, during the first stage of the platform's execution, a presentation was made, guaranteeing the understanding of its use. After the execution of the study, interviews were conducted, and these had a triple evaluation, seeking to identify any bias in interpretation and maintaining rigour in data analysis and evaluation.

6.2 External validation

External validation refers to the generalization of results. However, according to Host [Host et al. 2012], one cannot generalize thinking about the universe. Therefore, based on related works, as well as on the careful assessment performed during the application of the study with experienced people and having the diversity of use, it generated solid results enabling us to understand how to apply the survey in other environments.

6.3 Construct validity

This validation is related to the way of capturing the phenomenon under study, thus, a pilot was conducted, seeking to verify the understanding of the study and the questionnaire by users. In addition, it was validated by two other researchers, seeking to make the result more reliable.

7 Discussion

The suggested tool was tested in two ways. First, toy problems were proposed by the researchers involved in this work. Second, considering the industry problems provided by one company that offers integrated and innovative solutions for security and cybernetic infrastructure, and two others specialized in financial services. The results showed that the tool was able to solve 60% of all toy design problems, indicating that it can provide valuable and accurate recommendations. Furthermore, tests with industrial problems revealed that over 70% of the recommended patterns helped address the problems at hand. Interviews were also carried out, which added to the results of this study, with developers from three different companies who participated in the tests.

Compared to the study published by Celikkan and Bozaklar [Celikkan and Bozoklar 2019], which obtained an average success rate of 65 per cent for three recommendations, our work was slightly better for individual recommendations, rising to 70 per cent.

We can highlight some differences between this study and the others. First, the focus on recommending microservice standards, according to Rocha et al. [Rocha et al. 2023], has received attention from professionals and academics. However, adopting a microservices architecture also brings challenges since some new standards and demands would not be necessary for a monolithic system. Hence, another difference we can highlight is that, generally, works that adopt recommendations or decision-making focus mainly on GoF standards, which, despite their importance, do not meet the demands of microservices. Like most of the work, this paper also proposes a recommendation approach based on information retrieval. Another point of difference is that several studies present catalogues of standards for microservices. However, these catalogues require knowledge on the part of the developer to facilitate their adoption.

The information retrieval approach adopted here, as it has been validated in the software industry, facilitates the work of developers, who, even if they have little experience

with microservices, receive an accurate recommendation when presented with a design problem. This makes it easier to adopt the standards on a day-to-day basis, which is useful in practice, as beginner and experienced developers may face difficulties when selecting design patterns, given the substantial number of design patterns described in the past decades. Therefore, this work seeks to offer a solution to this difficulty, which is important because it can help to prevent microservices-based systems from being discontinued or re-engineered. Moreover, information retrieval can be easily extended to new standards, i.e., new standards can be added to the catalogue for microservices as they emerge and are used in the future.

In this work, it makes no sense to propose a recommendation approach based on text classification, since no dataset was found to train a classification model for problems related to systems based on microservices. According to Uysal [Uysal 2016], the effectiveness of classification-based systems depends on an extensive and representative dataset. Therefore, the recommendation approach that this work proposes is based on information retrieval. This work also differs from the works presented by proposing a Web Service capable of making recommendations and a tool integrated with this Web Service that offers a user interface.

8 Conclusion and Future Work

Microservices architecture is a distributed application in which all its modules are microservices. There are advantages to using microservice-based architectures for software development, such as a small code base, gradual transitions to new microservice versions, which leads to continuous integration and easier software maintenance, and freedom to choose resources for implementing each microservice. This model has led to the emergence of new standards, mainly linked to the architectural structure of the microservice application and how they communicate, and these standards are the subject of this study.

This article addressed the objective of providing software developers with a tool for selecting appropriate microservices patterns to solve problems in microservices-based systems. The proposed recommendation tool is based on Information Retrieval (IR) and it demonstrated its effectiveness through testing with both toy and industrial problems.

Tests with industrial problems also demonstrated the tool's versatility and ability to recommend a diverse range of patterns. While a small number of recommended patterns were labelled as "Does not solve", it is noteworthy that 90% of the problems still received recommendations for important patterns that could help to solve them. This highlights the usefulness of the tool, as developers can receive useful recommendations that will help in the process of developing and maintaining microservices-based systems. The tool's ability to recommend various patterns without bias towards specific options reinforces its adaptability and applicability to a wide range of scenarios.

Additionally, the feedback obtained through individual interviews with the five developers who participated in the tests further emphasized the positive reception of the tool. Their responses provided valuable insights for future work of evolution and enhancement of the recommendation tool. This feedback not only validates the effectiveness of the tool but also serves as a valuable resource for further improvements and refinements. The constructive feedback and positive evaluations from the developers underscore the potential of the recommendation tool to assist software developers in selecting microservices patterns.

References

[Abgaz et al,. 2023] Abgaz, Yalemisew and McCarren, Andrew and Elger, Peter and Solan, David and Lapuz, Neil and Bivol, Marin and Jackson, Glenn and Yilmaz, Murat and Buckley, Jim and Clarke, Paul, "Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review," in IEEE Transactions on Software Engineering, vol. 49, no. 8, pp. 4213-4242, Aug. 2023.

[Alexander 1977] Alexander, C.: "A Pattern Language: Towns, Buildings, Construction". (Oxford University Press, 1977).

[Ampatzoglou et al. 2019] Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M. & Chatzigeorgiou, A.: "Identifying, Categorizing and Mitigating Threats to Validity in Software Engineering Secondary Studies". Information And Software Technology. 106 pp. 201-230 (2019).

[Balalaie et al. 2016] Balalaie, A., Heydarnoori, A. & Jamshidi, P.: "Migrating to Cloud-native Architectures using Microservices: An Experience Report". Advances In Service-Oriented And Cloud Computing: Workshops Of ESOCC 2015, Taormina, Italy, September 15-17, 2015, Revised Selected Papers 4. pp. 201-215 (2016).

[Balalaie et al. 2018] Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. & Lynn, T.: "Microservices Migration Patterns". Software: Practice And Experience. 48, 2019-2042 (2018).

[Baresi et al. 2017] Baresi, L., Garriga, M. & De Renzis, A.: "Microservices Identification through Interface Analysis". Service-Oriented And Cloud Computing: 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings 6. pp. 19-33 (2017).

[Beautiful Soup 2023] Beautiful Soup Home Page. https://beautiful-soup-4.readthedocs.io. Last accessed on Apr 13, 2023.

[Celikkan and Bozoklar 2019] Celikkan, U. & Bozoklar, D.: "A Consolidated Approach for Design Pattern Recommendation". 2019 4th International Conference On Computer Science And Engineering (UBMK). pp. 1-6 (2019).

[Ceri and et al. 2013] Ceri, S., Bozzon, A., Brambilla, M., Della Valle, E., Fraternali, P., Quarteroni, S., Ceri, S., Bozzon, A., Brambilla, M., Della Valle, E. & Others.: "The Information Retrieval Process". Web Information Retrieval. pp. 13-26 (2013).

[Cerny et al. 2018] Cerny, T., Donahoo, M. & Trnka, M.: "Contextual Understanding of Microservice Architecture: Current and Future Directions". ACM SIGAPP Applied Computing Review. 17, 29-45 (2018).

[Chen et al. 2017] Chen, R., Li, S. & Li, Z.: "From Monolith to Microservices: A Dataflow-Driven Approach". 2017 24th Asia-Pacific Software Engineering Conference (APSEC). pp. 466-475 (2017).

[Croft et al. 2010] Croft, W., Metzler, D. & Strohman, T.: "Search Engines: Information Retrieval in Practice". (Addison-Wesley Reading, 2010).

[Douglass 2003] Douglass, B.: "Real-time design patterns: robust scalable architecture for real-time systems". (Addison-Wesley Professional, 2003).

[Dragoni et al. 2017] Dragoni, N., Giallorenzo, S., Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R. & Safina, L.: "Microservices: Yesterday, Today, and Tomorrow". Present And Ulterior Software Engineering. pp. 195-216 (2017).

[FastAPI 2023] FastAPI Home Page. https://fastapi.tiangolo.com. Last accessed on Apr 13, 2023.

[França and Soares 2015] França, J. & Soares, M.: "SOAQM: Quality Model for SOA Applications based on ISO 25010". ICEIS (2). pp. 60-70 (2015).

[Fritzsch et al. 2019] Fritzsch, J., Bogner, J., Zimmermann, A. & Wagner, S.: "From Monolith to Microservices: A Classification of Refactoring Approaches". Software Engineering Aspects Of Continuous Development And New Paradigms Of Software Production And Deployment: First International Workshop, DEVOPS 2018, Chateau De Villebrumier, France, March 5-6, 2018, Revised Selected Papers 1. pp. 128-141 (2019).

[Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R. & Vlissides, J.: "Design Patterns: Elements of Reusable Object-Oriented Software". (Pearson Deutschland GmbH, 1995).

[Hambarde and Proenca 2023] Hambarde, K. & Proenca, H.: "Information Retrieval: Recent Advances and Beyond". IEEE Access. 11 pp. 76581-76604 (2023).

[Hamdy and Elsayed 2018] Hamdy, A. & Elsayed, M.: "Topic Modelling for Automatic Selection of Software Design Patterns". Proceedings Of The International Conference On Geoinformatics And Data Analysis. pp. 41-46 (2018).

[Host et al. 2012] Host, M., Rainer, A., Runeson, P. & Regnell, B.: "Case Study Research in Software Engineering: Guidelines and Examples". (John Wiley & Sons, 2012).

[Hussain et al. 2019] Hussain, S., Keung, J., Sohail, M., Khan, A., Ilahi, M., Ahmad, G., Mufti, M. & Noor, M.: "A Methodology to Rank the Design Patterns on the Base of Text Relevancy". Soft Computing. 23, 13433-13448 (2019).

[Ibrihich et al. 2022] Ibrihich, S., Oussous, A., Ibrihich, O. & Esghir, M.: "A Review on Recent Research in Information Retrieval". Procedia Computer Science. 201 pp. 777-782 (2022).

[Java 2023] Java Home Page. https://www.java.com. Last accessed on Apr 13, 2023.

[Jones et al. 1997] Jones, K., Willett, P. & Others: "Readings in Information Retrieval". (Morgan Kaufmann, 1997).

[Koschel et al. 2017] Koschel, A., Astrova, I. & Dötterl, J.: "Making the Move to Microservice Architecture". 2017 International Conference On Information Society (i-Society). pp. 74-79 (2017).

[Landay and Hong 2003] Landay, J., Hong, J.: "The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience". (Addison-Wesley Professional, 2003).

[Mazlami et al. 2017] Mazlami, G., Cito, J. & Leitner, P.: "Extraction of microservices from monolithic software architectures". 2017 IEEE International Conference On Web Services (ICWS). pp. 524-531 (2017).

[Microservices 2022] Microservices Home Page. https://microservices.io. Last accessed on Mar 16, 2022.

[Moura 2023] Moura, A.S.: "Recommendation of Microservices Patterns Through Information Retrieval". Master Thesis. Federal University of Sergipe, 2023.

[MSSP Alert 2023] MSSP Alert Top 250 2022 Page. https://www.msspalert.com/top250/list-2022/22. Last accessed on Apr 21, 2023.

[Niknejad et al. 2018] Niknejad, N., Prasetyo, Y., Ghani, I. & Fajrillah, A.: "Service-Oriented Architecture Adoption: A Systematic Review". International Journal Of Integrated Engineering. 10 (2018).

[NLTK 2022] NLTK Home Page. https://www.nltk.org. Last accessed on Mar 16, 2022.

[Pandas 2023] Pandas Home Page. https://pandas.pydata.org. Last accessed on Apr 13, 2023.

[PostgreSQL 2023] PostgreSQL Home Page. https://www.postgresql.org. Last accessed on Apr 13, 2023.

[Python 2022] Python Home Page. https://www.python.org. Last accessed on Mar 16, 2022.

[Rahmati et al. 2019] Rahmati, R., Rasoolzadegan, A. & Dehkordy, D.: "An Automated Method for Selecting GoF Design Patterns". 2019 9th International Conference On Computer And Knowledge Engineering (ICCKE). pp. 345-350 (2019).

[Richardson 2018] Richardson, C.: "Microservices Patterns: with Examples in Java". (Simon, 2018).

[Rocha et al. 2023] Rocha, F. G.; Soares, M. S.; Rodriguez, G. Patterns in Microservices-based Development: A Grey Literature Review. In: Congresso Ibero-Americano em Engenharia de Software (CIBSE), 26. 2023, Montevideo, Uruguai. p. 61-76.

[Roshdi and Roohparvar 2015] Roshdi, A. & Roohparvar, A.: "Information Retrieval Techniques and Applications". International Journal Of Computer Networks And Communications Security. 3, 373-377 (2015).

[Saini et al. 2014] Saini, B., Singh, V. & Kumar, S. Information Retrieval Models and Searching Methodologies: Survey. Information Retrieval. **1** (2014).

[Sanyawong and Nantajeewarawat 2014] Sanyawong, N. & Nantajeewarawat, E.: "Design Pattern Recommendation Based-on a Pattern Usage Hierarchy". 2014 International Computer Science And Engineering Conference (ICSEC). pp. 134-139 (2014).

[Sanyawong and Nantajeewarawat 2015] Sanyawong, N. & Nantajeewarawat, E.: "Design Pattern Recommendation: A Text Classification Approach". (2015).

[Schumacher et al. 2013] Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F. & Sommerlad, P.: "Security Patterns: Integrating Security and Systems Engineering". (John Wiley & Sons, 2013).

[Scikit-Learn 2022] Scikit-Learn Home Page. https://scikit-learn.org. Last accessed on Mar 16, 2022.

[Silva-Rodríguez et al. 2020] Silva-Rodríguez, V., Nava-Muñoz, S., Castro, L., Martínez-Pérez, F., Pérez-González, H. & Torres-Reyes, F.: "Classifying Design-Level Requirements Using Machine Learning for a Recommender of Interaction Design Patterns". IET Software. 14, 544-552 (2020).

[Soares and França 2016] Soares, M. & França, J.: "Characterization of the Application of Service-Oriented Design Principles in Practice: A Systematic Literature Review". Journal of Software. 11, 403-418 (2016).

[Steinmetz et. al., 2018] Charles Steinmetz, Achim Rettberg, Fabíola Gonçalves C. Ribeiro, Greyce Schroeder, Michel S. Soares, Carlos E. Pereira. "Using Ontology and Standard Middleware for Integrating IoT based in the Industry 4.0". IFAC-PapersOnLine, Volume 51, Issue 10, Pages 169-174 (2018).

[Sousa et al. 2017] Sousa, L., Oliveira, R., Garcia, A., Lee, J., Conte, T., Oizumi, W., Mello, R., Lopes, A., Valentim, N., Oliveira, E. & Others: "How do Software Developers Identify Design Problems? A Qualitative Analysis". Proceedings Of The XXXI Brazilian Symposium On Software Engineering. pp. 54-63 (2017).

[Sousa et al. 2018] Sousa, L., Oliveira, A., Oizumi, W., Barbosa, S., Garcia, A., Lee, J., Kalinowski, M., Mello, R., Fonseca, B., Oliveira, R. & Others: "Identifying Design Problems in the Source Code: A Grounded Theory". Proceedings Of The 40th International Conference On Software Engineering. pp. 921-931 (2018).

[Spring 2023] Spring Home Page. https://spring.io. Last accessed on Apr 13, 2023.

[Taibi et al. 2017] Taibi, D., Lenarduzzi, V., Pahl, C. & Janes, A.: "Microservices in Agile Software Development: A Workshop-Based Study into Issues, Advantages, and Disadvantages". Proceedings Of The XP2017 Scientific Workshops. pp. 1-5 (2017).

[Thönes 2015] Thönes, J.: "Microservices". IEEE Software. 32, 116-116 (2015).

[Thymeleaf 2023] Thymeleaf Home Page. https://www.thymeleaf.org. Last accessed on Apr 13, 2023.

[Uysal 2016] Uysal, A.: "An Improved Global Feature Selection Scheme for Text Classification". Expert Systems With Applications. 43 pp. 82-92 (2016).

[Yugopuspito et al. 2017] Yugopuspito, P., Panduwinata, F. & Sutrisno, S.: "Microservices Architecture: Case on the Migration of Reservation-Based Parking System". 2017 IEEE 17th International Conference On Communication Technology (ICCT). pp. 1827-1831 (2017).

[Zhang et al. 2023] Zhang, Peng; Kelley, Adair; Schmidt, Douglas C; White, Jules: "Design Pattern Recommendations for Building Decentralized Healthcare Applications". Frontiers in Blockchain. (Frontiers, 2023).