

UNIVERSIDADE FEDERAL DE SERGIPE  
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# **Recommendation of Microservices Patterns Through Information Retrieval**

Dissertação de Mestrado

Álex dos Santos Moura



São Cristóvão – Sergipe

2023

UNIVERSIDADE FEDERAL DE SERGIPE  
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Álex dos Santos Moura

## **Recommendation of Microservices Patterns Through Information Retrieval**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Orientador(a): Michel dos Santos Soares

Coorientador(a): Fabio Gomes Rocha

São Cristóvão – Sergipe

2023

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL  
UNIVERSIDADE FEDERAL DE SERGIPE

M929r           Moura, Álex dos Santos  
                  Recommendation of microservices patterns through  
information retrieval / Álex dos Santos Moura ; orientador Michel  
dos Santos Soares ; coorientador Fábio Gomes Rocha. – São  
Cristóvão, SE, 2023.  
                  81 f. : il.

                  Dissertação (mestrado em Ciência da Computação) –  
Universidade Federal de Sergipe, 2023.

                  1. Computação. 2. Arquitetura orientada a serviços  
(Computador). 3. Recuperação da informação. I. Soares, Michel  
dos Santos, orient. II. Rocha, Fabio Gomes, coorient. III. Título.

                  CDU 004.2:025.4.036



UNIVERSIDADE FEDERAL DE SERGIPE  
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA  
COORDENAÇÃO DE PÓS-GRADUAÇÃO  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**Ata da Sessão Solene de Defesa da Dissertação do  
Curso de Mestrado em Ciência da Computação-UFS.  
Candidato: ALEX DOS SANTOS MOURA**

Em 01 dias do mês de dezembro do ano de dois mil e vinte três, com início às 09h, realizou-se na Sala de Seminários do PROCC da Universidade Federal de Sergipe, na Cidade Universitária Prof. José Aloísio de Campos, a Sessão Pública de Defesa de Dissertação de Mestrado do candidato **Álex dos Santos Moura**, que desenvolveu o trabalho intitulado: **“Recommendation of Microservices Patterns Through Information Retrieval”**, sob a orientação do Prof. Dr. Michel dos Santos Soares e coorientação do Prof. Dr. Fábio Gomes Rocha. A Sessão foi presidida pelo Prof. Dr. Michel dos Santos Soares (PROCC/UFS), que após a apresentação da dissertação passou a palavra aos outros membros da Banca Examinadora, Prof. Dr. Marcelo de Almeida Maia (UFU) e, em seguida, o Prof. Dr. Rubens de Souza Matos Junior (Procc/UFS) e o Prof. Dr. Fábio Gomes Rocha (Procc/UFS). Após as discussões, a Banca Examinadora reuniu-se e considerou o mestrando (a ) APROVADO. Atendidas as exigências da Instrução Normativa 05/2019/PROCC, do Regimento Interno do PROCC (Resolução 67/2014/CONEP), e da Resolução no 04/2021/CONEP que regulamentam a Apresentação e Defesa de Dissertação, e nada mais havendo a tratar, a Banca Examinadora elaborou esta Ata que será assinada pelos seus membros e pelo mestrando.

Cidade Universitária “Prof. José Aloísio de Campos”, 01 de dezembro de 2023.



Documento assinado digitalmente  
**MICHEL DOS SANTOS SOARES**  
Data: 01/12/2023 14:26:00-0300  
Verifique em <https://validar.iti.gov.br>

**Prof. Dr. Michel dos Santos Soares**  
**(PROCC/UFS)**  
**Presidente**



Documento assinado digitalmente  
**RUBENS DE SOUZA MATOS JUNIOR**  
Data: 07/12/2023 13:35:11-0300  
Verifique em <https://validar.iti.gov.br>

**Prof. Dr. Rubens de Souza Matos Junior**  
**(PROCC/UFS)**  
**Examinador Interno**



Documento assinado digitalmente  
**MARCELO DE ALMEIDA MAIA**  
Data: 01/12/2023 14:57:35-0300  
Verifique em <https://validar.iti.gov.br>

**Prof. Dr. Marcelo de Almeida Maia**  
**(UFU)**  
**Examinador Externo**



Documento assinado digitalmente  
**FABIO GOMES ROCHA**  
Data: 12/12/2023 10:44:24-0300  
Verifique em <https://validar.iti.gov.br>

**Prof. Dr. Fábio Gomes Rocha**  
**(Procc/UFS)**  
**Coorientador**



**ALEX DOS SANTOS MOURA**  
**Candidato**

*Este trabalho é dedicado às crianças adultas que,  
quando pequenas, sonharam em se tornar cientistas.*

# Acknowledgements

Em primeiro lugar, quero agradecer a Deus por me dar forças, coragem e não me deixar desistir. Quero agradecer aos meus pais, Aldevan e Edilma, e ao meu irmão, Enderson, por me auxiliarem e me apoiarem em diversos momentos. A minha noiva, Danielle, por me incentivar a ingressar no curso de mestrado em Ciência da Computação, pelas suas orientações, palavras positivas e por me motivar em momentos que pensei em desistir.

Aos meus professores e orientadores, Fabio e Michel, por todo auxílio e motivação. Sou muito grato pelas valiosas orientações, por compartilharem os seus conhecimentos e pela confiança que depositaram em mim. As instruções que os dois me deram, foram sem dúvidas, responsáveis por me fazer desenvolver uma grande pesquisa e chegar até aqui.

Quero agradecer também aos demais professores que de algum modo me ensinaram algo. Aos colegas de curso que pude conhecer em determinadas disciplinas e que de alguma forma me auxiliaram e aos colegas de trabalho que sempre compreenderam as minhas ausências para participar das aulas ou para desempenhar tarefas relacionadas ao curso. Enfim, quero agradecer imensamente a todos que de alguma maneira contribuíram com a minha jornada.

*“Temos que continuar aprendendo.  
Temos que estar abertos.  
E temos que estar prontos para espalhar nosso  
conhecimento a fim de chegar a uma  
compreensão mais elevada da realidade.”  
(Thich Nhat Hanh)*

# Abstract

O desenvolvimento de software envolve inerentemente a solução de problemas de projeto recorrentes, que podem afetar negativamente os atributos de qualidade de um sistema. Esses problemas são comumente resolvidos por meio da aplicação de padrões de projeto – soluções comprovadas e forjadas por desenvolvedores experientes. A arquitetura de microsserviços, uma variante da Arquitetura Orientada a Serviços, está em ascensão, com gigantes como Amazon, eBay e Netflix adotando essa arquitetura. Neste estilo de arquitetura, os sistemas são compostos por microsserviços que se comunicam por meio de mensagens. Cada microsserviço tem uma responsabilidade específica e é implantado, dimensionado e testado de forma independente. Os padrões de projeto usados para resolver problemas de projeto presentes em sistemas baseados em microsserviços são chamados de padrões de microsserviços. Resolver problemas de projeto é teoricamente simples, o desenvolvedor só precisa selecionar padrões de projeto e aplicá-los. Porém, na prática, tanto desenvolvedores iniciantes quanto experientes têm dificuldade em selecionar padrões de projeto, as razões para essa dificuldade são o número substancial de padrões de projeto e o conhecimento limitado. Assim, o objetivo geral deste trabalho é fornecer uma maneira de ajudar desenvolvedores a selecionar o padrão de microsserviços correto para resolver um determinado problema de projeto. Este trabalho foi desenvolvido usando a metodologia de pesquisa denominada Design Science Research, que propõe o desenvolvimento de artefatos de Tecnologia da Informação para solucionar problemas do mundo real. Inicialmente, foi proposta uma abordagem de recomendação baseada em Recuperação de Informação para fazer recomendações de padrões de microsserviços, onde o desenvolvedor pode relatar um problema de projeto, em linguagem natural (texto), e receber recomendações de padrões de microsserviços que possam resolver o problema de projeto relatado. No geral, os testes utilizando um conjunto de problemas de projeto mostraram que a abordagem de recomendação foi capaz de resolver 60% dos problemas de projeto e que há espaço para melhorias, uma vez que 40% dos problemas de projeto não foram resolvidos. Essa abordagem foi então usada em uma ferramenta chamada Floc. Essa ferramenta possui uma interface de usuário amigável onde o desenvolvedor pode gerenciar problemas de projeto e obter recomendações de padrões de microsserviços. A ferramenta foi avaliada utilizando problemas de projeto em uma empresa que atua na área de Segurança de Software e demonstrou resultados promissores. Entrevistas realizadas com desenvolvedores da indústria corroboraram a eficácia e praticidade da ferramenta de recomendação.

**Palavras-chave:** Microsserviços. Recomendação. Padrões de Projeto. Recuperação de Informação. Padrões de Microsserviços.



# Abstract

Software development inherently involves solving recurring design problems, which can negatively affect the quality attributes of a system. These problems are commonly solved through the application of design patterns – proven solutions forged by experienced developers. Microservices architecture, a variant of Service-Oriented Architecture, is on the rise, given that giants like Amazon, eBay and Netflix are adopting this architecture. In this architectural style, systems are composed of microservices that communicate through messages. Each microservice has a specific responsibility and is deployed, scaled, and tested independently. Design patterns used to solve design problems present in microservices-based systems are called microservices patterns. Solving design problems is theoretically simple, the developer only needs to select design patterns and apply them. However, in practice, both beginners and experienced developers have difficulty selecting design patterns, the reasons for this difficulty are the substantial number of design patterns and limited knowledge. Thus, the overall goal of this work is to provide a way to help developers select the right microservices pattern to solve a given design problem. This work was developed by using the research methodology named Design Science Research, which proposes the development of Information Technology artifacts to solve real-world problems. Initially, a recommendation approach based on Information Retrieval was proposed to make recommendations for microservices patterns, where the developer can report a design problem, in natural language (text), and receive recommendations for microservices patterns that can solve the design problem reported. Overall, testing using a set of toy design problems showed that the recommendation approach was able to solve 60% of design problems and that there is room for improvement as 40% of design problems were not solved. This approach was then used in a tool called Floc. This tool has a friendly user interface where the developer can manage design problems and get microservices pattern recommendations for them. The tool was evaluated using industrial design problems and demonstrated promising results. Furthermore, interviews with industry developers corroborated the effectiveness and practicality of the recommendation tool.

**Keywords:** Microservices. Recommendation. Design Patterns. Information Retrieval. Microservices Patterns.

# List of Figures

Figure 1 – Steps Performed by an Information Retrieval System. . . . .	23
Figure 2 – CMP Screenshot. . . . .	29
Figure 3 – Microservices Patterns per Group. . . . .	29
Figure 4 – Communication Microservices Patterns per Subgroup. . . . .	30
Figure 5 – Recommendation Approach: Steps. . . . .	31
Figure 6 – Class Diagram: Recommendation Approach. . . . .	37
Figure 7 – Sequence Diagram: Interaction Between the Classes. . . . .	39
Figure 8 – Solved and Unsolved Design Problems. . . . .	42
Figure 9 – Distribution of Important Recommendations per Rank. . . . .	43
Figure 10 – Coverage. . . . .	44
Figure 11 – Floc: Components. . . . .	46
Figure 12 – MPR OpenAPI Specification. . . . .	47
Figure 13 – Sequence Diagram: Communication Between a Client and the MPR. . . . .	48
Figure 14 – Database: Entity Relationship Diagram (ERD). . . . .	48
Figure 15 – Sequence Diagram: Interaction Between the Layers. . . . .	51
Figure 16 – Login Page. . . . .	52
Figure 17 – Sign Up Page. . . . .	52
Figure 18 – My Design Problems Page. . . . .	53
Figure 19 – New Design Problem Page. . . . .	54
Figure 20 – Design Problem Details Page. . . . .	55
Figure 21 – Edit Design Problem Page. . . . .	56
Figure 22 – Floc: Deployment Diagram. . . . .	56
Figure 23 – Class Diagram: Login. . . . .	57
Figure 24 – Class Diagram: Sign Up. . . . .	59
Figure 25 – Class Diagram: List of Design Problems. . . . .	60
Figure 26 – Class Diagram: Solve Design Problem. . . . .	61
Figure 27 – Class Diagram: View Design Problem Details. . . . .	62
Figure 28 – Class Diagram: Edit Design Problem. . . . .	63
Figure 29 – Class Diagram: Delete Design Problem. . . . .	65
Figure 30 – Class Diagram: Feedback for a Recommended Microservices Pattern. . . . .	67
Figure 31 – Distribution of Recommended Microservices Patterns per Label. . . . .	69
Figure 32 – Distribution of Recommendations per Microservices Pattern. . . . .	70

# List of Tables

Table 1 – Information of Each Microservices Pattern. . . . .	22
Table 2 – Boolean Model: Documents Represented. . . . .	24
Table 3 – Query Examples. . . . .	25
Table 4 – Recommendation Approach: Treatments Applied in the First Step. . . . .	32
Table 5 – Example of a Vector Space: Microservices Patterns. . . . .	34
Table 6 – Example of a Vector Space. . . . .	35
Table 7 – Text Class: Treatment that Each Method Does. . . . .	38
Table 8 – Each Toy Design Problem with the Microservices Pattern that Can Solve it. . . . .	41
Table 9 – Recommendations per Design Problem (DP). . . . .	42
Table 10 – Metrics per Design Problem. . . . .	43
Table 11 – Time of Experience of Developers Involved in the Tests. . . . .	68
Table 12 – Questions Asked in the Interviews. . . . .	68

# Lista de códigos

Código 1 – MPR: Dockerfile. . . . .	55
-------------------------------------	----

# List of abbreviations and acronyms

CDP	Catalog of Design Patterns
CMP	Corpus of Microservices Patterns
CS	Cosine Similarity
CSV	Comma-Separated Values
CTDP	Collection of Toy Design Problems
DDD	Domain-Driven Design
DF	Document Frequency
DP	Design Problem
DSR	Design Science Research
ERD	Entity Relationship Diagram
GoF	Gang of Four
IDF	Inverse Document Frequency
IR	Information Retrieval
IRS	Information Retrieval System
IS	Information Soup
ISs	Information Soups
IT	Information Technology
LDA	Latent Dirichlet Allocation
MAP	Mean Average Precision
MPR	Microservices Patterns Recommender
MVC	Model-View-Controller
NaN	Not a Number
NDCG	Normalized Discounted Cumulative Gain
NLTK	Natural Language Toolkit

OOP	Object-Oriented Programming
RPI	Remote Procedure Invocation
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SRP	Single Responsibility Principle
TF-IDF	Term Frequency-Inverse Document Frequency
TF	Term Frequency
UFS	Universidade Federal de Sergipe
UI	User Interface
WSDL	Web Services Description Language

# Contents

<b>1</b>	<b>Introduction</b>	<b>14</b>
1.1	Context	14
1.2	Problem	15
1.3	Objectives	15
1.4	Methodology	16
1.5	Related Works	17
1.5.1	Approach: Information Retrieval	17
1.5.2	Approach: Text Classification	17
1.5.3	Approach: Mix	18
1.5.4	Comparison of this Research with Others	18
1.6	Work Structure	19
<b>2</b>	<b>Theoretical Background</b>	<b>20</b>
2.1	Microservices	20
2.2	Design Patterns	21
2.3	Information Retrieval	22
2.3.1	Steps Performed by an Information Retrieval System	22
2.3.1.1	Step 1: Preprocess Documents	22
2.3.1.2	Step 2: Represent Documents and Query	24
2.3.1.3	Step 3: Match	24
2.3.2	Evaluation Metrics	25
2.3.2.1	Precision	25
2.3.2.2	Recall	25
2.3.2.3	F1-Score	25
2.3.2.4	Coverage	26
<b>3</b>	<b>A Microservices Patterns Recommendation Approach</b>	<b>27</b>
3.1	Corpus of Microservices Patterns	28
3.2	Recommendation Approach	30
3.2.1	Step 1: Preprocess Documents	31
3.2.2	Step 2: Represent Documents and Query	33
3.2.3	Step 3: Match	34
3.3	Implementation of the Recommendation Approach	35
3.3.1	Step 1: Preprocess Documents	36
3.3.2	Step 2: Represent Documents and Query	38
3.3.3	Step 3: Match	38

3.4	Evaluation of the Recommendation Approach . . . . .	40
3.5	Results . . . . .	41
<b>4</b>	<b>Floc: A Microservices Patterns Recommendation Tool . . . . .</b>	<b>45</b>
4.1	Floc Components View . . . . .	45
4.1.1	Microservices Patterns Recommender . . . . .	46
4.1.2	Floc Database . . . . .	47
4.2	Floc Implementation View . . . . .	49
4.3	Floc Scenarios of Execution . . . . .	51
4.4	Floc Deployment View . . . . .	53
4.5	Floc Features . . . . .	56
4.5.1	Login . . . . .	57
4.5.2	Sign Up . . . . .	58
4.5.3	List of Design Problems . . . . .	59
4.5.4	Solve Design Problem . . . . .	60
4.5.5	View Design Problem Details . . . . .	62
4.5.6	Edit Design Problem . . . . .	63
4.5.7	Delete Design Problem . . . . .	64
4.5.8	Feedback for a Recommended Microservices Pattern . . . . .	65
4.5.9	See More Information About a Microservices Pattern . . . . .	66
4.5.10	Logout . . . . .	67
4.6	Floc Evaluation . . . . .	67
4.7	Results . . . . .	68
4.7.1	Tests with Industrial Design Problems . . . . .	69
4.7.2	Interviews with Developers . . . . .	70
4.8	Threats to Validity . . . . .	72
4.8.1	Construct Validity . . . . .	72
4.8.2	Internal Validity . . . . .	72
4.8.3	External Validity . . . . .	72
4.8.4	Conclusion Validity . . . . .	72
<b>5</b>	<b>Conclusions, Contributions and Future Works . . . . .</b>	<b>73</b>
5.1	Conclusions . . . . .	73
5.2	Contributions . . . . .	75
5.3	Future Works . . . . .	75
	<b>Bibliography . . . . .</b>	<b>77</b>



# 1

## Introduction

### 1.1 Context

When a developer is developing or maintaining a system, it is common for this developer to face design problems. A design problem is a recurring problem ([SANYAWONG; NANTAJEE-WARAWAT, 2015](#)), which is capable of negatively impacting the quality attributes of a system ([SOUSA et al., 2017](#)), for this reason it is important to solve it. Design problems are commonly solved through design patterns. A design pattern is a proven solution, developed by experienced developers ([HUSSAIN et al., 2019](#)). Thus, developers often use design patterns to solve design problems.

Usually, a design pattern can be found in a Catalog of Design Patterns (CDP) that proposes solutions to solve design problems of a specific context. For example, the book named “Design Patterns: Elements of Reusable Object-Oriented Software” ([GAMMA et al., 1995](#)) is a popular CDP that proposes solutions to solve design problems present in systems developed using concepts of Object-Oriented Programming (OOP).

The Service-Oriented Architecture (SOA) is an architectural style that proposes that a system is composed of services/Web services, which are independent and well-defined modules that offer functionalities that can be used by other systems or Web services ([PAPAZOGLU; HEUVEL, 2007](#)). Web services are described using a standard definition language, such as Web Services Description Language (WSDL) and, typically, communication between Web services occurs through a protocol, such as Simple Object Access Protocol (SOAP) ([PAPAZOGLU; HEUVEL, 2007](#)).

Microservices architecture is an architectural style known as a variant of Service-Oriented Architecture (SOA) ([SOARES; FRANCA, 2016](#)), this architectural style has become increasingly popular. Several companies, such as Amazon, eBay and Netflix are adopting this architectural style ([CHEN; LI; LI, 2017](#)) which proposes that a system is composed of microservices that

communicate through messages (DRAGONI et al., 2017). A microservice is a small application that has a single responsibility and can be independently deployed, scaled and tested (THÖNES, 2015).

Another architectural style is monolithic architecture that proposes the encapsulation of all business rules in a single system (CHEN; LI; LI, 2017). Monolithic architecture is widely used and can be considered a traditional architectural style. As mentioned, several companies are adopting the microservices architecture, migrating from monolithic architecture to microservices architecture, this is the case of the companies mentioned (Amazon, eBay and Netflix) (CHEN; LI; LI, 2017). During this migration, a developer can face, for example, a design problem related to decomposition of microservices.

There is a CDP that proposes solutions to solve design problems present in systems based on microservices architecture, this CDP is Richardson's book called "Microservices Patterns: With Examples in Java" (RICHARDSON, 2018). The design patterns present in this catalog are known as microservices patterns, this catalog contains 50 microservices patterns. Thus, for developers who work on systems based on microservices architecture, understanding the catalog of microservices patterns can be considered essential.

## 1.2 Problem

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 in Section 1.1, 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 have difficulty when selecting design patterns, given the substantial number of design patterns and limited knowledge (HAMDY; ELSAYED, 2018; SANYAWONG; NANTAJEEWARAWAT, 2015). Therefore, this work seeks to offer a solution to this difficulty.

## 1.3 Objectives

The general objective of this work is to provide a way to help developers to select the correct microservices pattern to solve a given design problem. To achieve the general objective, the following specific objectives are proposed:

- Propose a microservices patterns recommendation approach;
- Develop a tool that makes use of this approach for developers to use.

## 1.4 Methodology

This work was developed using the research methodology called “Design Science Research” (DSR), which proposes the development of Information Technology (IT) artifacts to solve real-world problems (HEVNER et al., 2010). An IT artifact can be, for example, a system, an algorithm or even a framework. According to Hevner (HEVNER, 2007), this research methodology has 3 cycles: Relevance, Design and Rigor.

The real-world problem is identified in the “Relevance Cycle”. In the “Rigor Cycle”, existing solutions are investigated, a rigorous evaluation is carried out, involving, for example, metrics, and the results are communicated. The “Design Cycle” is considered the “Central Cycle”, the “Design Cycle” involves the development of an artifact to solve the problem identified and preliminary evaluation of this artifact.

As presented in Section 1.2, the real-world problem that this work seeks to solve is the difficulty that beginners and experienced developers may experience when selecting a design pattern, given a design problem. The reasons of this difficulty are the considerable number of existing patterns and lack of mastery of these patterns. Thus, as explained in Section 1.3, the general objective of this work is to provide a solution capable of helping developers to select design patterns to solve design problems.

It is important to highlight that this work is focused on microservices-based systems, this means that this work is restricted only to design patterns applied in these systems, also known as microservices patterns, and to design problems present in these systems. Systems that have many design problems are subject to being discontinued or re-engineered (SOUSA et al., 2018). The problem that this work seeks to solve is important because it can help to prevent microservices-based systems reaching this point.

After identifying the problem, understanding its importance and defining objectives, existing solutions were investigated. These solutions can be viewed in Section 1.5. In summary, the solutions found largely deal with “Gang of Four” (GoF) design patterns and design problems present in systems developed using concepts of Object-Oriented Programming (OOP). These solutions are based on Information Retrieval (IR), text classification or mix. No solution was found that deals with microservices patterns and design problems present in microservices-based systems.

The artifact to solve the problem addressed in this work was developed after understanding existing solutions. In summary, this artifact is able to receive a design problem reported by a developer and recommend 3 microservices patterns that can solve the design problem reported. Chapters 3 and 4 show how this artifact was developed, evaluated and evolved.

Given the information presented, the 3 cycles (Relevance Cycle, Rigor Cycle and Design Cycle) were applied in this work. The artifact developed can be considered innovative, as no solution was found that deals with microservices patterns and design problems present in

microservices-based systems. According to Hevner (HEVNER et al., 2010), it is important that the artifact developed in a DSR-based research is innovative.

## 1.5 Related Works

In this Section, some work related to the recommendation of design patterns are discussed and what are the differences of this work in relation to 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.

### 1.5.1 Approach: Information Retrieval

Hamdy and Elsayed (HAMDY; 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 unigrams 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; RASOOLZADEGAN; DEHKORDY, 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; HONG et al., 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.

### 1.5.2 Approach: Text Classification

Sanyawong and Nantajeewarawat (SANYAWONG; 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 (SANYA-

[WONG; 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 in relation to the other algorithms, the authors observed this after checking 4 metrics: accuracy, precision, recall and F1-score.

### 1.5.3 Approach: Mix

Celikkan and Bozoklar ([CELIKKAN; 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. In order 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.

### 1.5.4 Comparison of this Research with Others

The main difference between this work in relation to the presented works is that it deals with recommending microservices patterns, while the presented works deal with other types of patterns, mainly GoF patterns. Like most of the works presented, this work also proposes a recommendation approach based on information retrieval.

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.

## **1.6 Work Structure**

This work contains 4 additional Chapters: Chapter 2, where important concepts are presented for the understanding of the work; Chapter 3, where the microservices patterns recommendation approach is presented; Chapter 4, where a tool for developers is presented, which makes use of this approach; and Chapter 5, which is the last one, where conclusions, contributions and future works are presented.

# 2

## Theoretical Background

### 2.1 Microservices

The microservices architecture proposes that an application is composed of microservices that communicate through messages (DRAGONI et al., 2017). A microservice is a small application that has a single responsibility and can be independently deployed, scaled and tested (THÖNES, 2015).

This architecture has become increasingly popular, as it offers considerable benefits when compared to monolithic architecture, such as easier maintenance; ability to deploy and scale a module individually; and chance to use different and more appropriate technologies in each module (TAIBI; LENARDUZZI; PAHL, 2017). Although it offers benefits like these, it is not so easy to adopt it, as one has to deal with extra machinery that can impose considerable costs (SINGLETON, 2016). For example, it may be necessary to invest in tools related to communication, deployment and monitoring. In addition, there are still efforts, such as determining services; make them communicate; and be tested and deployed automatically (TAIBI; LENARDUZZI; PAHL, 2018).

The monolithic architecture is a traditional alternative and widely used in applications, it consists of encapsulating all modules in a single application (CHEN; LI; LI, 2017). Communication between modules is simple, it occurs through method calls. A system based on this architecture can be easily deployed since there is only one component. It is also easy to scale such a system, just create instances of it and use a load balancer (KOSCHEL; ASTROVA; DÖTTERL, 2017).

Some problems of this architecture arise in a system based on it, as this system grows, as follows: as the number of modules increases and the existing modules as well, the coupling between them makes the maintenance and evolution of the system difficult; and the larger the system becomes, the more time is required to compile, test and deploy it (CHEN; LI; LI, 2017). Other problems are already incorporated into the architecture itself, such as: making any changes

to a specific module available requires deploying the entire system (CHEN; LI; LI, 2017); scaling a specific module individually is not possible (FRITZSCH et al., 2019); and it is also not possible to use different and appropriate technologies in different modules (TAIBI; LENARDUZZI; PAHL, 2017).

The microservices architecture is not a “silver bullet”<sup>1</sup> for these problems, but this architecture has proven to be an interesting solution. Therefore, several companies are adopting this architecture, while others, such as Amazon, eBay and Netflix, have already adopted it (CHEN; LI; LI, 2017).

## 2.2 Design Patterns

A problem that recurs in systems is called a design problem (SANYAWONG; NANTA-JEEWARAWAT, 2015) and can negatively impact the quality attributes of a system (SOUSA et al., 2017). Applications that have many design problems may need to be discontinued or re-engineered (SOUSA et al., 2018). One example is an application with low maintainability, thanks to one or more design problems, where it becomes difficult to make changes to correct defects or adapt to new requirements.

To solve design problems, developers make use of design patterns, which are proven solutions devised by experienced developers through trials and errors (HUSSAIN et al., 2019). For example, there is a design pattern called Remote Procedure Invocation (RPI) that is used when the problem is “how to make microservices communicate”. This pattern allows microservices to communicate through a request/response based protocol.

A design pattern used to solve a design problem in a system based on microservices architecture is called a microservices pattern, RPI is an example. Chris Richardson’s book, briefly introduced in Section 1.1, is a relevant catalog of such patterns. The author also has a website where he exposes the information of each pattern (RICHARDSON, 2023). This information is presented in Table 1.

In addition to this catalog, there are others that expose patterns to solve problems related to other contexts, such as the popular book called “Design Patterns: Elements of Reusable Object-Oriented Software” which presents 23 patterns to deal with problems in systems based on Object-Oriented Programming (OOP) (GAMMA et al., 1995).

In view of the damage caused by design problems, it is possible to note that design patterns are extremely important to guarantee quality software, especially when they are correctly selected and applied. This makes evident how fundamental it is for a developer to master design patterns.

---

<sup>1</sup> The term “silver bullet” is used by Brooks (BROOKS; KUGLER, 1987) as a metaphor for a single solution capable of quickly and easily solving a problem related to software development.



Table 1 – Information of Each Microservices Pattern.

Information	Description
Name	One or more words designating the pattern
Group	Class the pattern belongs to, for example, RPI is a communication pattern
Subgroup	Subclass that the pattern belongs to, for example, RPI is a communication style pattern
Layer	Level at which the pattern is used: Infrastructure, Application Infrastructure or Application
Context	Circumstance in which the default should be used
Problem	Question that the pattern solves
Forces	Represent a concrete scenario that exposes the motivation for using the pattern, considering the context and the problem to be solved
Solution	How the pattern solves the problem
Examples	Show how to use the pattern
Resulting Context	Describes the postconditions and side effects that arise when using the pattern
Issues	Pertinent points that must be addressed before the pattern is used
Related Patterns	Names of other patterns that have some relation to the pattern

## 2.3 Information Retrieval

Information Retrieval (IR) is the process of retrieving documents that match a query (HAMBARDE; 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 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).

### 2.3.1 Steps Performed by an Information Retrieval System

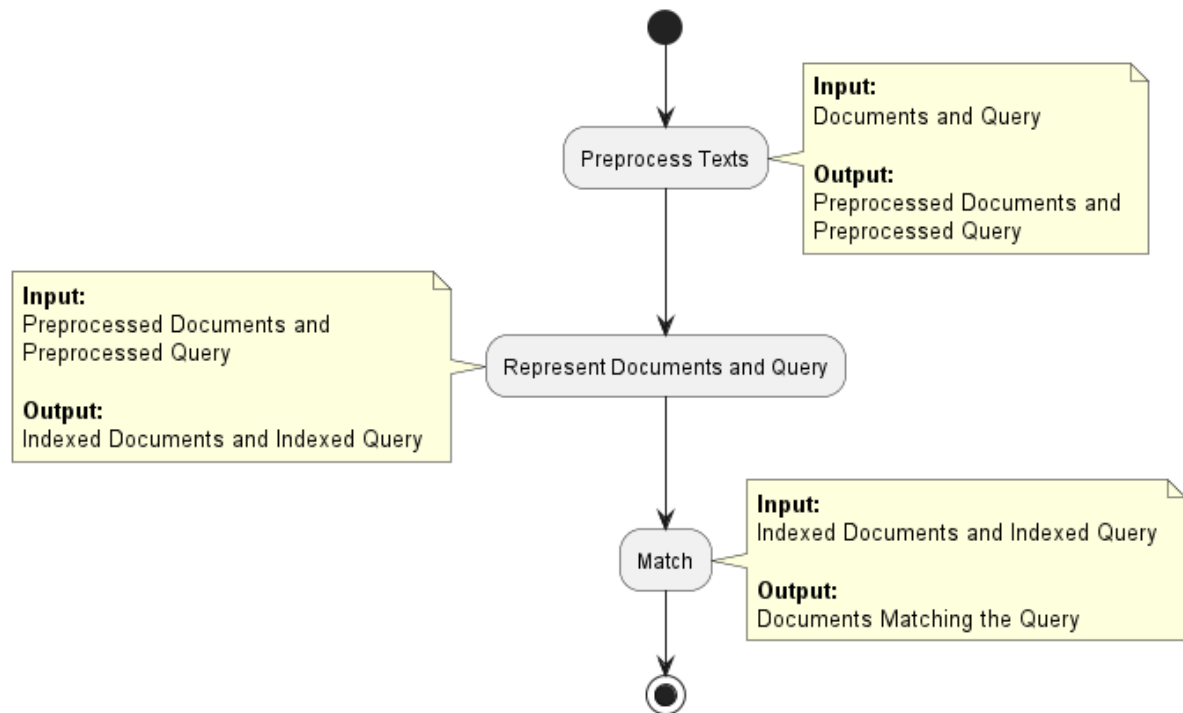
An IRS that deals with text documents performs 3 steps to retrieve them: 1) Preprocess Documents, 2) Represent Documents and Query, and 3) Match (ROSHDI; ROOHPARVAR, 2015). Figure 1 shows these steps, including their inputs and outputs.

#### 2.3.1.1 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 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

Figure 1 – Steps Performed by an Information Retrieval System.



“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 Subsection 3.2.1.

### 2.3.1.2 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 Retrieval;
- **Document 3:** Master dissert.

Table 2 – Boolean Model: Documents Represented.

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

In the VSM, the preprocessed text documents and the preprocessed query are represented by vectors (IBRIHICH et al., 2022). Subsections 3.2.2 and 3.2.3 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.

### 2.3.1.3 Step 3: Match

In 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 like “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 is used a measure of similarity to check how similar a document is to a query. Documents that have a considerable degree of similarity are retrieved. In Subsection 3.2.3, this process is explained in detail. This step is the last one, its outputs are the recovered documents.

### 2.3.2 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.

Table 3 – 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 $D_8$

#### 2.3.2.1 Precision

The Precision metric indicates the relationship between the number of Important Documents Retrieved (IDR) and RDs. This metric is given by Equation 2.1. 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.

$$Precision = \frac{|important\ documents \cap retrieved\ documents|}{|retrieved\ documents|} \quad (2.1)$$

#### 2.3.2.2 Recall

The Recall metric indicates the relationship between the number of IDR and IDs. This metric is given by Equation 2.2. 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 ( $\approx 35\%$ ), Q3 (100%) and Q4 (100%). The higher a Recall, the better.

$$Recall = \frac{|important\ documents \cap retrieved\ documents|}{|important\ documents|} \quad (2.2)$$

#### 2.3.2.3 F1-Score

The Precision and Recall metrics are frequently used to measure the quality of an IRS (SAINI; SINGH; KUMAR, 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 2.3.

$$F1\text{-Score} = \frac{2 \cdot (\textit{Precision} \cdot \textit{Recall})}{\textit{Precision} + \textit{Recall}} \quad (2.3)$$

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

#### 2.3.2.4 Coverage

The Coverage metric indicates the proportion of IDs that the IRS was able to retrieve. This metric is given by Equation 2.4. 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, multiplying by 100, 100%.

$$\textit{Coverage} = \frac{\textit{total important documents retrieved}}{\textit{total number of important documents available}} \quad (2.4)$$

# 3

## A Microservices Patterns Recommendation Approach

In this Chapter, an IR-based recommendation approach is proposed to make microservices patterns recommendations, where the developer can report a design problem, in natural language (text), and receive microservices patterns recommendations that can solve the reported design problem. From this approach, it is possible to develop a tool to help developers in selecting microservices patterns to solve design problems present in microservices-based systems.

The Corpus of Microservices Patterns (CMP) is an important component, as it is responsible for storing the microservices patterns that can be recommended. This means that the microservices patterns recommended by the recommendation approach are retrieved from the CMP. Thus, the CMP can be considered a fundamental component, since it is from it that the recommendation approach obtains the microservices patterns necessary for its recommendations. Section 3.1 describes the CMP in details.

In Section 3.2, a detailed explanation of how the recommendation approach works is provided. Section 3.3, in turn, presents a computer program developed in Python that represents the implementation of this approach. For the development of this program, object-oriented programming principles were followed, along with the use of some popular libraries, such as Natural Language Toolkit (NLTK) and Scikit-Learn, in order to guarantee a certain ease and speed in development.

The recommendation approach is evaluated through tests carried out with toy design problems, in which microservices patterns recommendations were requested for these problems. Section 3.4 presents these toy design problems and details of how the recommendation approach was evaluated. After obtaining the recommendations, they were analyzed to verify the quality of the recommendation approach through metrics. The results of the tests carried out, together with the analysis of the recommendations obtained and the evaluation of quality metrics, are presented in Section 3.5.

## 3.1 Corpus of Microservices Patterns

The CMP is a fundamental component for recommending microservices patterns, given a design problem, through information retrieval. The CMP is responsible for having the description of each microservices pattern. This description is formed by 14 pieces of information, 12 of which can be seen in Table 1. The other 2 pieces of information are as follows:

- **Id:** Unique number used to identify the microservices pattern;
- **Link:** Page address on Richardson’s website ([RICHARDSON, 2023](#)) that provides additional information about the microservices pattern.

Technically, the CMP is a Comma-Separated Values (CSV) file. A CSV file is a simple text file used to store data in tabular format, where each line of the file represents a row of the table and the values within each row are separated by commas or other delimiters, such as semicolons. Figure 2 shows a screenshot of the CMP. At row 1, where the table columns are, there are 7 pieces of information that were not mentioned, they are: “consequent\_benefits”, “consequent\_drawbacks”, “consequent\_issues”, “successors”, “alternatives”, “generalizations” and “specializations”. In fact, this information was synthesized into 2 pieces of information, where “Resulting Context” represents the first 3 pieces of information: “consequent\_benefits”, “consequent\_drawbacks” and “consequent\_issues”, while “Related Patterns” represents other information: “successors”, “alternatives”, “generalizations” and “specializations”. Thus, actually these 7 pieces of information were presented.

The CMP contains the description of 10 microservices patterns presented in the book called “Microservices Patterns: With Examples in Java” by Richardson ([RICHARDSON, 2018](#)). Out of these 10 microservices patterns, 2 are decomposition patterns and 8 are communication patterns, as shown in Figure 3. Communication patterns can be divided into subgroups, Figure 4 shows the number of communication patterns per subgroup. One can note that 2 of the communication patterns belong to the communication style subgroup, 1 to the reliability subgroup, 4 to the discovery subgroup and 1 to the transactional messaging subgroup.

The 10 microservices patterns present in CMP are: Decompose by Business Capability, Decompose by Subdomain, Remote Procedure Invocation, Circuit Breaker, Self Registration, Client-Side Discovery, 3rd Party Registration, Server-Side Discovery, Messaging and Transactional Outbox. Microservices patterns are presented per group and, in the case of communication patterns, per subgroup, as follows:

- **Decomposition:**
  - Decompose by Business Capability
  - Decompose by Subdomain

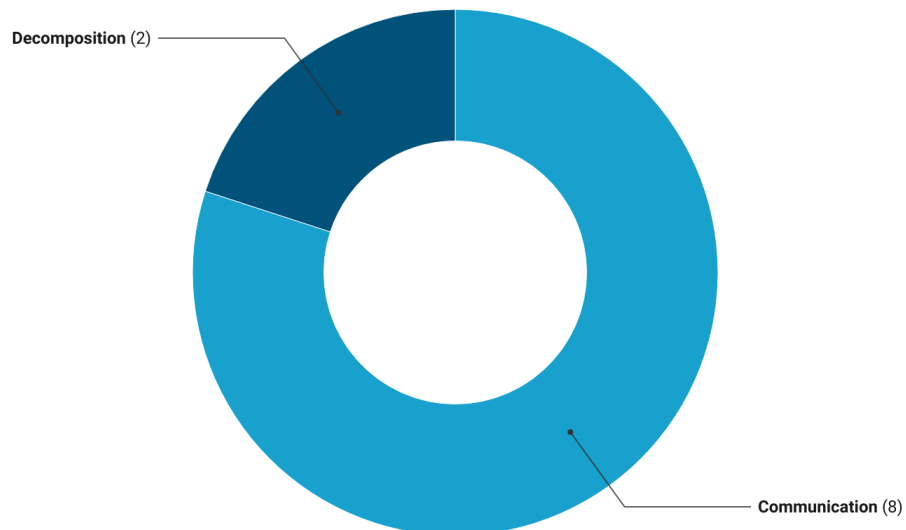
Figure 2 – CMP Screenshot.

```

1 id,link,name,group,subgroup,layer,context,problem,forces,solution,consequent_benefits,consequent_drawbacks,
  consequent_issues,issues,predecessors,successors,alternatives,generalizations,specializations
2 1,https://microservices.io/patterns/decomposition/decompose-by-business-capability.html,Decompose by business capability,
  Decomposition,,Application,"<p>
3   You are developing a large, complex application and want to use the
4   <a href=""/>microservices.html">microservice architecture</a>. The
5   microservice architecture structures an application as a set of loosely
6   coupled services. The goal of the microservice architecture is to accelerate
7   software development by enabling continuous delivery/deployment.
8 </p>
9
10 <p><img src=""/>successtriangle.png" class="img-responsive" /></p>
11
12 <p>The microservice architecture does this in two ways:</p>
13
14 <ol>
15   <li>Simplifies testing and enables components to deployed independently</li>
16   <li>
17     Structures the engineering organization as a collection of small (6-10
18     members), autonomous teams, each of which is responsible for one or more
19     services
20   </li>
21 </ol>
22
23 <p>
24   These benefits are not automatically guaranteed. Instead, they can only be
25   achieved by the careful functional decomposition of the application into
26   services.
27 </p>
28
29 <p>

```

Figure 3 – Microservices Patterns per Group.



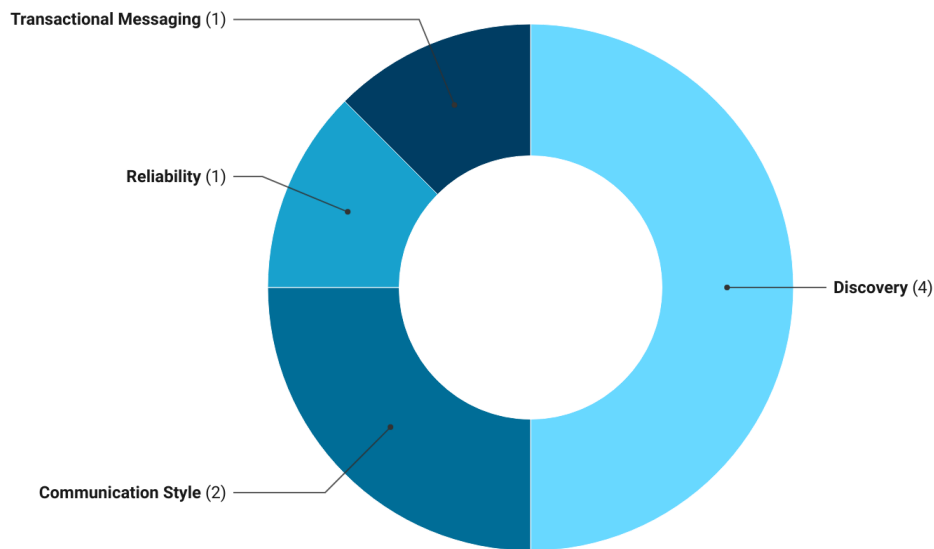
- **Communication:**

- **Communication Style:**

- \* Remote Procedure Invocation
    - \* Messaging



Figure 4 – Communication Microservices Patterns per Subgroup.



- **Reliability:**
  - \* Circuit Breaker
- **Discovery:**
  - \* Self Registration
  - \* Client-Side Discovery
  - \* 3rd Party Registration
  - \* Server-Side Discovery
- **Transactional Messaging:**
  - \* Transactional Outbox

At the time, 50 microservices patterns are presented on the Richardson website (RICHARDSON, 2023). These patterns belong to 9 different groups. This means that the CMP contains 20% of the patterns presented by Richardson, covering more than 20% of the patterns groups. It is worth mentioning that the information that forms the description of each microservices pattern was taken from two sources, the book and Richardson’s website. The CMP is important because it contains the microservices patterns that can be recommended.

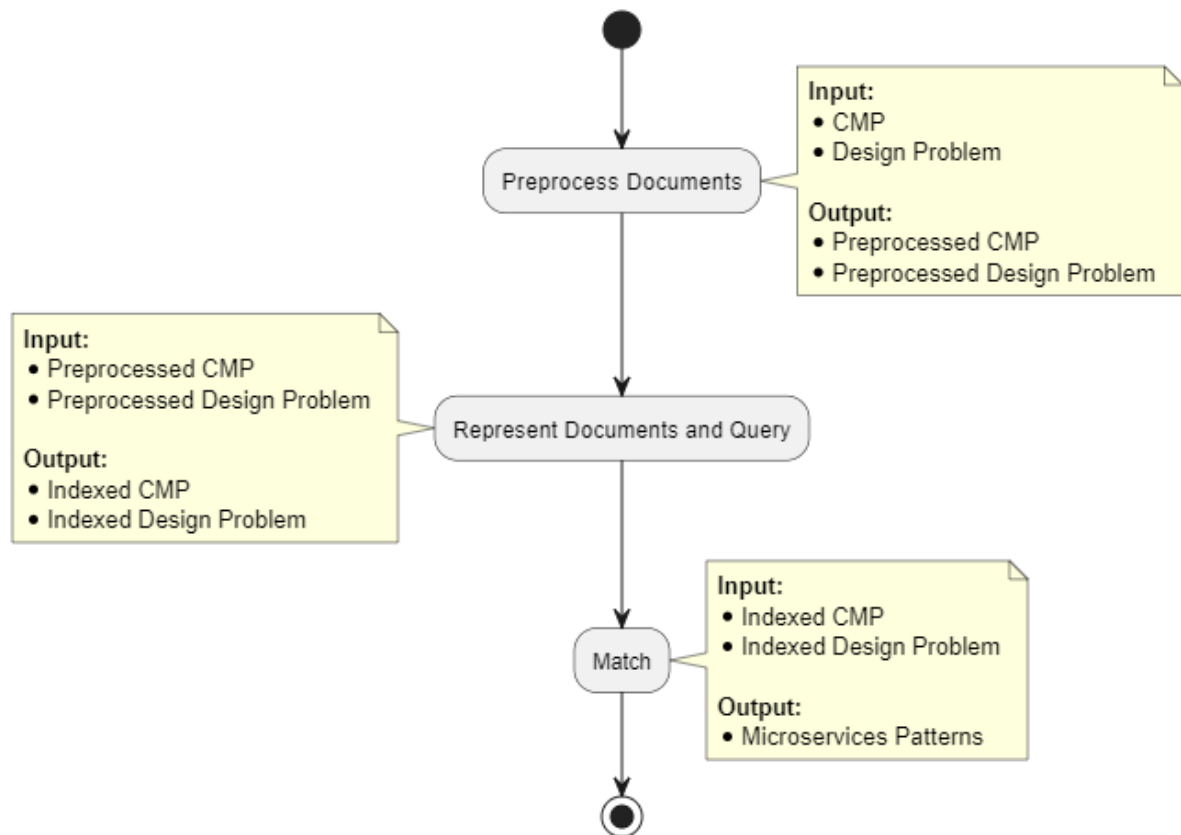
## 3.2 Recommendation Approach

To recommend microservices patterns, given a design problem, this work proposes a recommendation approach based on information retrieval. Classification algorithms can be used as another way to make recommendations, but these algorithms depend on a dataset with many examples and covering all the different possible categories to generate effective results

(UYSAL, 2016). In the context of this work, the examples would be design problems and the categories would be microservices patterns. As no such dataset was found, this work proposes a recommendation approach based on information retrieval.

As Figure 5 shows, the proposed approach is defined in 3 steps, which are: 1) Preprocess Documents, 2) Represent Documents and Query, and 3) Match. Each of these steps is explained in this Section. Subsection 3.2.1 explains the first step. Subsection 3.2.2 explains the second step. Finally, Subsection 3.2.3 explains the third and last step.

Figure 5 – Recommendation Approach: Steps.



### 3.2.1 Step 1: Preprocess Documents

In this Chapter, the term “documents” is used to refer to each microservices pattern present in the CMP, and to the design problem for which microservices pattern recommendations are sought. This design problem is reported, in natural language, by the developer. To be more specific, this design problem is informed in text. As Figure 5 shows, the CMP and this design problem are the inputs of this step that is responsible for preprocessing them.

Preprocessing documents means applying a set of treatments to them that make them more suitable for being represented and then, in the case of microservices patterns, recovered. Thus, it is possible to have an effective information retrieval. In total, 12 treatments are applied,

Table 4 – Recommendation Approach: Treatments Applied in the First Step.

Treatment	Input (E.g.)	Output (E.g.)
Remove Null Values	Null	Empty String
Change Letters to Lowercase	This is a TEXT	this is a text
Remove Not a Number (NaN)	NaN	Empty String
Remove New Lines	This is a \n Text	This is a Text
Remove Tabs	This is a \t Text	This is a Text
Remove Returns	This is a \r Text	This is a Text
Remove HTML Tags	This is a <b>Text</b>	This is a Text
Remove Punctuation	This is a Text!!!	This is a Text
Remove Extra Spaces	This is a Text	This is a Text
Get Uncontracted Words	It's hard	It is hard
Remove Stop Words	This is a communication pattern	communication pattern
Get Stems	Communication	Commun

as follows: 1) Remove Null Values, 2) Change Letters to Lowercase, 3) Remove Not a Number (NaN), 4) Remove New Lines, 5) Remove Tabs, 6) Remove Returns, 7) Remove HTML Tags, 8) Remove Punctuation, 9) Remove Extra Spaces, 10) Get Uncontracted Words, 11) Remove Stop Words and 12) Get Stems. Table 4 exposes these 12 treatments, including input and output examples.

In the case of microservices patterns, only 6 pieces of information are preprocessed, they are: Context, Problem, Forces, Solution, Resulting Context and Related Patterns. These 6 pieces of information are used in the recovery process. Other information is not considered because it does not contribute to this process. The following are the data not considered and the reason for this:

- **Id:** Serves only as an identifier for each microservices pattern in the CMP;
- **Name:** It is a specific information, but it may not be interesting because if a design problem has the name of a certain microservices pattern, this microservices' pattern will probably be recommended, which is not interesting for the developer, because if he/she reported the name this microservices pattern is because he/she already knows it;
- **Group:** Generic information, present in many microservices patterns;
- **Subgroup:** Generic information, present in many microservices patterns;
- **Layer:** Generic information, present in many microservices patterns;
- **Examples:** Information from specific contexts, for example, supermarket. In addition to involving images, which is not interesting for the recommendation approach because it does not support them;
- **Issues:** Useful information when the microservices pattern is already known, as it indicates important points that must be considered before applying the pattern;

- **Link:** It is only useful for the developer to know more about a microservices pattern.

After documents undergo these treatments, they are called preprocessed documents. It is worth mentioning that the 6 preprocessed information from each microservices pattern are concatenated, forming what is called Information Soup (IS). Thus, the IS of each microservices pattern and the preprocessed design problem are the outputs of this step, also called preprocessed documents.

### 3.2.2 Step 2: Represent Documents and Query

Represent documents and query means transforming preprocessed documents into something that the computer is capable of understanding and analyzing. With the documents preprocessed and represented, it is possible to compare the design problem with each IS of each microservices pattern and retrieve the 3 microservices patterns that most closely resemble the design problem. This comparison is made in step 3.

In this step, the Term Frequency-Inverse Document Frequency (TF-IDF) model is applied to represent the preprocessed documents. This model assigns a numerical value to each term/word of a document. This numerical value can also be called weight and is given by two measurements, Term Frequency (TF) and Inverse Document Frequency (IDF). This weight indicates the importance of a term in a document, considering the importance of this term in a corpus of documents. In the case of microservices patterns, the corpus considered is the CMP. The mentioned weight is calculated using Equation 3.1, where “t” represents a term and “d” a document.

$$TF - IDF(t, d) = TF(t, d) \cdot IDF(t) \quad (3.1)$$

As mentioned and shown in Equation 3.1, to calculate the weight it is necessary to calculate the TF and the IDF. The TF is the number of times a term appears in a document divided by the total number of terms this document has, i.e., it measures the frequency of a term in a document. The IDF measures how rare a term is in a corpus of documents. Rare terms have a higher IDF, while common terms have a lower IDF. The IDF is calculated using Equation 3.2. Where “n” is the total number of documents present in a corpus, and Document Frequency (DF) is the number of documents that have the term represented by “t”.

$$IDF(t) = \log \frac{1 + n}{1 + DF(t)} + 1 \quad (3.2)$$

The greater the weight of a term in a document, the more important this term is for this document. Terms that appear frequently in a document, but are rare in other documents in a certain corpus, have higher weights. To illustrate, one can consider the IS of each microservices

pattern, in total there are 10 Information Soups (ISs), as the CMP contains 10 microservices patterns. The IS of a microservices pattern called “Messaging” contains the “communication” term, in total, this IS contains 100 terms. Assuming that this term appears 5 times in this IS and that this term appears in 2 of these 10 ISs, the value of TF is “0.05” and the value of IDF is approximately “1.60”. Therefore, the weight of the “communication” term in this specific IS is “0.08”. This weight indicates that this term is very important for this IS.

After the preprocessed documents are represented, they can be called represented documents. These documents are the outputs of this step. It is worth mentioning that the documents represented are the ISs and the design problem. These documents are useful in the next step.

### 3.2.3 Step 3: Match

The represented documents, which are the outputs of the previous step, are actually vectors in a multidimensional space, where each dimension corresponds to a term. There are 11 vectors in this space, 1 referring to the design problem and the other 10 referring to the 10 ISs. With these vectors available, it is possible to measure the similarity between the vector that represents the design problem and the vectors that represent microservices patterns, this step is responsible for this.

To exemplify, one can consider the 3 microservices patterns presented in Table 5 and this design problem “microservic need commun”. One can note that the microservices patterns and design problem are preprocessed. What the step 2 did initially was build a vocabulary based on the terms present in microservices patterns and the design problem. Then the weight of each term in a microservices pattern was calculated using TF-IDF, as explained in Subsection 3.2.2. This was also done for the design problem. In the end, the result is a vector space with the 5 vectors, where each microservices pattern is represented by a vector and the design problem is also represented by a vector.

Table 5 – Example of a Vector Space: Microservices Patterns.

ID	Microservices Pattern	Simplified IS
MP1	Decompose by Subdomain	defin servic correspond ddd subdomain
MP2	Remote Procedure Invocation	use rpibas protocol interservic commun
MP3	Messaging	use asynchron messag interservic commun

Table 6 shows the vector space of the commented example, where the first column represents the vocabulary was built, while the other columns represent the vectors. The numeric values present in each column that represent a vector are the weights. It is worth mentioning that, to simplify the example, the values have only one decimal place.

From a vector space like the one shown in Table 6, this step measures the similarity between the vector that represents the design problem and the vectors that represent microservices

Table 6 – Example of a Vector Space.

Terms	Design Problem Vector	MP1 Vector	MP2 Vector	MP3 Vector
microservic	0.7	0.0	0.0	0.0
need	0.7	0.0	0.0	0.0
commun	0.5	0.0	0.3	0.3
defin	0.0	0.3	0.0	0.0
servic	0.0	0.3	0.0	0.0
correspond	0.0	0.3	0.0	0.0
ddd	0.0	0.3	0.0	0.0
subdomain	0.0	0.3	0.0	0.0
use	0.0	0.0	0.3	0.3
rpibas	0.0	0.0	0.3	0.0
protocol	0.0	0.0	0.3	0.0
interservic	0.0	0.0	0.3	0.3
asynchron	0.0	0.0	0.0	0.3
messag	0.0	0.0	0.0	0.3

patterns. This is done using a Cosine Similarity (CS) that is given by Equation 3.3, where “V” is the microservices pattern vector, “W” is the design problem vector and “n” is the size of the vectors. Thus, the first 3 microservices patterns, ordered in descending order by CS value, are indicated to solve the design problem.

$$CS(V, W) = \frac{\sum_{i=1}^n V_i W_i}{\sqrt{\sum_{i=1}^n V_i^2} \sqrt{\sum_{i=1}^n W_i^2}} \quad (3.3)$$

### 3.3 Implementation of the Recommendation Approach

This Section presents the implementation of the recommendation approach presented in Section 3.2. Each Subsection in this Section describes the development of a specific step. Subsection 3.3.1 explains the implementation of the first step, which involves preprocessing the documents. Subsection 3.3.2 describes the implementation of the second step, where documents are represented. Finally, Subsection 3.3.3 explains the implementation of the last step, which involves measuring the similarity between the design problem that needs to be solved and the microservices patterns present in the CMP.

To implement the recommendation approach, a computer program was developed in Python. Python is a programming language that has a concise syntax and holds libraries that helped simplify the implementation of the approach. As development environment, Collaboratory was used, which is a cloud service offered by Google. Collaboratory allows to work with Python in an environment where multiple people can contribute. This environment was shared with 2 contributing researchers.

The computer program was developed using OOP, this computer program consists of

7 classes, as Figure 6 shows. The “Recommender” class is responsible for coordinating the execution of each step of the recommendation approach. This class uses the other classes to perform each step. One can note that the “Recommender” class has only 1 method called “recommend\_microservices\_patterns\_for”. This method receives a design problem and returns a list with 3 recommendations. The “Recommender” class performs its responsibility through the “recommend\_microservices\_patterns\_for” method.

The list, returned by the “recommend\_microservices\_patterns\_for” method, with 3 recommendations, is a list with 3 instances of the “Recommendation” class that, as its name suggests, represents a recommendation. One can note that the “Recommendation” class has 2 attributes, “rank” and “microservices\_pattern”. The “rank” attribute represents how important the recommendation is for the design problem, its value ranges from 1 to 3. Considering 3 recommendations: R1, R2 and R3, and assuming that the rank of R1 is “2”, the rank of R2 is “1” and the rank of R3 is “3”, R2 is more important to the design problem than R1 and R3, while R1 is more important than R3. The “rank” attribute is related to the similarity that the recommended microservices pattern has with the design problem, the more similar the recommended microservices pattern is to the design problem, the more important this pattern is. About the “microservices\_pattern” attribute, it represents the recommended microservices pattern.

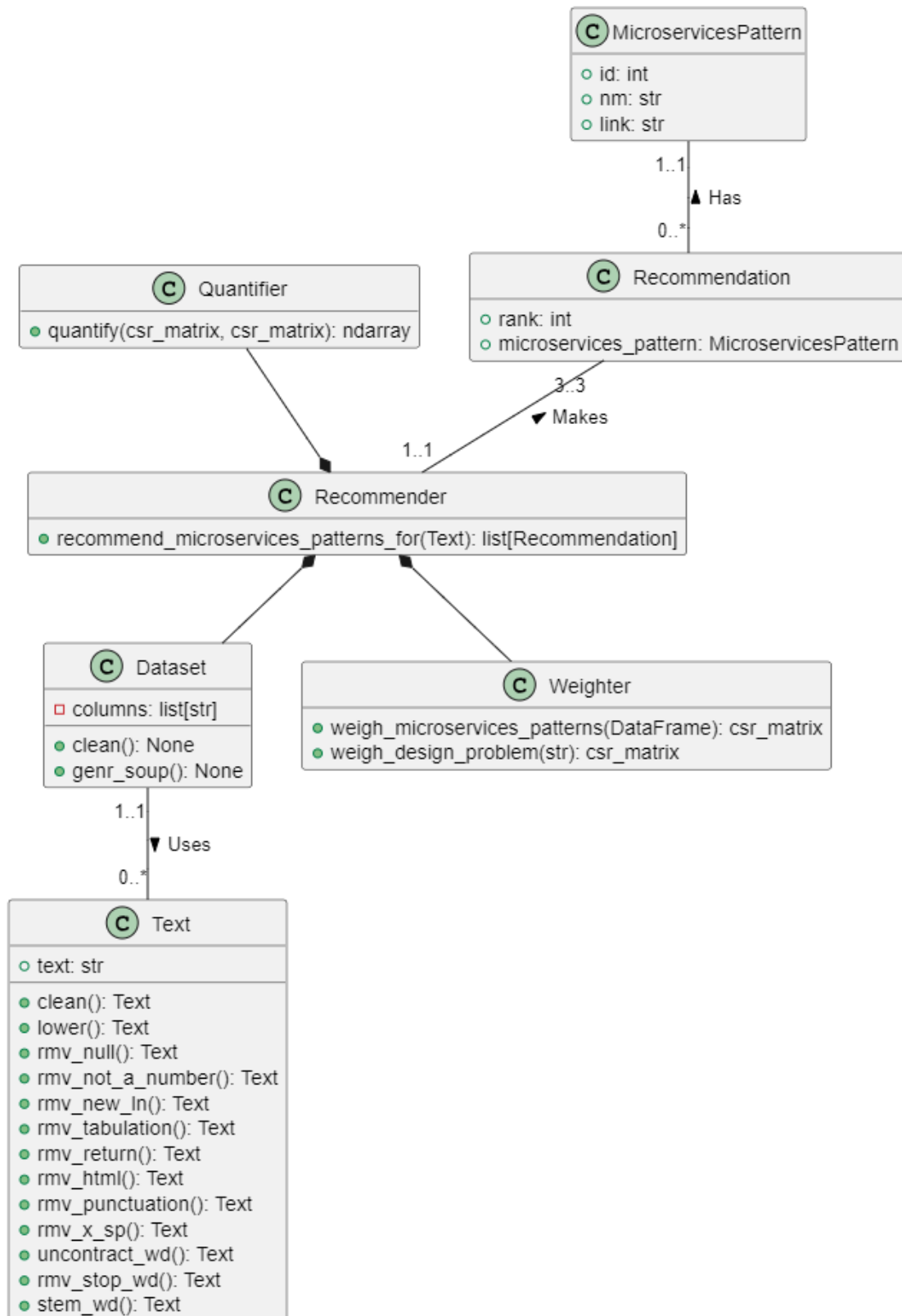
The “microservices\_pattern” attribute of the “Recommendation” class is of type “MicroservicesPattern”. The “MicroservicesPattern” class, as its name suggests, represents a microservices pattern and has 3 attributes, “id”, “nm” and “link”. The “nm” attribute represents the microservices pattern name. The “id” and “link” attributes have a clear name, which requires no explanation. The value for each of these attributes comes from the CMP. It is worth mentioning that this information was already presented in Section 3.1. In summary, the “Recommendation” and “MicroservicesPattern” classes are data structures that the “recommend\_microservices\_patterns\_for” method of the “Recommender” class uses as return. The other classes present in Figure 6 that were not mentioned are presented in the following 3 Subsections.

### 3.3.1 Step 1: Preprocess Documents

There are 2 classes involved in preprocessing of documents, the “Text” and “Dataset” classes. The “Text” class represents a text information, for example, a design problem or some information that a microservices pattern has and that must be preprocessed (Context, Problem, Forces, Solution, Resulting Context or Related Patterns). While a “Dataset” class represents the CMP.

The “Text” class offers methods capable of performing the treatments presented in Section 3.2.1. These treatments are performed on the text information that the “text” attribute stores. Table 7 shows the treatment that each method of the “Text” class does. The “clean” method is responsible for coordinating the execution of each treatment, invoking the other methods in

Figure 6 – Class Diagram: Recommendation Approach.



proper order. Thus, to preprocess, for example, a design problem, it is necessary to create an instance of “Text”, assign this design problem to the “text” attribute and then invoke the “clean”



method.

Table 7 – Text Class: Treatment that Each Method Does.

<b>Treatment</b>	<b>Method</b>
Remove Null Values	rmv_null
Change Letters to Lowercase	lower
Remove Not a Number (NaN)	rmv_not_a_number
Remove New Lines	rmv_new_ln
Remove Tabs	rmv_tabulation
Remove Returns	rmv_return
Remove HTML Tags	rmv_html
Remove Punctuation	rmv_punctuation
Remove Extra Spaces	rmv_x_sp
Get Uncontracted Words	uncontract_wd
Remove Stop Words	rmv_stop_wd
Get Stems	stem_wd

The “Text” class is also useful for preprocessing the microservices patterns present in the CMP, but only the “Text” class is not enough, because it is necessary to generate the IS of each microservices pattern. The “Dataset” class has 2 methods, “clean” and “genr\_soup”, and 1 attribute called “columns”, which represents the information that must be preprocessed about the microservices patterns. The “clean” method uses the “Text” class to preprocess the microservices patterns, while the “genr\_soup” method generates the ISs.

It is important to mention the libraries used to implement this step, which include “BeautifulSoup” for removing HTML tags, “NLTK” for eliminating stopwords and extracting stems, and “Pandas” for loading and manipulating the CMP.

### 3.3.2 Step 2: Represent Documents and Query

In this step, only 1 class is involved, this class is called “Weighter”. The “Weighter” class has 2 methods, “weigh\_microservices\_patterns” and “weigh\_design\_problem”. The “weigh\_microservices\_patterns” method is responsible for representing the microservices patterns or in other words, creating a vector for each IS, while the “weigh\_design\_problem” method is responsible for representing the design problem, creating a vector for the design problem. To represent microservices patterns and the design problem, this step was developed using a library called “Scikit-Learn”. This library is used in the mentioned methods.

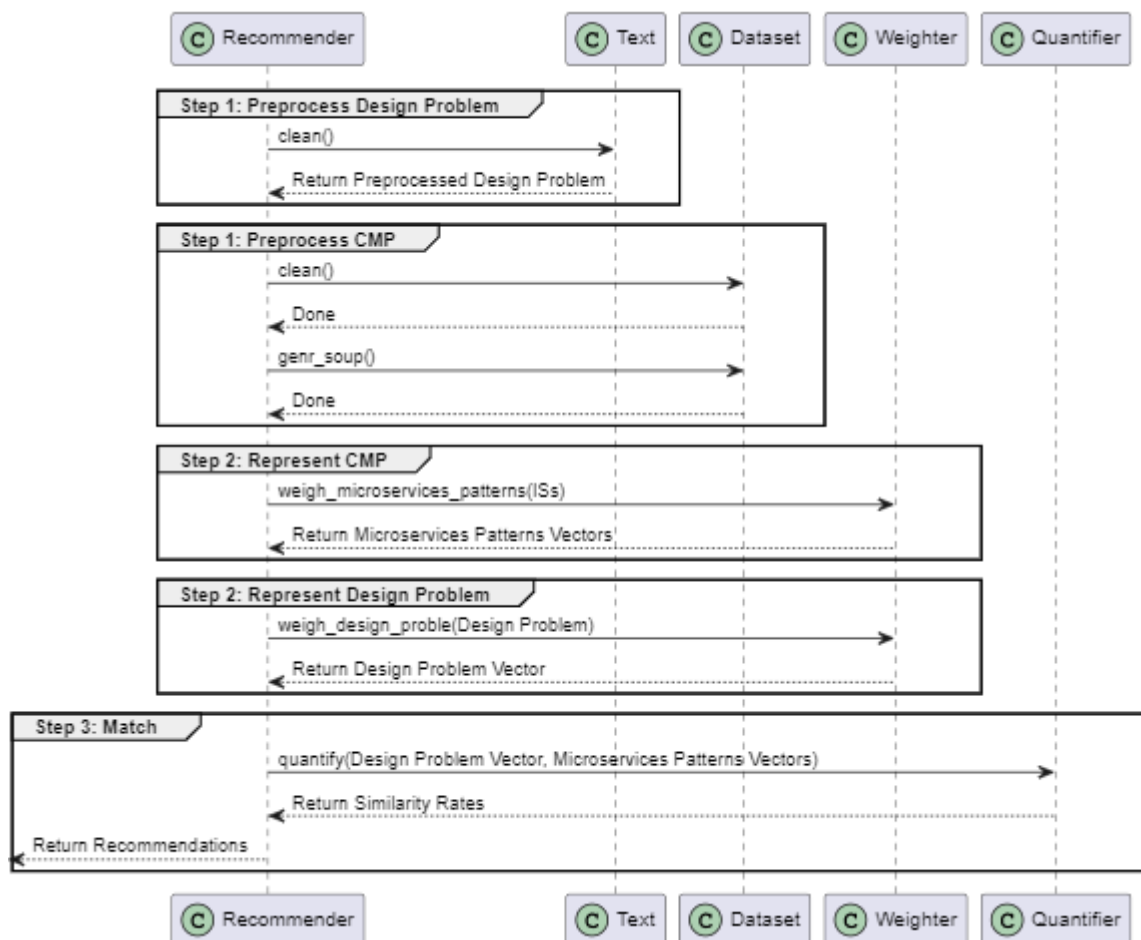
### 3.3.3 Step 3: Match

This step was also developed using the “Scikit-Learn” library. This library is used to measure the similarity between the design problem and microservices patterns. In this step, only 1 class is involved, this class is called “Quantifier”. The “Quantifier” class has only 1 method, this method is called “quantify”. The “quantify” method is responsible for receiving the vector

of the design problem and the vector of each microservices pattern, and then measuring the similarity between the vector of the design problem and the vectors of the microservices patterns, as explained in Section 3.2.3.

Considering that all classes have been presented, it is relevant to explain how the “Recommender” class interacts with each class. First, the “Recommender” class asks the “Text” class to preprocess the design problem passed by parameter to the “recommend\_microservices\_patterns\_for” method. Then, the “Recommender” class asks the “Dataset” class to preprocess the CMP. With the design problem and the CMP, both preprocessed, the “Recommender” class asks the “Weighter” class to represent them. After obtaining the design problem and the CMP, both represented, the “Recommender” class asks the “Quantifier” class to measure the similarity between the design problem and each microservices pattern present in the CMP. Finally, the “Recommender” class orders the microservices patterns by similarity and recommends the 3 microservices patterns that better fits to the design problem. Figure 7 shows this interaction.

Figure 7 – Sequence Diagram: Interaction Between the Classes.



## 3.4 Evaluation of the Recommendation Approach

With the intention of testing and evaluating the recommendation approach, 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. In view 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;
- **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 labeled with the microservices pattern that can solve it. Table 8 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.

Table 8 – Each Toy Design Problem with the Microservices Pattern that Can Solve it.

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

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 8 to verify the quality of the recommendation approach through the metrics presented in Subsection 2.3.2.

## 3.5 Results

This Section presents the results of tests carried out with the CTDP. The microservices patterns recommended for the design problems of this collection can be seen in Table 9. In total 30 microservices patterns were recommended, 3 microservices patterns recommended per design problem. The recommended microservices patterns are organized into 3 columns, "Recommendation 1", "Recommendation 2" and "Recommendation 3". These columns have relationship with the "rank" attribute, explained in Section 3.3.

When checking if each design problem received as recommendation the microservices pattern that is capable of solving it, as shown in Table 9, 6 of the 10 design problems received

Table 9 – Recommendations per Design Problem (DP).

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

and only 4 did not receive. The 6 design problems that received were: 1, 2, 3, 6, 7 and 10. While 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. Figure 8 shows this information, where the 6 that receives are called “Design Problems Solved” and the 4 that do not receives are called “Unsolved Design Problems”.

Figure 8 – Solved and Unsolved Design Problems.

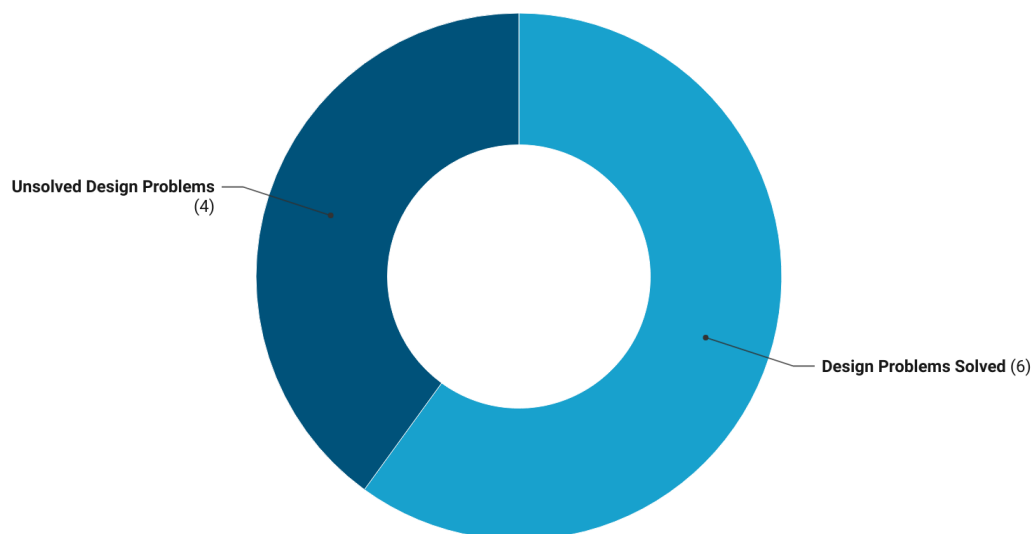


Table 10 exposes, for each design problem, the following metrics: Precision, Recall and F1-Score. One can note that for the design problems solved (1, 2, 3, 6, 7 and 10), the values of

these metrics are the same, as only 1 microservices pattern was considered as label, as explained in Section 3.4. 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. 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.

Table 10 – Metrics per Design Problem.

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%	$\approx 50\%$

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 8. 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.

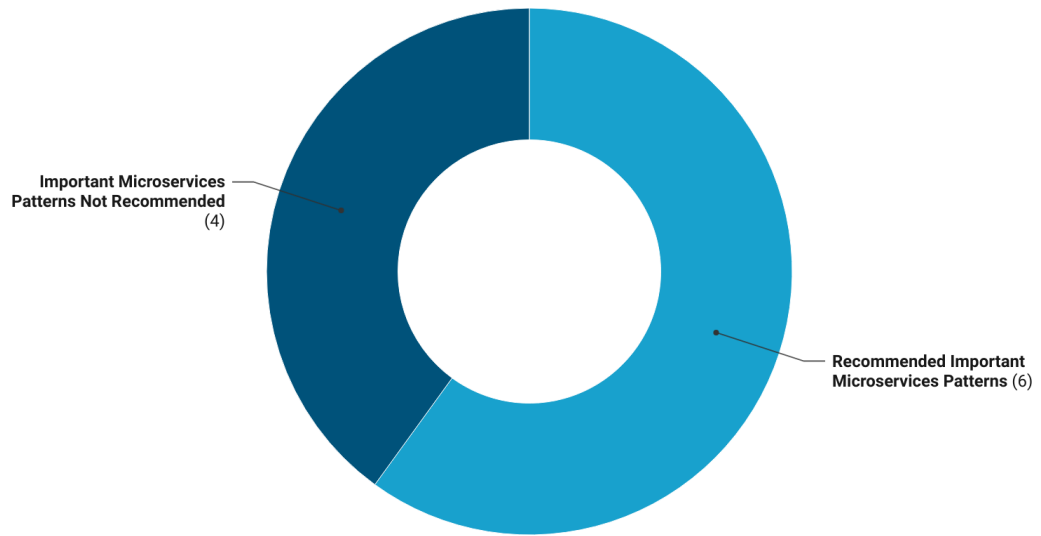
Figure 9 shows a distribution of important recommendations per rank. One can note that 4 of the 6 important microservices patterns were recommended in first and 2 in second. In other words, this means that 40% of these patterns were recommended in first and 20% in second.

Figure 9 – Distribution of Important Recommendations per Rank.



Out of 10 microservices patterns used as labels, 6 were recommended and 4 were not, as Figure 10 shows. In other words, the recommendation approach covered/recommended 60% of these patterns. This indicates that the recommendation approach 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 recommendation approach presented good results.

Figure 10 – Coverage.



# 4

## Floc: A Microservices Patterns Recommendation Tool

In order to provide a way to help developers solve design problems present in microservices-based systems, this work proposes a tool called Floc. This tool is capable of, given a design problem, recommending microservices patterns.

Floc is a web tool that offers a friendly and responsive User Interface (UI), that is, capable of adapting to different devices, such as smartphones, tablets, notebooks and desktops. The tool offers 10 features, this information is detailed in Sections 4.3 and 4.5. This shows that, despite having as its main feature the recommendation of microservices patterns, the tool has additional features.

The tool in question was developed in Java. It is also important to mention that 3 projects that are part of the Spring ecosystem<sup>1</sup> were used to develop this tool, they are: Spring Boot, a facilitator, since it reduces the need to make configurations in Spring-based systems; Spring Data, responsible for enabling access to data; and Spring Security, responsible for providing an authentication and access control structure. In addition to these projects, Thymeleaf, a modern template engine that integrates well with Spring (THYMELEAF, 2023), was also used. Section 4.1 provides more information on how the tool was developed.

This Chapter has 5 more Sections: Section 4.2 presents the tool layers; Section 4.4 explains how the tool was deployed; Section 4.6 explains how the tool was evaluated; Section 4.7 presents the results from the evaluation carried out; and Section 4.8 addresses threats to validity.

### 4.1 Floc Components View

The software system, Floc, is composed of 2 components, as shown in Figure 11. One of them, the database, is responsible for storing relevant data for the system. And the other,

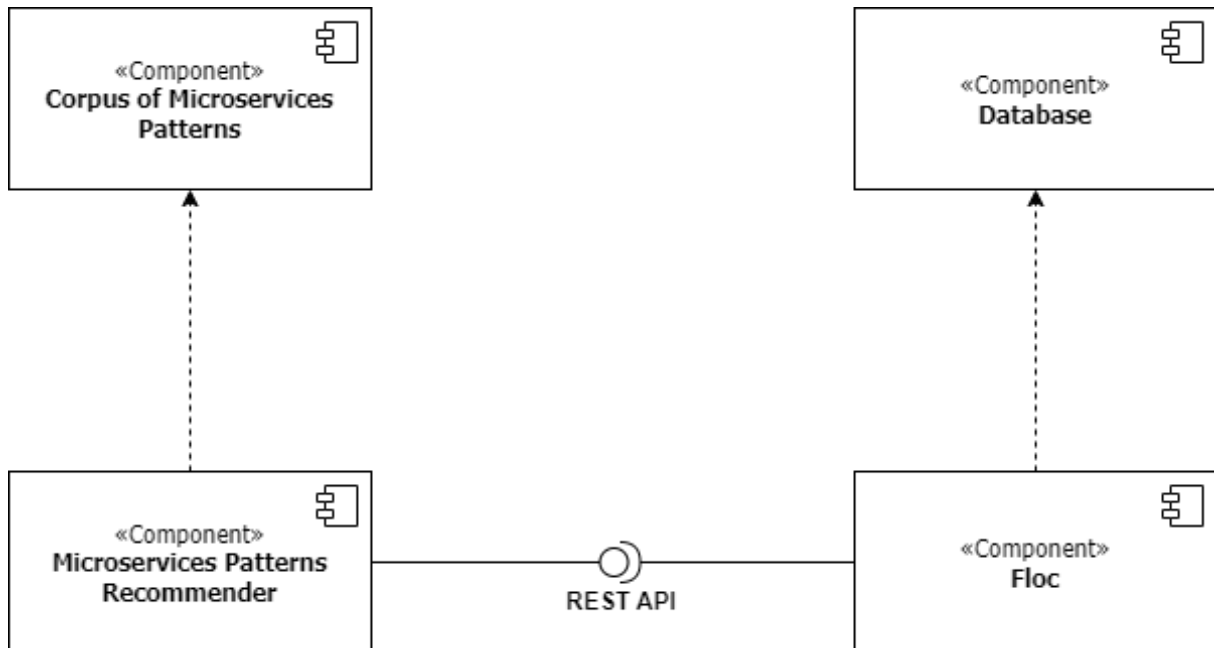
---

<sup>1</sup> The Spring ecosystem provides several projects that drive the development of Java-based systems, as they focus on productivity, simplicity and speed (VMWARETANZU, 2023).



called Microservices Patterns Recommender (MPR), is responsible for actually making the recommendations. Both are explained in detail in this Section. The component called Corpus of Microservices Patterns, already introduced in Section 3.1, is crucial to MPR.

Figure 11 – Floc: Components.



### 4.1.1 Microservices Patterns Recommender

The MPR is a REST API that makes use of the recommendation approach presented in Chapter 3. It is important to mention that the MPR consists of the implementation discussed in the Chapter 3 with the addition of an endpoint that exposes the recommendation of microservices patterns, given a design problem. To develop the REST API, the framework called FastAPI<sup>2</sup> was used.

The cited endpoint is */recommendations*, it can be used through an HTTP request, using GET as a verb. It is crucial to indicate the design problem one wants to solve so that solutions, i.e., microservices patterns, are recommended. The design problem must be indicated in a query parameter named *design\_problem* that must be part of the request. In response, the endpoint returns a list of 3 recommendations, each recommendation is represented by an object that has 2 properties called *rank* and *microservices\_pattern*, it is worth mentioning that these properties have already been explained in Chapter 3. The response format is JSON.

These communication details can be seen in an OpenAPI specification, available in */docs*. The framework, FastAPI, is capable of generating this specification automatically. Figure 12

<sup>2</sup> FastAPI is a modern web framework used for developing API with the programming language “Python” (TIANGOLO, 2023).

exposes the MPR OpenAPI specification. As shown in Figure 12, the MPR OpenAPI specification provides details about the */recommendations* endpoint, including the required HTTP verb, the necessary query parameter, and the expected response.

Figure 12 – MPR OpenAPI Specification.

The screenshot displays the OpenAPI specification for the `GET /recommendations` endpoint. The endpoint is titled "Recommend Microservices Pattern". A required query parameter `design_problem` is defined. The response is a 200 "Successful Response" with a media type of `application/json`. The response body is a JSON array containing one object with the following structure:

```
[
  {
    "rank": 0,
    "microservices_pattern": {
      "id": 0,
      "name": "string",
      "link": "string"
    }
  }
]
```

Figure 13 provides an example illustrating the communication between a client and the MPR. In the example, the client makes a request to 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 Floc communicates with the MPR exactly as explained throughout this Section, so it is a client of the MPR.

### 4.1.2 Floc Database

To create this component, PostgreSQL was the chosen database, which is an open source relational database system that is reliable, in addition to having robust features. As can be seen in Figure 14, the database has four tables: *user*, *design\_problem*, *microservices\_pattern* and *recommendation*, each with its respective columns. The relationships established between the tables are crucial to the functioning of the system. A user can create multiple design problems, and a design problem is associated with a specific user. Furthermore, a design problem receives three distinct recommendations. It is worth mentioning that a recommendation is related to a design problem and a microservices pattern.

Figure 13 – Sequence Diagram: Communication Between a Client and the MPR.

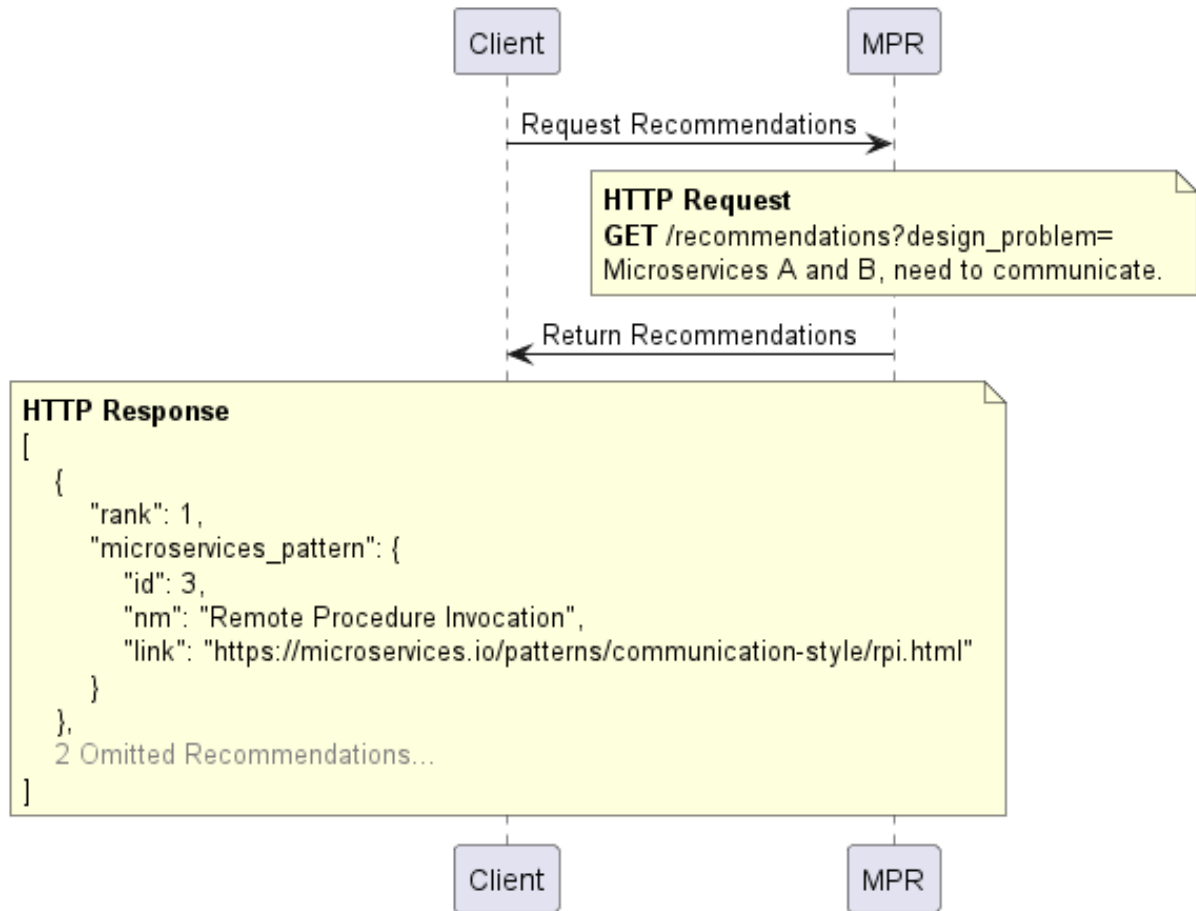
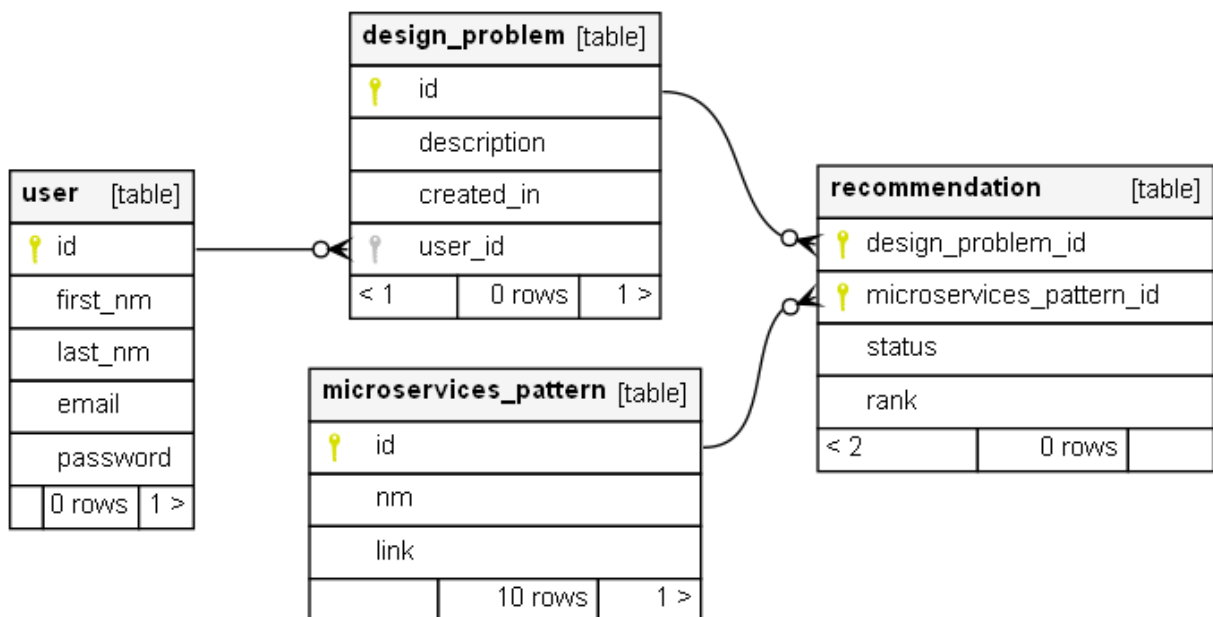


Figure 14 – Database: Entity Relationship Diagram (ERD).



The *user* table has four essential columns. The *id* column serves as the primary key and provides a unique identification for each user in the system. The *first\_nm* and *last\_nm* columns respectively represent the user's first and last name, allowing personal identification. The *email* column stores the unique email address associated with each user, used for authentication purposes. Finally, the *password* column is used to securely store, that is, encrypted, the user's password, ensuring the protection of login information.

The *design\_problem* table also contains four columns. The *id* column is the primary key and assigns a unique identifier to each design problem. The *description* column is used to store the design problem described by the user. The *created\_in* column stores the date the design problem was created, aiding time tracking. The *user\_id* column establishes a relationship with the *user* table, indicating which user is the author of the design problem, allowing problems to be assigned to specific users.

The *microservices\_pattern* table has three columns. The *id* column acts as the primary key and assigns a unique identifier to each microservices pattern. The *nm* column represents the name of the microservices pattern. The *link* column is used to store the pattern's page address on Richardson's website ([RICHARDSON, 2023](#)), allowing direct access to additional information about the microservices pattern, such as documentation or usage examples.

The *recommendation* table has four columns. The *design\_problem\_id* column establishes a relationship with the *design\_problem* table, indicating which design problem is being addressed by the recommendation. The *microservices\_pattern\_id* column relates to the *microservices\_pattern* table, specifying which microservices pattern is being recommended. The *status* column is for the user to indicate if the recommended pattern solves the design problem, partially solves it or does not solve it. Finally, the *rank* column is used to rank the recommendation against other recommendations for the same design problem, providing an ordering that is used for prioritization. It is worth mentioning that the *microservices\_pattern* table receives a data preload for normalization and performance purposes. Thus, the existing microservices patterns in Corpus of Microservices Patterns are previously inserted in this table.

With the database structured in this way, it is possible to store and manage information related to users, design problems, microservices patterns, and recommendations. This structure is highly relevant to the functioning of the tool, Floc.

## 4.2 Floc Implementation View

The tool is divided into 5 layers, namely: view, controller, service, model, and repository. As can be observed, 3 of these layers (model, view, and controller) are part of the Model-View-Controller (MVC) architecture. This widely-adopted architecture pattern aims to segregate the user interface (view) and the domain model (model), ensuring they remain decoupled. This decoupling enhances the adaptability of the interface to changes ([DEACON, 2009](#)). In consideration of the

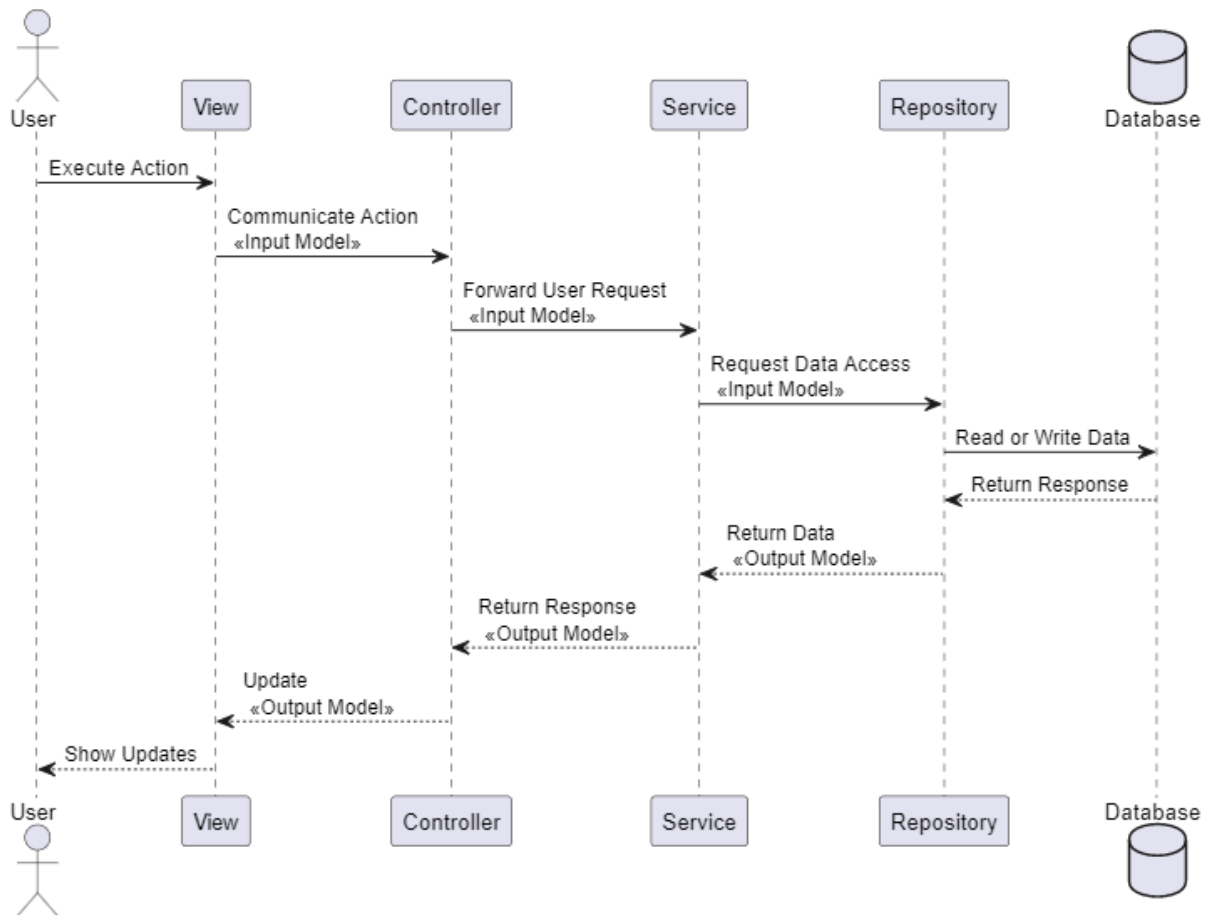
Single Responsibility Principle (SRP), wherein each layer should have a single, specific reason for change (MARTIN, 2017), the remaining 2 layers, service and repository, assume responsibilities abstracted from the model. The responsibility of each of these 5 layers is described as follows:

- **View:** This layer represents the user interface, it is responsible for ensuring the display of information appropriately, as well as communicating the user's actions to the controller;
- **Controller:** Responsible for receiving user requests and making decisions on how to handle these requests, triggering, if necessary, the service to process these requests;
- **Service:** This layer knows about business rules. Its responsibilities include: processing what the controller requests, coordinating transactions to ensure data consistency in the database, and handling exceptions and errors related to business rules. As business rules are centralized in this layer, they can be reused in other parts of the application;
- **Model:** This layer is the core of the application, as it is responsible for providing a set of data structures related to the business domain, which are utilized for communication between different layers;
- **Repository:** Responsible for data access. This layer provides interfaces that enable communication with the database. With this layer it is possible to change the data source without impacting the service layer. It is worth noting that only the service layer interacts with the repository layer.

Figure 15 shows how the layers interact with each other. Starting with the user performing a certain action in the view layer. Then, the view layer communicates to the controller layer about the user's action, and can, if necessary, pass an input model. The controller layer, aware of this action that can be understood as a user request, asks the service layer to process this request. If necessary, the service layer interacts with the repository layer to access the desired data in the database, this means reading or writing data. After this, the service layer returns a response to the controller layer that updates the view layer, ending the interaction with the view layer showing the updates to the user, thus fulfilling the user's initially performed action. The model layer interacts with the other layers, as explained previously, the model layer provides data structures relating to the business that are used in communication between layers.

In addition to the tool having a layered architecture and using concepts from the MVC architecture, the tool also has a monolithic architecture, as all of its functionalities, except for recommending microservices patterns, are centralized in a single deployment unit. This means that the tool was developed as a single program and all its parts share the same database. It is worth mentioning that the tool is integrated with MPR.

Figure 15 – Sequence Diagram: Interaction Between the Layers.



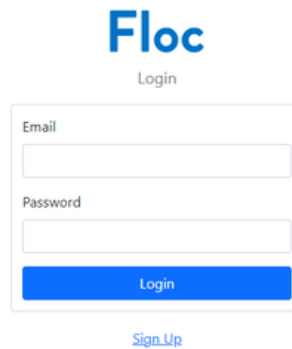
### 4.3 Floc Scenarios of Execution

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 inform an email and password. When registering, the user can log in. The Login and Sign Up pages can be seen respectively in Figures 16 and 17.

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 18 and 19 show the cited 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.

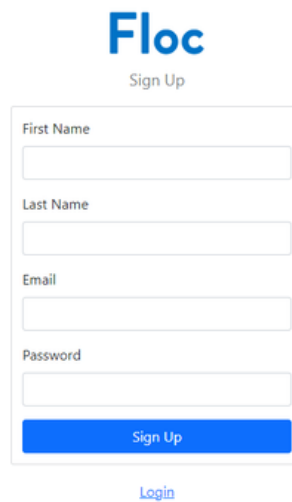
After the user clicks Solve, he/she is redirected to the Design Problem Details page that presents data regarding the registered design problem: Description, Created In and Recommended Microservices Patterns. It is possible to view detailed information about a pattern by clicking the button which has a link icon, it redirects to the exact page on Richardson's Website which explains

Figure 16 – Login Page.



The screenshot shows the login page for Floc. At the top center is the 'Floc' logo in blue. Below it is the text 'Login'. The main form is a white box with a thin border containing two input fields: 'Email' and 'Password'. Below these fields is a blue button with the text 'Login'. Below the form box is a blue link labeled 'Sign Up'.

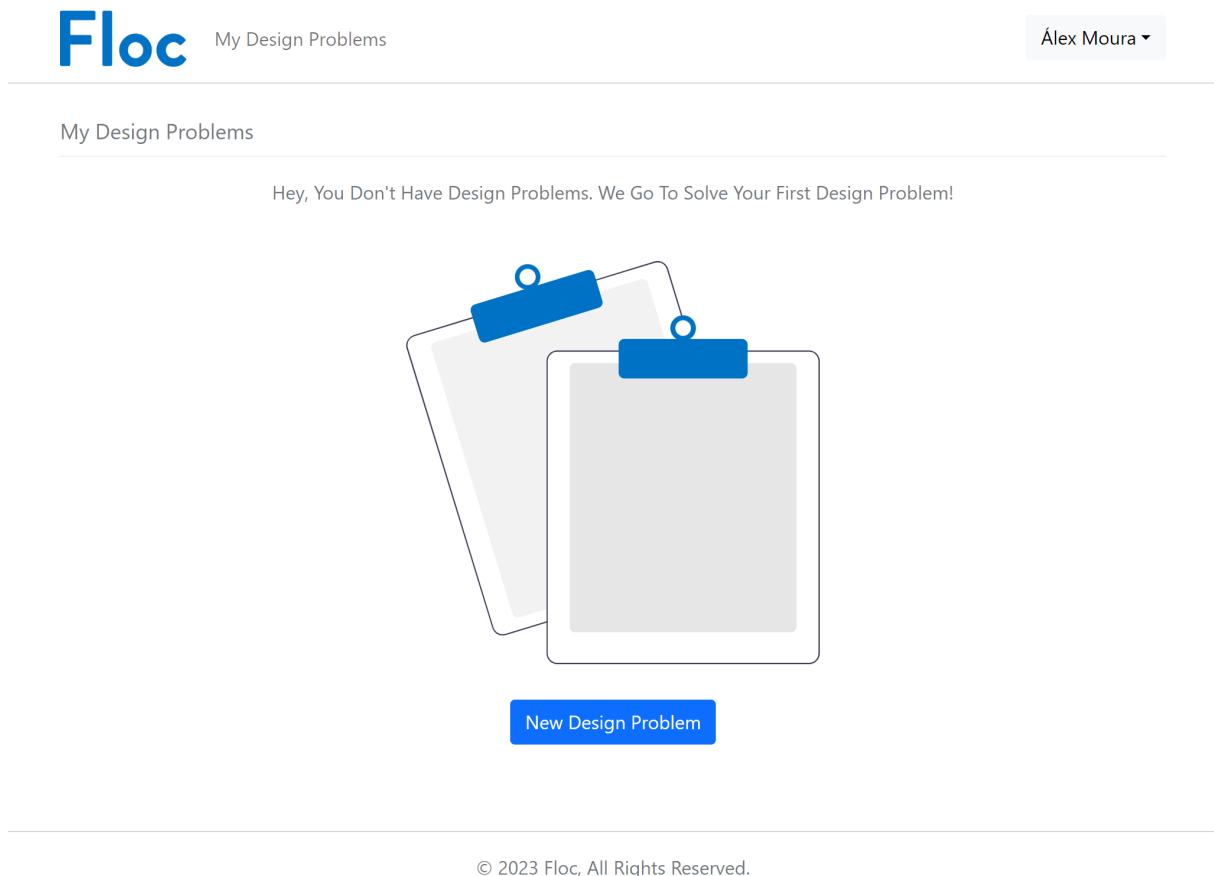
Figure 17 – Sign Up Page.



The screenshot shows the sign up page for Floc. At the top center is the 'Floc' logo in blue. Below it is the text 'Sign Up'. The main form is a white box with a thin border containing four input fields: 'First Name', 'Last Name', 'Email', and 'Password'. Below these fields is a blue button with the text 'Sign Up'. Below the form box is a blue link labeled 'Login'.

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.

Figure 18 – My Design Problems Page.



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. Figures 20 and 21, respectively show the Design Problem Details and Edit Design Problem pages.

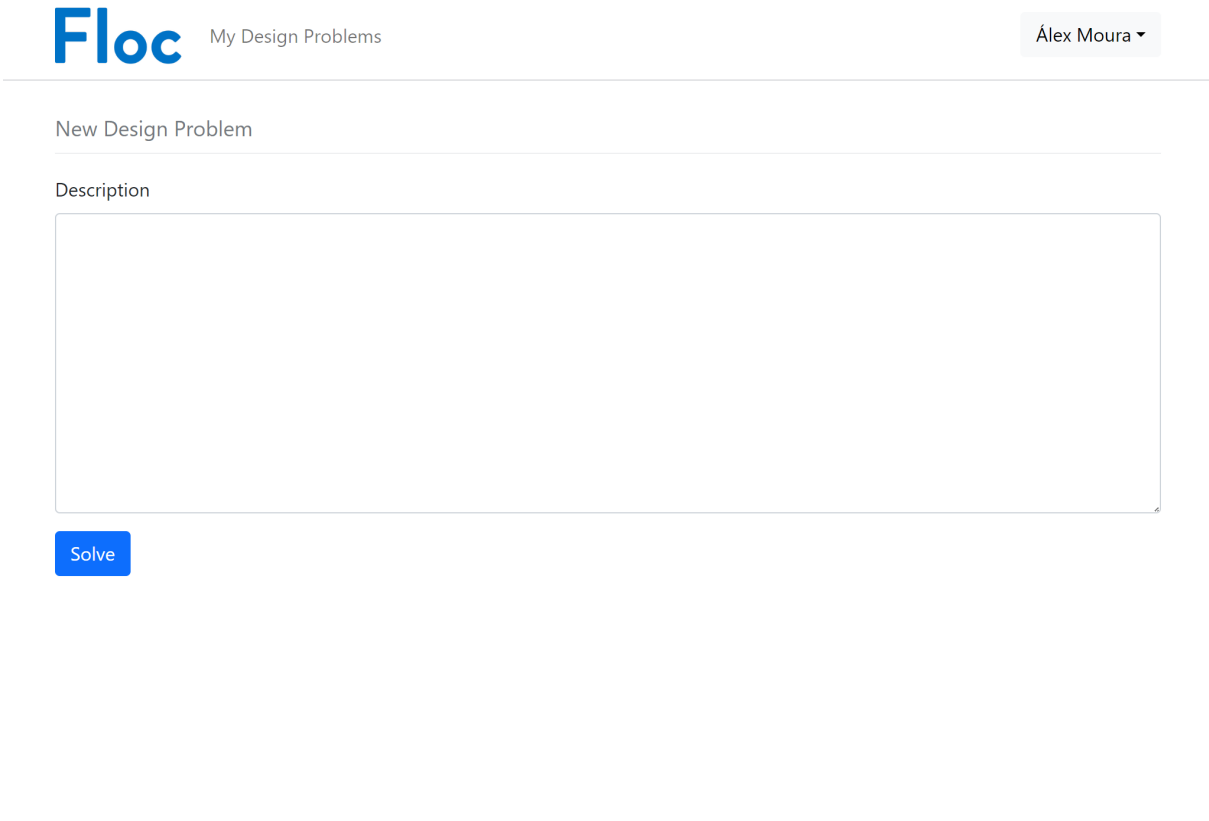
## 4.4 Floc Deployment View

To deploy the components, two servers were used, server 1 and server 2, as shown in Figure 22. On server 1, two components were deployed, the tool (Floc) and the database. To serve Floc, Tomcat (APACHE, 2023), a Web server, was used. As explained in Section 4.1, the tool depends on the database, and as no other component depends on the database, the database was also deployed on server 1 and it is not possible to communicate with it outside of server 1.

On server 2, the MPR was implemented, which depends on the Corpus of Microservices Patterns. The Corpus of Microservices Patterns is embedded in MPR, so it is correct to say that only MPR has been deployed. MPR contains a file called Dockerfile, in this file there are



Figure 19 – New Design Problem Page.



The screenshot shows the 'New Design Problem' page in the Floc application. At the top left is the 'Floc' logo in blue, followed by the text 'My Design Problems'. On the top right, there is a user profile 'Álex Moura' with a dropdown arrow. Below the header is a horizontal line. Underneath, the text 'New Design Problem' is displayed. Below that is a 'Description' label and a large, empty rectangular text input area. At the bottom left of this area is a blue button with the text 'Solve'.

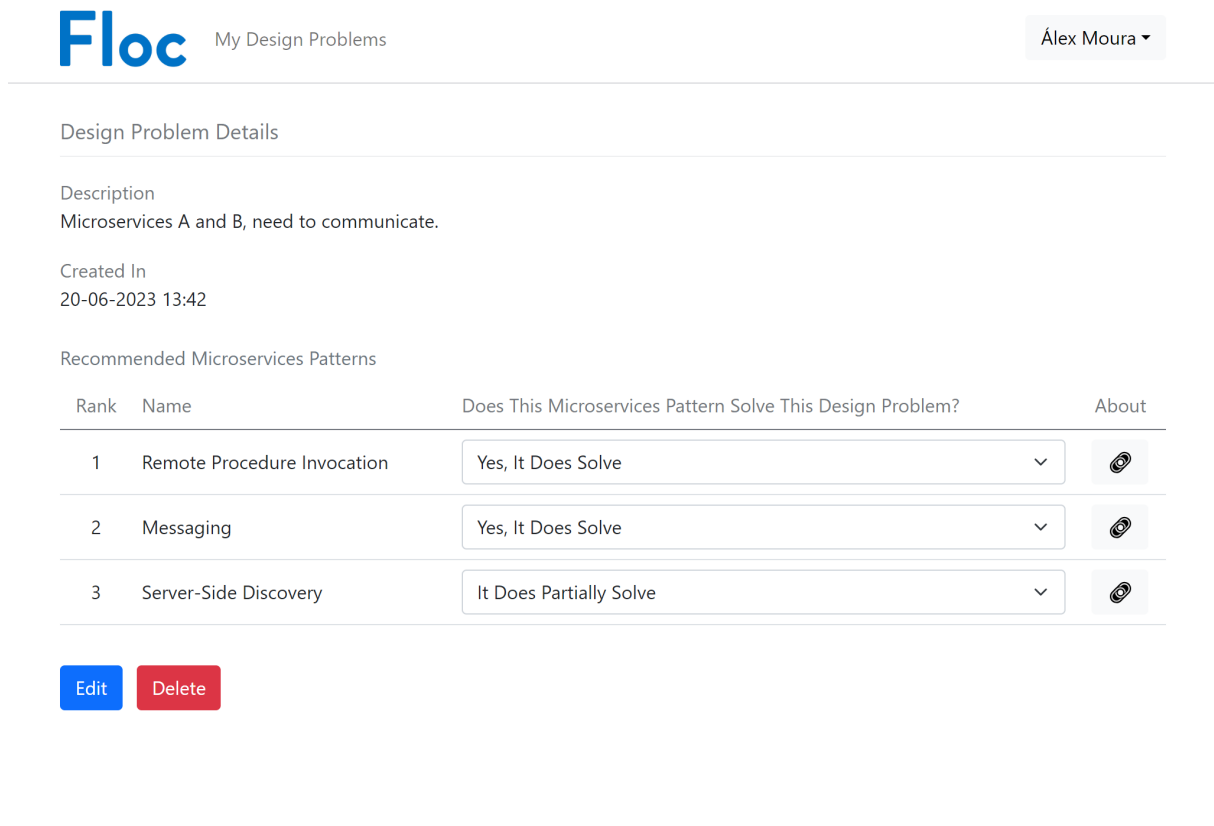
© 2023 Floc, All Rights Reserved.

instructions to create an image with what is needed to run MPR in a Docker container. With this, Docker<sup>3</sup> was installed on server 2 and MPR was placed running in a Docker container. Communication between the Floc and the MPR happens as explained in Section 4.1.

The instructions present in the Dockerfile can be seen in Code 1. In line 1, the official Python 3.10 image is defined as the base image. Base image is an image that serves as a starting point for creating another image. On line 3, an environment variable called PYTHONUNBUFFERED is defined and receives “True” as its value. Line 3 directs Python to display output in the terminal, this makes the terminal useful for viewing logs. On line 4, an environment variable called APP\_HOME is defined and receives “/app” as its value. On line 5, the value of the APP\_HOME variable is defined as the working directory, which is “/app”. After setting the working directory, all subsequent commands are executed in that directory. In line 6, the files and directories present where the Dockerfile is located are copied to the working directory. In other words, on line 6, the MPR is copied to the working directory. On lines 8, 9 and 10, MPR dependencies are installed. Finally, on line 12, the command that is executed when starting a container based on the created image is defined. This command starts a web server called Uvicorn

<sup>3</sup> Docker is a container virtualization platform that simplifies packaging, distributing, and running applications in isolated, portable environments (DOCKER, 2023).

Figure 20 – Design Problem Details Page.



**Floc** My Design Problems Álex Moura ▾




---

Design Problem Details

Description  
Microservices A and B, need to communicate.

Created In  
20-06-2023 13:42

Recommended Microservices Patterns

Rank	Name	Does This Microservices Pattern Solve This Design Problem?	About
1	Remote Procedure Invocation	<input type="text" value="Yes, It Does Solve"/> ▾	
2	Messaging	<input type="text" value="Yes, It Does Solve"/> ▾	
3	Server-Side Discovery	<input type="text" value="It Does Partially Solve"/> ▾	

[Edit](#) [Delete](#)

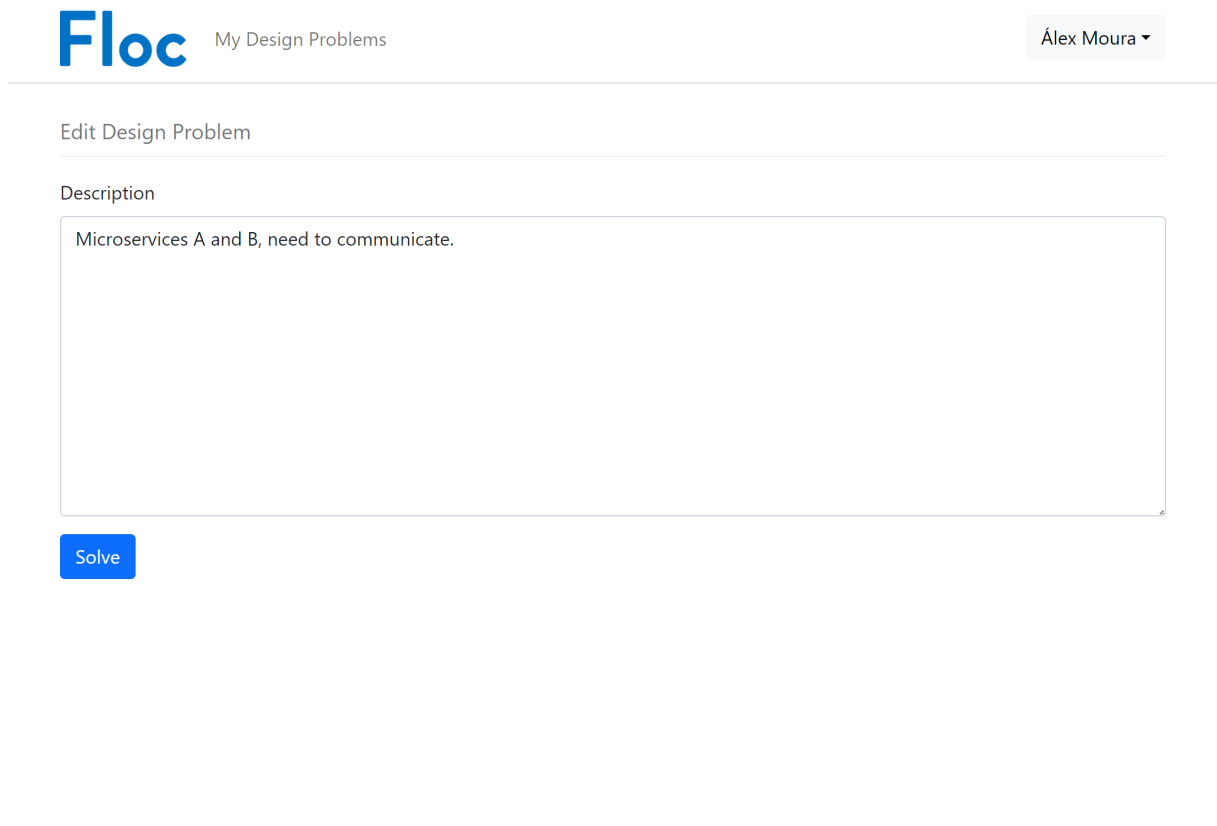
© 2023 Floc, All Rights Reserved.

([ENCODE, 2023](#)), which serves the MPR.

#### Código 1 – MPR: Dockerfile.

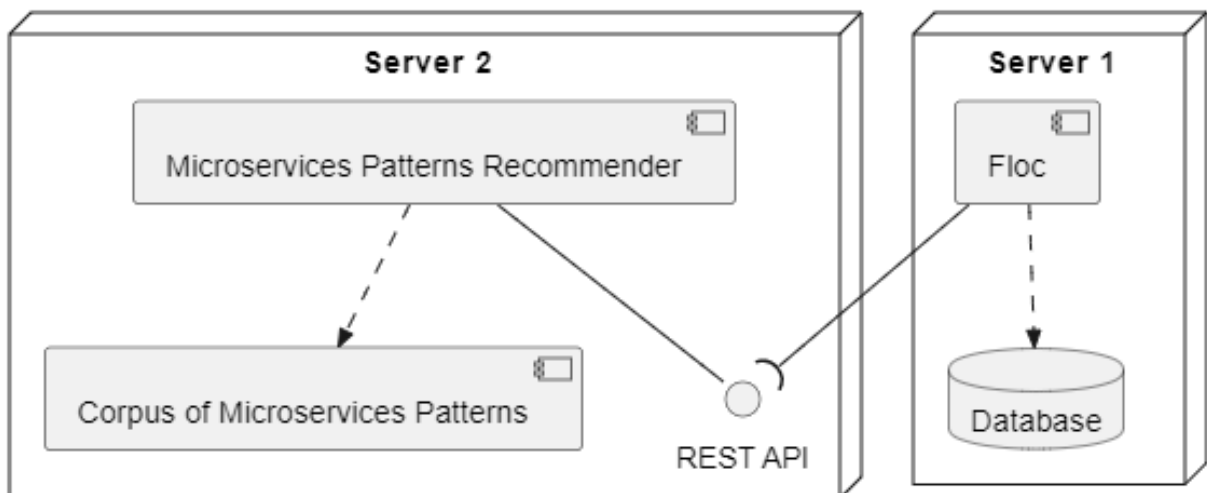
```
1 FROM python:3.10-slim-bullseye
2
3 ENV PYTHONUNBUFFERED True
4 ENV APP_HOME /app
5 WORKDIR $APP_HOME
6 COPY . ./
7
8 RUN pip install --no-cache-dir -r requirements.txt
9 RUN python -m nltk.downloader punkt
10 RUN python -m nltk.downloader stopwords
11
12 CMD ["uvicorn", "main:webapi", "--host", "0.0.0.0", "--port", "8080"]
```

Figure 21 – Edit Design Problem Page.



© 2023 Floc, All Rights Reserved.

Figure 22 – Floc: Deployment Diagram.



## 4.5 Floc Features

In this Section, the tool's features are described, totaling 10 features. Each feature is detailed individually in its own subsection, where its implementation and related classes are

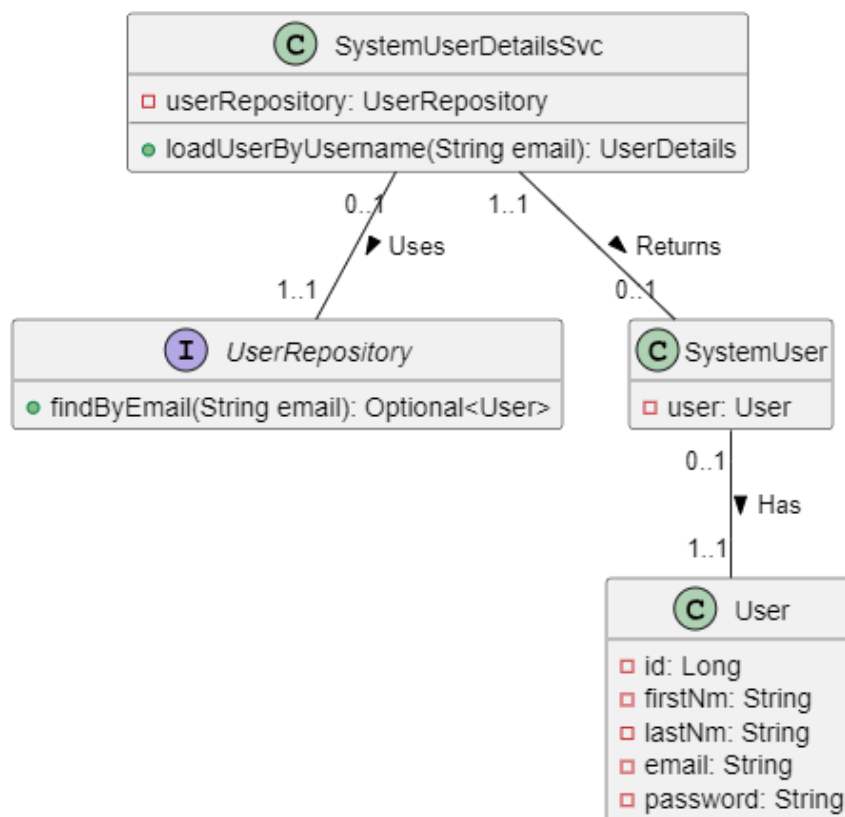
covered. This Section aims to offer a complete understanding of the features of the tool in question.

### 4.5.1 Login

The login functionality was implemented using Spring Security, with customizations to search for the user based on the email provided. If the system does not find a user corresponding to the email provided, it throws an exception called “UsernameNotFoundException”. If the system finds the user with the email provided, it proceeds with password verification. It is important to note that the password entered by the user must match the password registered for that user, Spring Security takes care of this verification. If the user cannot be found by email or the password is incorrect, the system displays an error message on the login page.

Figure 23 presents a class diagram with 3 classes and 1 interface that are responsible for the feature in question. The “SystemUserDetailsSvc” class is composed of the “UserRepository” interface, as the “SystemUserDetailsSvc” class needs to use the “findByEmail” method that this interface offers. The “SystemUserDetailsSvc” class depends on the “SystemUser” class, as the “loadUserByUsername” method returns an instance of that class. The “SystemUser” class is composed of the “User” class.

Figure 23 – Class Diagram: Login.



Customizations for user authentication have been implemented in the service class named `SystemUserDetailsSvc`, which implements the “`UserDetailsService`” interface of Spring Security. When Spring Security receives a login request, it is directed to the “`SystemUserDetailsSvc`” class through the invocation of the “`loadUserByUsername`” method. This overridden method is responsible for fetching a user based on the username, which in this case is the email address.

To fetch the user from the database, the “`SystemUserDetailsSvc`” class utilizes an instance of the “`UserRepository`” interface. Spring Data is responsible for providing this instance, simplifying data access. When the user is found in the database based on the provided email address, the “`loadUserByUsername`” method returns an instance of the “`SystemUser`” class.

The “`SystemUser`” class was created to customize the information stored in the user’s session by Spring Security. The “`SystemUser`” class is a specialization of another class called `User`, which is part of Spring Security. The Spring Security “`User`” class implements an interface called `UserDetails`. It is worth noting that the “`SystemUser`” class is composed of the model class, also named `User`, which contains user information.

## 4.5.2 Sign Up

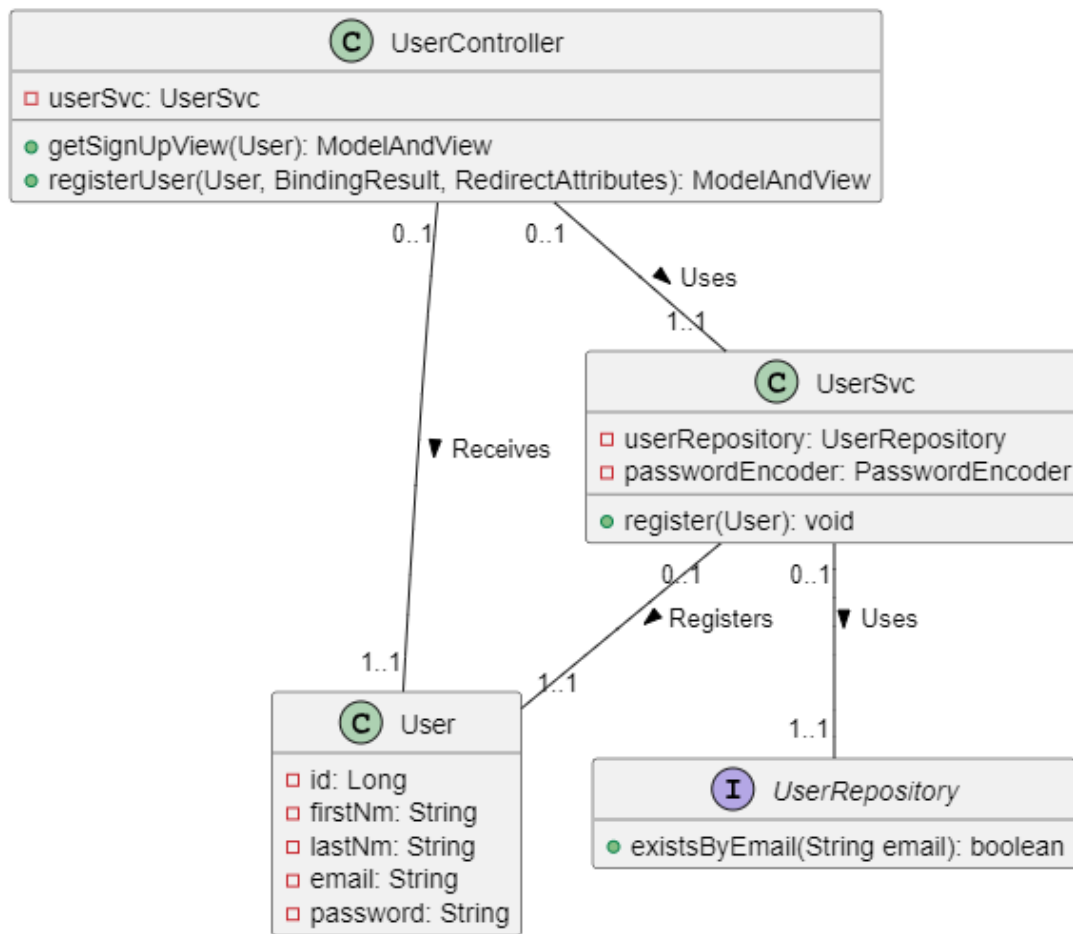
As presented in Section 4.3, to register, the user must provide 4 pieces of information: first name, last name, email, and password. To access the user registration page, the user must click on “Sign Up” on the login page. On the page called Sign Up, as already explained, the user must fill out the registration form with the required information.

Figure 24 shows 3 classes and 1 interface that are related to the “Sign Up” feature, where the “`UserController`” class contains two methods, one of them, called `getSignUpView`, is responsible for returning the “Sign Up” page when the user clicks on “Sign Up” on the “Login” page. The method called `registerUser`, which has a user as one of its parameters, is invoked when the user clicks on the “Sign Up” button present on the registration form.

When the user clicks on the mentioned button, the “`registerUser`” method receives the user who should be registered. Then, the “`registerUser`” method invokes the “`register`” method that belongs to the “`UserSvc`” service class. The “`register`” method knows what must be done to register a user. Firstly, the “`register`” method verifies whether the email provided by the user is already in use. If the email is in use, then an exception called “`EmailIsAlreadyInUseException`” is thrown, resulting in an error message being presented to the user. If the email is not in use, then the password provided by the user is encoded and the user is inserted into the database. It is worth mentioning that validations about, for example, required data, are already done in the controller, before data is passed to the service.

To insert the user into the database, the “`UserSvc`” class uses the “`UserRepository`” interface. Although Figure 24 does not show, for simplification purposes, the “`UserRepository`” interface extends another interface called `JpaRepository` that belongs to Spring Data. The

Figure 24 – Class Diagram: Sign Up.



“JpaRepository” interface offers the save method that can be used to perform this insertion. After inserting the user into the database, the user is redirected to the “Login” page, where a success message is presented.

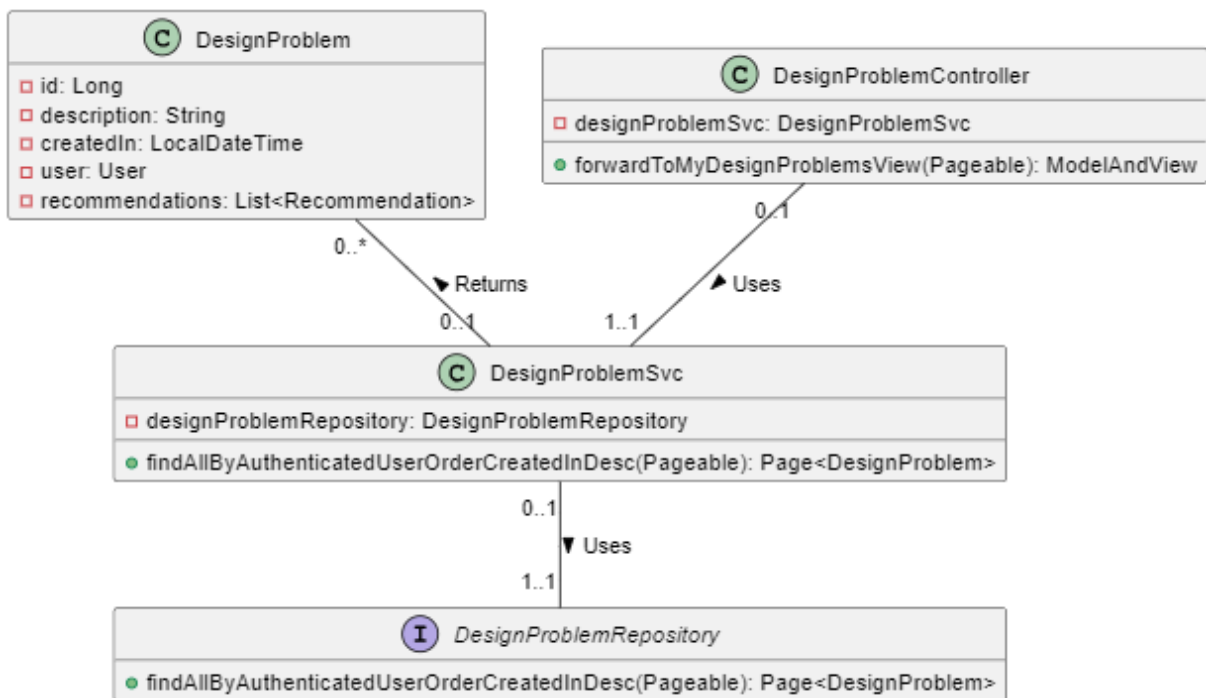
### 4.5.3 List of Design Problems

As explained in Section 4.3, upon logging in, the user is redirected to the “My Design Problems” page, where he/she can view his/her design problems. All design problems reported by the user on the “New Design Problem” page to receive microservices pattern recommendations are displayed on the “My Design Problems” page in the order in which they were created.

Figure 25 presents 3 classes and 1 interface that have a relationship with the feature in question, where the “DesignProblemController” class holds the “forwardToMyDesignProblemsView” method which is responsible for returning the “My Design Problems” page with the design problems that belong to the user. The “forwardToMyDesignProblemsView” method receives a single parameter, which is used to page the design problems that belong to the user. This parameter is of type “Pageable”. Pageable is an interface that Spring offers to help implement

paged queries.

Figure 25 – Class Diagram: List of Design Problems.



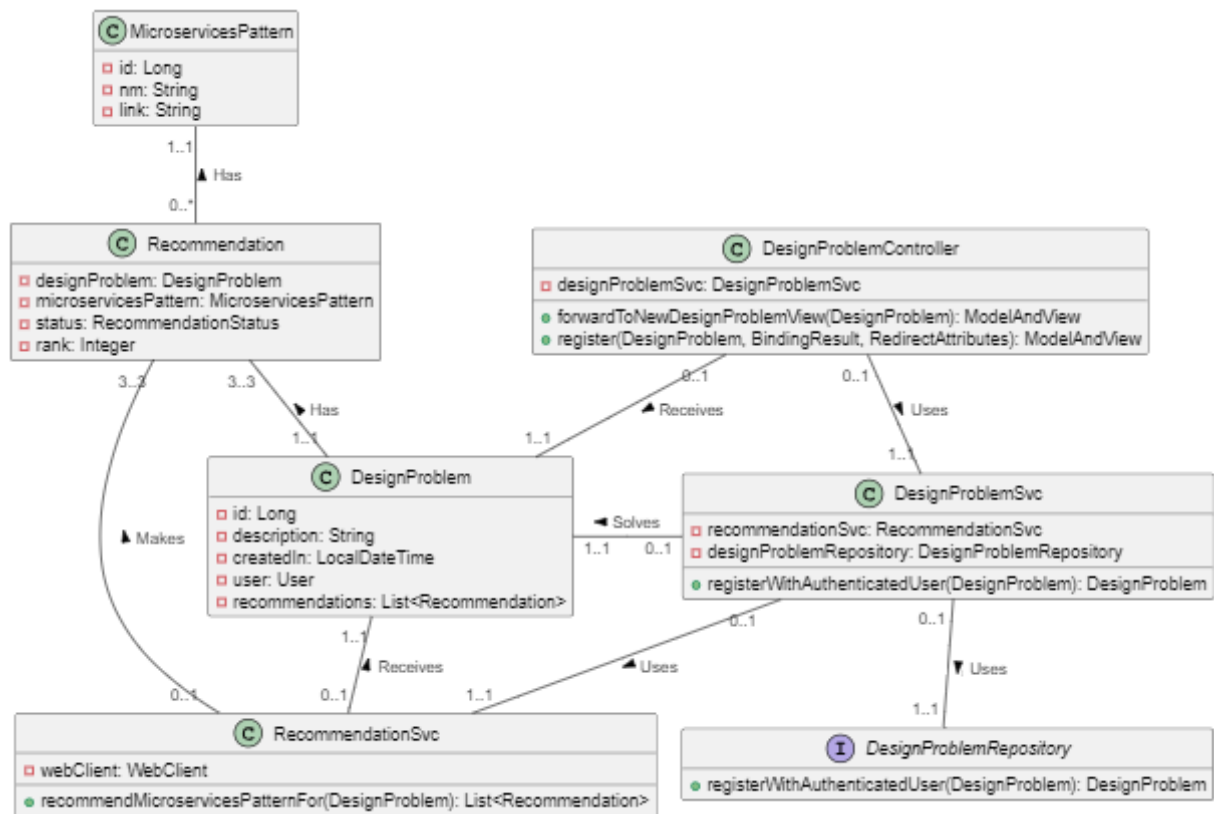
The “forwardToMyDesignProblemsView” method invokes the “findAllByAuthenticatedUserOrderCreatedInDesc” method that belongs to the “DesignProblemSvc” service class. The “findAllByAuthenticatedUserOrderCreatedInDesc” method receives the Pageable and interacts with an instance of the “DesignProblemRepository” interface to obtain the design problems that belong to the logged user. After obtaining these design problems, the method returns a design problem page with a maximum of 9 problems. Although a design problem has 1 user and a recommendation list, for the “List of Design Problems” feature it is not important that design problems have these two attributes fulfilled.

The design problem page returned by the “findAllByAuthenticatedUserOrderCreatedInDesc” method is used by the “forwardToMyDesignProblemsView” method to update the “My Design Problems” page, so the user is able to view their design problems.

#### 4.5.4 Solve Design Problem

As explained in Section 4.3, given a design problem, the user can request microservices pattern recommendations on the “New Design Problem” page. Figure 26 shows the 6 classes and 1 interface that are related to this feature. The “forwardToNewDesignProblemView” method of the “DesignProblemController” class is responsible for returns to the Client, i.e., the browser, the “New Design Problem” page. When the user clicks the “Solve” button, the “register” method that belongs to “DesignProblemController” class is invoked.

Figure 26 – Class Diagram: Solve Design Problem.



The “register” method that belongs to the “DesignProblemController” class is responsible for receiving the design problem described by the user and invoking the “registerWithAuthenticatedUser” method that belongs to the “DesignProblemSvc” class, passing the design problem. The “registerWithAuthenticatedUser” method, of the “DesignProblemSvc” class, invokes the “recommendMicroservicesPatternFor” method that belongs to the “RecommendationSvc” class.

As the method name suggests, the “recommendMicroservicesPatternFor” method is responsible for obtaining microservices pattern recommendations for the problem. This method obtains these recommendations by communicating with the MPR, this communication occurs as explained in Subsection 4.1.1. To make requests to the MPR, the “recommendMicroservicesPatternFor” method uses the HTTP client provided by Spring WebFlux, this client is called WebClient. The “RecommendationSvc” class has a WebClient as an attribute. After obtaining the recommendations, the “recommendMicroservicesPatternFor” method returns a “List” with these recommendations.

After the “registerWithAuthenticatedUser” method, of the “DesignProblemSvc” class, receives the recommendations, it attributes them to the “recommendations” attribute of the design problem. After that, the “registerWithAuthenticatedUser” method that belongs to the “DesignProblemRepository” interface is invoked to insert the design problem and recommendations into the database.

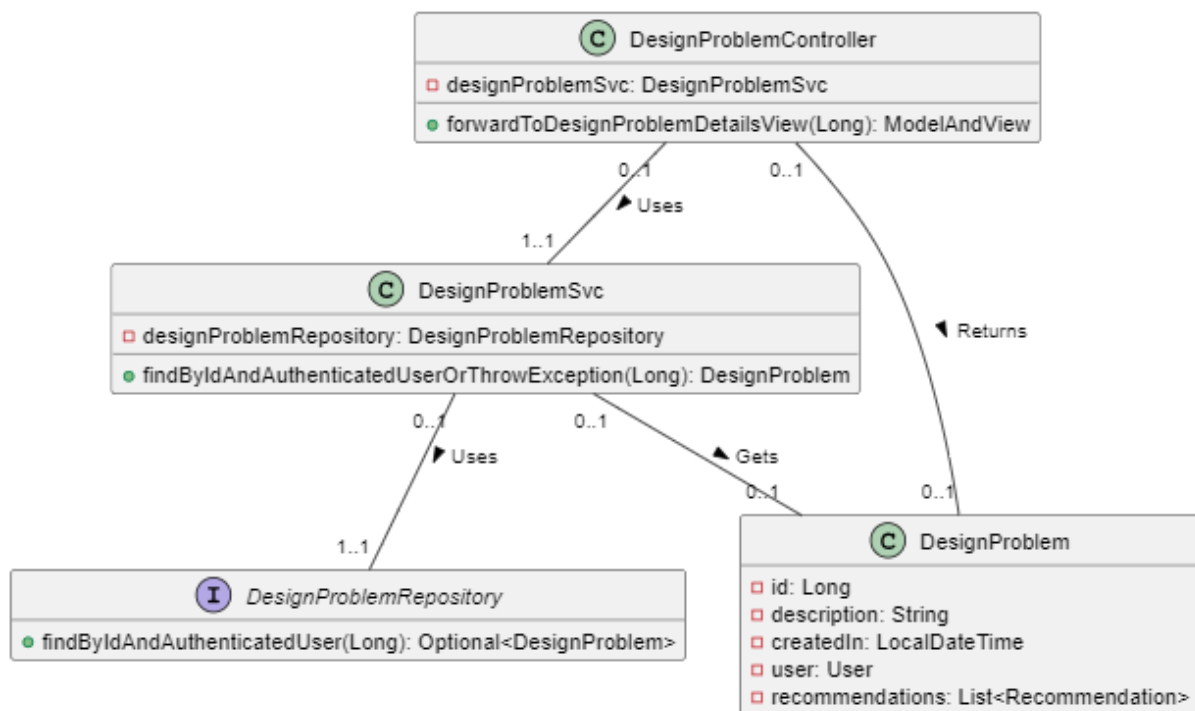


After database insertions occur, the design problem that was inserted is returned to the “register” method with its id, which redirects the user to the “Design Problem Details” page and presents a success message, indicating that recommendations were made for the design problem described by the user.

#### 4.5.5 View Design Problem Details

The user can see the details of a design problem on the “Design Problem Details” page. The user can access this page by using the “Solve Design Problem” feature or by clicking on a design problem on the “My Design Problems” page. Figure 27 presents 3 classes and 1 interface responsible for the “View Design Problem Details” feature.

Figure 27 – Class Diagram: View Design Problem Details.



The “forwardToDesignProblemDetailsView” method of the “DesignProblemController” class is responsible for receiving the user request, invoking the “findByIdAndAuthenticatedUserOrThrowException” method that belongs to the “DesignProblemSvc” class to obtain the design problem that the user wants to see the details and then return to the “Design Problem Details” page presenting the obtained design problem data.

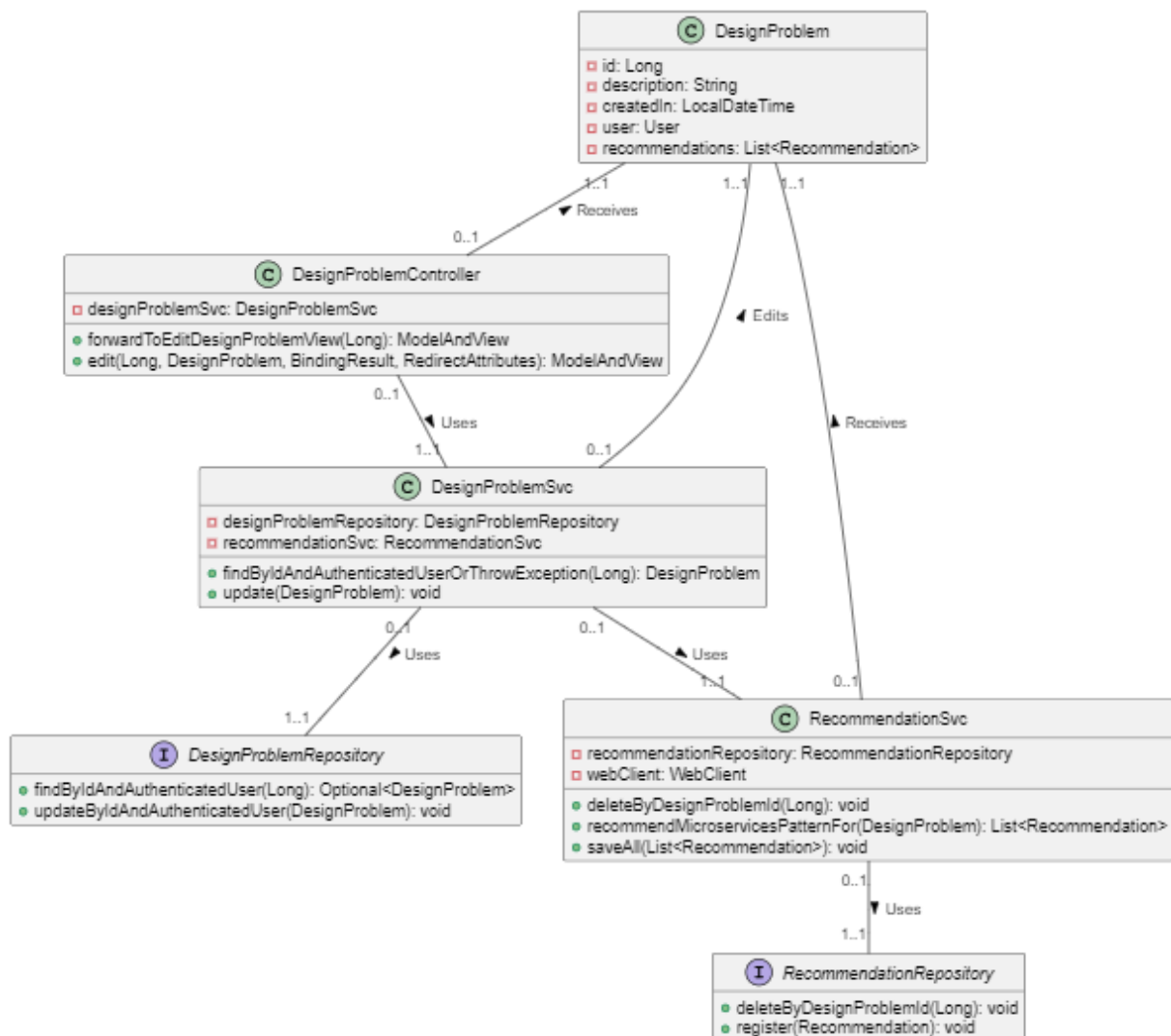
The “findByIdAndAuthenticatedUserOrThrowException” method interacts with the “findByIdAndAuthenticatedUser” method of the “DesignProblemRepository” interface to search the database for the design problem, including its recommendations. If the design problem is not found, the method “findByIdAndAuthenticatedUserOrThrowException” throws an “DesignProblemNotFoundException”. As the exception name suggests, it indicates that the design problem

was not found, therefore, the method “forwardToDesignProblemDetailsView” informs the user. If the design problem is found, then the “forwardToDesignProblemDetailsView” method returns the “Design Problem Details” page presenting the design problem data.

### 4.5.6 Edit Design Problem

It is possible to edit a design problem on the “Design Problem Details” page by clicking the “Edit” button. To understand how this feature was implemented, it is important to observe Figure 28. When the user clicks on the mentioned button, the “forwardToEditDesignProblemView” method, that belongs to the “DesignProblemController” class, is invoked. This method is responsible for redirecting the user to the “Edit Design Problem” page.

Figure 28 – Class Diagram: Edit Design Problem.



Before redirecting, the “forwardToEditDesignProblemView” method gets the design problem that the user wants to edit to fill the “Description” field with the current description of the design problem. It is worth mentioning that the “Edit Design Problem” page was

presented in Section 4.3. To get the design problem, the mentioned method invokes the “`findByIdAndAuthenticatedUserOrThrowException`” method of the “`DesignProblemSvc`” class, which invokes the “`findByIdAndAuthenticatedUser`” of the “`DesignProblemRepository`” interface.

Thus, the “`findByIdAndAuthenticatedUser`” method fetches the design problem from the database and returns to the “`findByIdAndAuthenticatedUserOrThrowException`” method, which returns to the “`forwardToEditDesignProblemView`” method. Thus, the “`forwardToEditDesignProblemView`” method redirects the user to the “Edit Design Problem” page and exposes the current design problem description.

When entering the “Edit Design Problem” page and changing the design problem description, the user must click the “Solve” button which invokes the “`edit`” method that belongs to the “`DesignProblemController`” class. This method is responsible for invoking the “`update`” method of the “`DesignProblemSvc`” class, which serves to update the design problem, and redirect the user to the “Design Problem Details” page with the updated design problem.

To update the design problem, firstly the “`update`” method invokes the “`updateByIdAndAuthenticatedUser`” method of the “`DesignProblemRepository`” interface, which is responsible for updating the design problem description in the database. Secondly, the “`update`” method invokes the “`deleteByDesignProblemId`” method of the “`RecommendationSvc`” class, which invokes the “`deleteByDesignProblemId`” method of the “`RecommendationRepository`” interface to delete the current recommendations from the design problem in the database. Thirdly, the “`update`” method invokes the “`recommendMicroservicesPatternFor`” method of the “`RecommendationSvc`” class to obtain new recommendations for the design problem. Finally, the “`update`” method invokes the “`saveAll`” method of the “`RecommendationSvc`” class to insert the new recommendations made for the design problem into the database. It is worth mentioning that the “`saveAll`” method makes the insertions by invoking the “`register`” method of the “`RecommendationRepository`” interface.

After updating the design problem and its recommendations, the “`edit`” method redirects the user to the “Design Problem Details” page, where he/she can see the updated information, as well as a success message informing that the design problem was updated successfully.

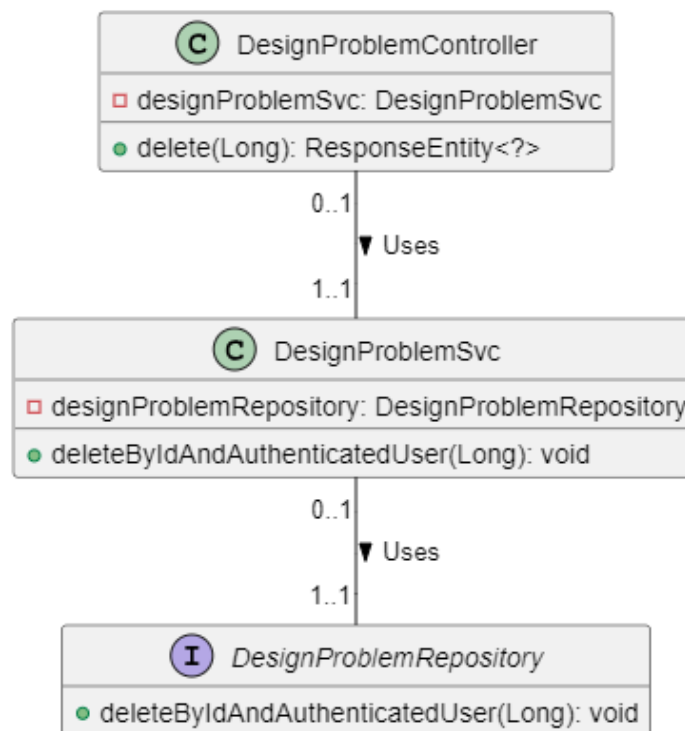
### 4.5.7 Delete Design Problem

On the “Design Problem Details” page, it is possible to delete the design problem, as presented in Section 4.3. To delete the design problem, in the view called “Design Problem Details” page, there is a JavaScript code that is responsible for making an Asynchronous HTTP Request to the server. This request is made to “`/design-problems/{id}`” and the HTTP verb used is “`DELETE`”. This snippet “`{id}`” is replaced by design problem id which should be deleted.

Figure 29, shows 2 classes and 1 interface that are responsible for the “Delete Design Problem” feature. The “`delete`” method, which belongs to the “`DesignProblemController`” class, is responsible for fulfilling the mentioned request. The “`delete`” method invokes the “`deleteByI-`

dAndAuthenticatedUser” method of the “DesignProblemSvc” service class, which interacts with the “deleteByIdAndAuthenticatedUser” method that belongs to the “DesignProblemRepository” interface to delete the design problem in the database. The 3 methods mentioned receive a parameter of type “Long”. This parameter is the design problem id that is passed between the methods. An important rule is that the design problem can only be deleted by the user who reported it.

Figure 29 – Class Diagram: Delete Design Problem.



After deleting the design problem in the database, the “delete” method of the “Design-ProblemController” class returns an HTTP Response with status code “204” which indicates that the design problem was successfully deleted and no content was returned by the server. If the design problem was not reported by the logged user, an HTTP Response is returned with status code “404” which indicates that the design problem was not found. Upon receiving the response, in case of success, the view redirects the user to the “My Design Problems” page and displays a success message. In case of error, the view presents an error message to the user.

#### 4.5.8 Feedback for a Recommended Microservices Pattern

On the “Design Problem Details” page, in the table where the recommended microservices patterns are presented, there is a column with the title “Does This Microservices Pattern Solve This Problem?”, for each row in this table, there is a selection field in this column. This field contains 3 options, they are: 1) Yes, It Does Solve, 2) It Does Partially Solve and 3) It Does Not

Solve. As explained in Section 4.3, the user can give a feedback for each recommendation by choosing one of these 3 options.

When choosing one of the mentioned options, an Asynchronous HTTP Request is made to “/{designProblemId},{microservicesPatternId}/modify-status”, using the HTTP verb “PUT”. The “{designProblemId}” snippet is replaced by the design problem id and the “{microservicesPatternId}” snippet is replaced by the microservices pattern id, these two data make up the recommendation id. In addition to this data, the request body contains, in JSON format, the recommendation with its new status/feedback and its id.

Figure 30 shows 3 classes, 1 interface and 1 enum that are part of this feature. The “modifyStatus” method that belongs to the “RecommendationController” class, is responsible for receiving the mentioned request. The “modifyStatus” method of the “RecommendationController” class receives 3 parameters, the first is the design problem id, the second is the microservices pattern id and the third is the recommendation sent in the request body. What this method does is include the id in the recommendation and invoke the “modifyStatus” method of the “RecommendationSvc” class to update the recommendation status.

The “modifyStatus” method of the “RecommendationSvc” class interacts with the “modifyStatus” method that belongs to the “RecommendationRepository” interface to update the recommendation status in the database. It is worth mentioning that the recommendation status is represented by the “status” attribute present in the “Recommendation” class model. The type of this attribute is RecommendationStatus which is an enum. The “RecommendationStatus” enum contains the 3 statuses (SOLVED, PARTIALLY\_SOLVE and UNSOLVED) that the user can choose.

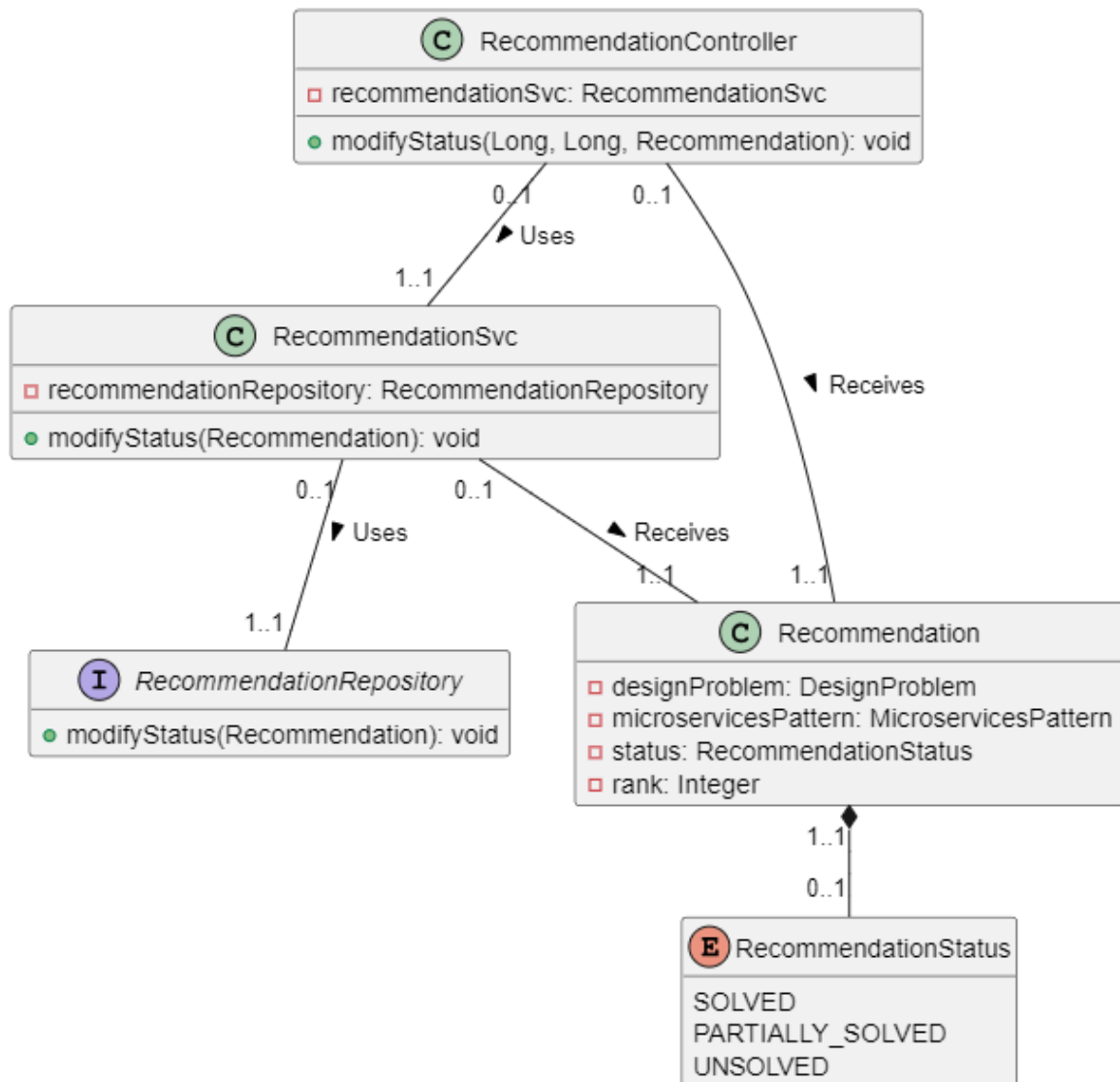
After updating the recommendation status, the “modifyStatus” method of the “RecommendationController” class, returns an HTTP Response with the status code “200”, indicating that the feedback was given successfully. Thus, the view presents a success message.

#### 4.5.9 See More Information About a Microservices Pattern

As described in Section 4.3, on the “Design Problem Details” page, it is possible to access more information about the recommended microservices patterns. To do so, simply click the button with a link icon, located in the last column of the table where the recommended microservices patterns are displayed. Upon clicking this button, the user is redirected to a page on the Richardson website (RICHARDSON, 2023), where information about the pattern is available.

The implementation of this feature is simple because each microservices pattern includes an attribute called “link” that represents the address of the corresponding page. Therefore, it is only necessary to instruct the button that, when clicked, it should redirect the user to this address. It is worth noting that when accessing the “Design Problem Details” page, the recommended patterns are loaded, as explained in Subsection 4.5.5.

Figure 30 – Class Diagram: Feedback for a Recommended Microservices Pattern.



### 4.5.10 Logout

To do “Logout”, the user must click on the drop-down menu that presents their name, then the next step is to click on the “Logout” option. Thus, “Logout” is done and the user is redirected to the “Login” page. The “Logout” feature was implemented with Spring Security. When the user clicks “Logout”, a request is made to “/logout”, and Spring Security deletes the user session. This feature is simple to implement with Spring Security.

## 4.6 Floc Evaluation

To evaluate the tool, it was published so that tests could be carried out in a company that offers integrated and innovative cyber security and infrastructure solutions. In 2022, this

company was classified, by MSSP Alert (MSSP, 2023), as one of the 40 best security companies in the world. Throughout this dissertation, this company is called by the fictitious name of ASM.

Three developers from ASM 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 11 shows the time of experience, in years, that each developer has with software development and systems based on microservices.

Table 11 – Time of Experience of Developers Involved in the Tests.

<b>ID</b>	<b>Time of Experience in Software Development (Years)</b>	<b>Experience Time in Microservices-Based Systems (Years)</b>
D1	29	7
D2	23	3
D3	12	1

After testing, the developers involved were individually interviewed. A total of 9 questions were asked, which can be found in Table 12, and they pertained to the tool. The interviews were essential to collect feedback from each developer and information that could contribute to improving the tool.

Table 12 – Questions Asked in the Interviews.

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

## 4.7 Results

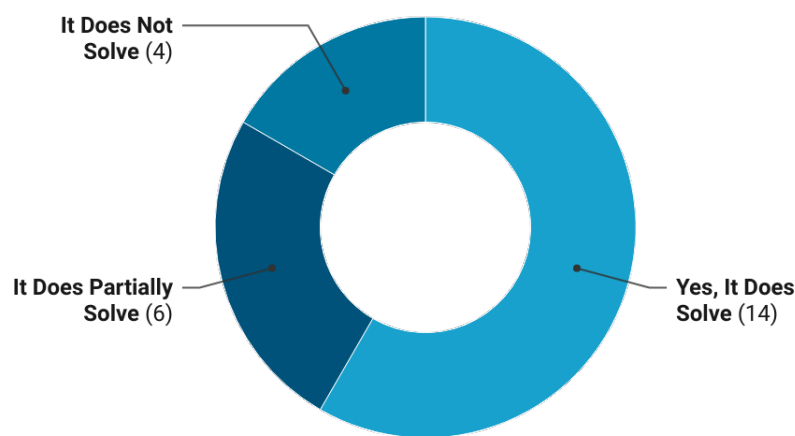
This section presents the results of the tests and interviews, as explained in Section 4.6. Subsection 4.7.1 provides the results of the tests conducted with industrial design problems, while Subsection 4.7.2 presents the responses obtained from the interviews with the developers involved in the tests.



### 4.7.1 Tests with Industrial Design Problems

As explained in Section 4.6, tests were performed at ASM with industrial design problems, where 3 developers have requested recommendations for existing design problems in certain software products, using Floc. After getting the recommendations, they labeled 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 which was presented in Section 4.3.

Figure 31 – Distribution of Recommended Microservices Patterns per Label.



In total, eight design problems were reported by the developers. Thus, 24 microservices patterns recommendations were made, given that for a design problem, 3 patterns are recommended, as explained in Subsection 4.1.1. Considering the labels already mentioned, it is possible to see in Figure 31 that 14 recommended patterns were labeled “Yes, It Does Solve”, 6 were labeled “It Does Partially Solve” and 4 were labeled “It Does Not Solve”.

With the intention of presenting some examples of industrial design problems reported during the tests performed and providing an idea of how these problems were described, below it is possible to view three of the eight industrial design problems reported by the developers:

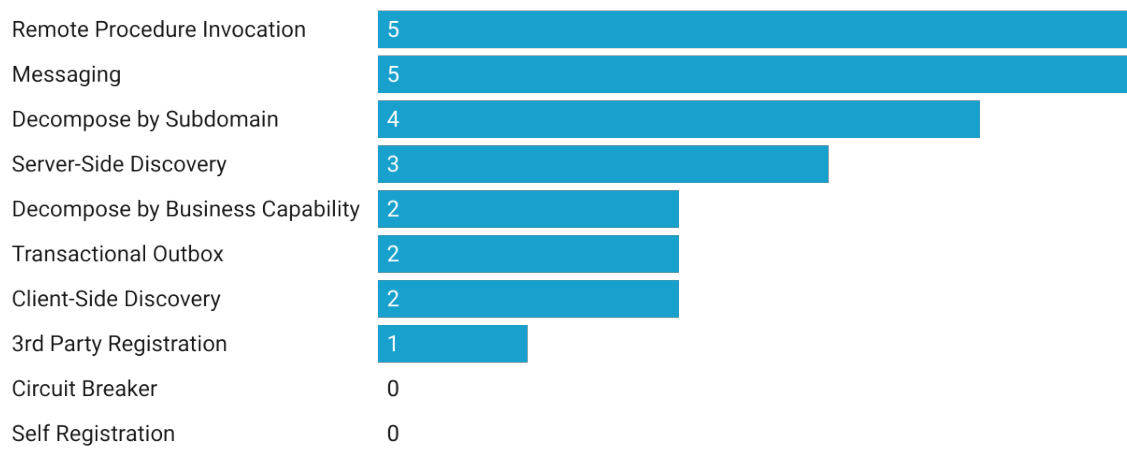
- **Industrial Design Problem 1:** In a microservices architecture, it is common for services to need to communicate with each other to fulfill requests from clients. However, if the communication between microservices is not optimized, it can lead to performance issues and slow down the entire system. 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 32, only two patterns were not recommended, Circuit Breaker and Self Registration. 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.

Figure 32 – Distribution of Recommendations per Microservices Pattern.



Data presented in this Subsection indicate that more than 80% of the recommended patterns were helpful in dealing with the problems. Although four recommended patterns were labeled “It Does Not Solve”, 100% of the problems received recommendations for 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.

#### 4.7.2 Interviews with Developers

After running the tests at ASM, the three 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 Section. In general, the responses were positive and revealed improvements that can be made in future research.

About **Q1**, it was pointed out by D1 that the tool is useful in a context where it is necessary to develop new functionalities, as well as maintaining existing functionalities. This developer 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.

Regarding **Q2**, the three interviewees stated that the tool can help developers to 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 and D2 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 three 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 there are patterns recommendations that 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. D3 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 patterns 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. In view of this, D2 stated that the tool can improve developers' productivity and communication between them.

Regarding **Q6**, D1 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.

About **Q7**, D1 reported as an improvement the expansion of the CMP. D2 pointed, out as an improvement, the display of statistics with the aim of helping 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.

Regarding **Q8**, D1 pointed out that he 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 and D2 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 to understand the information presented and suggested the inclusion of explanations.

## 4.8 Threats to Validity

According to Ampatzoglou et al. (AMPATZOGLOU et al., 2019), threats to validity are aspects that compromise research credibility. Thus, aiming to guarantee the credibility of this work, it was necessary to mitigate these aspects. As reported by Zhou et al. (ZHOU et al., 2016), it is interesting to ensure: Construct Validity, Internal Validity, External Validity, and Conclusion Validity.

### 4.8.1 Construct Validity

This research requires understanding about the microservices architecture, microservices patterns, and information retrieval. This understanding is given in Chapter 2, where it is also possible to observe how important it is to help developers in the selection of microservices patterns.

### 4.8.2 Internal Validity

This work was conducted by 3 researchers. To ensure that the results are free from bias, the tests and interviews carried out were analyzed and evaluated by the 3 researchers. It is important to mention that a presentation about the tool was made to the developers involved in the tests, ensuring understanding of its use.

### 4.8.3 External Validity

Regarding the generalization of the results, tests were initially carried out using a collection of toy design problems. Then, tests were carried out using industrial design problems in a Cybersecurity company. Testing using industrial design problems was carried out by three experienced developers who were interviewed individually to ensure responses were free from bias. The results of the tests and interviews are useful in understanding how to apply this research in other environments or contexts.

### 4.8.4 Conclusion Validity

This research can be replicable considering Section 1.4 which explains the methodology used to develop this work and Chapters 3 and 4 which respectively explain, in detail, the recommendation approach and the Floc tool.

# 5

## Conclusions, Contributions and Future Works

### 5.1 Conclusions

The general objective of this work was to provide a means of helping developers to select microservices patterns to solve design problems present in systems based on microservices architecture. Aiming to achieve the general objective, two specific objectives were defined:

- **Specific Objective 1:** Propose a microservices patterns recommendation approach;
- **Specific Objective 2:** Develop a tool that makes use of this approach for developers to use.

The research methodology used in this work was Design Science Research (DSR) which proposes the development of Information Technology (IT) artifacts to solve real-world problems (HEVNER et al., 2010). In summary, the final result of this work is a tool called Floc that allows developers to manage design problems and obtain microservices patterns recommendations. Thus, the tool is the IT artifact that was developed to solve the difficulty that developers may experience when selecting microservices patterns to solve design problems present in systems based on microservices architecture.

About **specific objective 1**, it was proposed a recommendation approach based on Information Retrieval (IR) that is capable of receiving a design problem and recommending 3 microservices patterns that can solve it. This approach was implemented and evaluated through tests carried out with toy design problems, using metrics.

Each toy design problem was labeled with a microservices pattern that can solve it, this pattern can be called label. Out of 10 toy design problems, 6 received the label as one of the 3 recommended microservices patterns. For these 6 toy design problems, Precision was  $\approx 35\%$ , Recall was 100% and F1-Score was  $\approx 50\%$ . The approach presented a Coverage of 60%, as out of 10 labels, 6 were recommended. In general, the approach presented good results, as 60% of toy

design problems obtained the label as one of the 3 recommended microservices patterns and only 40% did not.

Regarding **specific objective 2**, it was developed the Floc tool that offers a friendly and responsive User Interface (UI), where developers can manage design problems and get microservices pattern recommendations. It was also developed a REST API called Microservices Patterns Recommender (MPR) which, as the name suggests, is responsible for recommending microservices patterns. This REST API uses the proposed recommendation approach to perform recommendations. To recommend microservices patterns, the tool interacts with the MPR.

The tool was evaluated through tests carried out with industrial design problems in a company that offers integrated and innovative cyber security and infrastructure solutions. Three developers from this company ran tests in which they asked for microservices patterns recommendations 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. After testing, the developers involved were individually interviewed

The results from tests with industrial design problems showed that out of 24 recommendations, 14 were evaluated as “Yes, It Does Solve”, 6 as “It Does Partially Solve” and only 4 as “It Does Not Solve”. This means that more than 80% of the recommendations contributed to solving the problems indicated in the tests. Although 4 recommendations were rated as “It Does Not Solve”, for each problem there was at least 1 useful recommendation. Out of 10 microservices patterns, only 2 were not recommended, this shows that the recommended patterns vary and that the recommendations are not biased.

Regarding the interviews carried out with the developers involved in the tests, 9 questions were asked. In general, the responses obtained indicated that the tool is useful in the context of systems based on microservices, and can help in the development of new features and the maintenance of existing features; can help less experienced developers; improves the development process; can help developers with continuous learning; increases productivity; helps standardize solutions; helps with quality assurance; serves as support for decision making; it is easy to use; and finally, it could improve by supporting more patterns, presenting practical examples and preferences from other developers.

Thus, when developing or maintaining a system based on microservices architecture, developers and organizations can benefit from microservices patterns recommendations that help solve design problems. The tests carried out and the interviews carried out corroborated the effectiveness of the tool and the approach, as the tool uses the approach to make recommendations.

## 5.2 Contributions

This research generated two articles, the “Recommendation of Microservices Patterns Through Automatic Information Retrieval Using Problems Specified in Natural Language” paper (P1) (MOURA et al., 2022) and the “Microservices Patterns Recommendation Based on Information Retrieval” paper (P2).

P1 was presented and published at the 22nd International Conference on Computational Science and Its Applications (ICCSA 2022) (ICCSA, 2022). P2 was submitted this year to a journal. P1 is about specific objective 1 and P2 is about specific objective 2.

Thus, this work contributed with: 1) a recommendation approach based on IR that is capable of receiving a design problem and recommending 3 microservices patterns that can solve it; and 2) the Floc tool that offers a friendly and responsive UI, where developers can manage design problems and get microservices patterns recommendations.

## 5.3 Future Works

Regarding future works, it is interesting to consider the limitations, improvements and difficulties that the interviewed developers pointed out:

- **Limitations:**
  - CMP no longer has microservices patterns;
  - Do not show practical examples;
  - Do not display other developers’ preferences for solving certain types of problems.
- **Improvements:**
  - Include more microservices patterns in the CMP;
  - Include statistics that indicate other developers’ preferences for solving certain types of problems;
  - Include suggestions on how to organize folders and files.
- **Difficulties:**
  - Understand the information presented.

It is interesting to include more microservices patterns in the CMP so that the tool is capable of dealing with other types of problems. It is important to indicate other developers’ preferences for solving certain types of problems, as this can serve to indicate the best solutions. It is relevant to include practical examples, involving folders and files. It is also interesting to include explanations so that the information presented is understood without difficulties.

In addition to considering limitations, improvements and difficulties, it is also interesting to think about carrying out tests on larger scales to develop a dataset of design problems, using the feedback feature described for the tool. This dataset can be used to make recommendations through machine learning, making possible a comparison with the IR-based recommendation approach.

It is possible to combine the proposed approach with a questionnaire to better understand the design problem reported by developers and reduce the number of microservices patterns that must be compared with the design problem, this can result in more assertive recommendations. It is also possible to improve the proposed approach by using a lexical database, such as WordNet ([PRINCETONUNIVERSITY, 2023](#)), to consider synonyms.

Finally, it is worth thinking about using the Elasticsearch<sup>1</sup> tool, as well as make comparisons with solutions such as the popular ChatGPT.

---

<sup>1</sup> Elasticsearch is a distributed RESTful search and data analysis engine capable of serving a growing number of use cases ([ELASTICSEARCH, 2023](#)).

# Bibliography

AMPATZOGLU, A. et al. Identifying, Categorizing and Mitigating Threats to Validity in Software Engineering Secondary Studies. *Information and Software Technology*, Elsevier, v. 106, p. 201–230, 2019. Citado na página 72.

APACHE. Home. [S.l.], 2023. Disponível em: <<https://tomcat.apache.org>>. Acesso em: September 18, 2023. Citado na página 53.

BROOKS, F.; KUGLER, H. No Silver Bullet. *H.-J. Kugler, ed., Elsevier Science Publishers B.V., Information Processing '86*, North-Holland, 1987. Citado na página 21.

CELIKKAN, U.; BOZOKLAR, D. A Consolidated Approach for Design Pattern Recommendation. *2019 4th International Conference on Computer Science and Engineering (UBMK)*, p. 1–6, 2019. Citado na página 18.

CERI, S. et al. The information retrieval process. *Web Information Retrieval*, Springer, p. 13–26, 2013. Citado na página 22.

CHEN, R.; LI, S.; LI, Z. From Monolith to Microservices: A Dataflow-Driven Approach. *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, p. 466–475, 2017. Citado 4 vezes nas páginas 14, 15, 20, and 21.

DEACON, J. Model-View-Controller (MVC) Architecture. *JOHN DEACON Computer Systems Development, Consulting & Training*, v. 28, 2009. Citado na página 49.

DOCKER. Home. [S.l.], 2023. Disponível em: <<https://www.docker.com>>. Acesso em: September 18, 2023. Citado na página 54.

DOUGLASS, B. P. Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. [S.l.]: *Addison-Wesley Professional*, 2003. Citado na página 17.

DRAGONI, N. et al. Microservices: Yesterday, Today, and Tomorrow. *Present and Ulterior Software Engineering*, Springer, p. 195–216, 2017. Citado 2 vezes nas páginas 15 and 20.

ELASTICSEARCH. Home. [S.l.], 2023. Disponível em: <<https://www.elastic.co>>. Acesso em: October 23, 2023. Citado na página 76.

ENCODE. Home. [S.l.], 2023. Disponível em: <<https://www.uvicorn.org>>. Acesso em: September 18, 2023. Citado na página 55.

FRITZSCH, J. et al. From Monolith to Microservices: A Classification of Refactoring Approaches. In: SPRINGER. *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*. [S.l.], 2019. p. 128–141. Citado na página 21.

GAMMA, E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. 1. ed. [S.l.]: *Addison-Wesley Professional*, 1995. Citado 3 vezes nas páginas 14, 17, and 21.



- HAMBARDE, K. A.; PROENCA, H. Information Retrieval: Recent Advances and Beyond. *IEEE Access*, v. 11, p. 76581–76604, 2023. Citado na página 22.
- HAMDY, A.; ELSAYED, M. Topic Modelling for Automatic Selection of Software Design Patterns. *Proceedings of the International Conference on Geoinformatics and Data Analysis*, p. 41–46, 2018. Citado 2 vezes nas páginas 15 and 17.
- HEVNER, A. et al. Design Science Research in Information Systems. *Design Research in Information Systems: Theory and Practice*, Springer, p. 9–22, 2010. Citado 3 vezes nas páginas 16, 17, and 73.
- HEVNER, A. R. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, v. 19, n. 2, p. 4, 2007. Citado na página 16.
- HUSSAIN, S. et al. A Methodology to Rank the Design Patterns on the Base of Text Relevancy. *Soft Computing*, Springer, v. 23, p. 13433–13448, 2019. Citado 3 vezes nas páginas 14, 17, and 21.
- IBRIHICH, S. et al. A Review on Recent Research in Information Retrieval. *Procedia Computer Science*, Elsevier, v. 201, p. 777–782, 2022. Citado 3 vezes nas páginas 22, 23, and 24.
- ICCSA. Home. [S.l.], 2022. Disponível em: <<https://2022.iccsa.org>>. Acesso em: October 22, 2023. Citado na página 75.
- KOSCHEL, A.; ASTROVA, I.; DÖTTERL, J. Making the Move to Microservice Architecture. *2017 International Conference on Information Society (i-Society)*, p. 74–79, 2017. Citado na página 20.
- LANDAY, J. A.; HONG, J. I. et al. The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. [S.l.]: *Addison-Wesley Professional*, 2003. Citado na página 17.
- MARTIN, R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. [S.l.]: *Prentice Hall*, 2017. Citado na página 50.
- MOURA, Á. et al. Recommendation of Microservices Patterns Through Automatic Information Retrieval Using Problems Specified in Natural Language. *International Conference on Computational Science and Its Applications*, p. 489–501, 2022. Citado na página 75.
- MSSP. MSSP Alert Top 250 2022 Page. [S.l.], 2023. Disponível em: <<https://www.msspalert.com/top-250-2022>>. Acesso em: September 12, 2023. Citado na página 68.
- PAPAZOGLU, M. P.; HEUVEL, W.-J. V. D. Service Oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB Journal*, Springer, v. 16, p. 389–415, 2007. Citado na página 14.
- PRINCETONUNIVERSITY. Home. [S.l.], 2023. Disponível em: <<https://wordnet.princeton.edu>>. Acesso em: December 11, 2023. Citado na página 76.
- RAHMATI, R.; RASOOLZADEGAN, A.; DEHKORDY, D. T. An Automated Method for Selecting GoF Design Patterns. *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*, p. 345–350, 2019. Citado na página 17.

RICHARDSON, C. *Microservices Patterns: With Examples in Java*. [S.l.]: *Simon and Schuster*, 2018. Citado 2 vezes nas páginas 15 and 28.

RICHARDSON, C. What are Microservices? [S.l.], 2023. Disponível em: <<https://microservices.io>>. Acesso em: June 26, 2023. Citado 5 vezes nas páginas 21, 28, 30, 49, and 66.

ROSHDI, A.; ROOHPARVAR, A. Information Retrieval Techniques and Applications. *International Journal of Computer Networks and Communications Security*, v. 3, n. 9, p. 373–377, 2015. Citado na página 22.

SAINI, B.; SINGH, V.; KUMAR, S. Information Retrieval Models and Searching Methodologies: Survey. *Information Retrieval*, v. 1, n. 2, 2014. Citado na página 25.

SANYAWONG, N.; NANTAJEEWARAWAT, E. Design Pattern Recommendation Based-on a Pattern Usage Hierarchy. *2014 International Computer Science and Engineering Conference (ICSEC)*, p. 134–139, 2014. Citado na página 18.

SANYAWONG, N.; NANTAJEEWARAWAT, E. Design Pattern Recommendation: A Text Classification Approach. *Proceedings of the 6th International Conference on Information and Communication Technology for Embedded Systems*, 2015. Citado 4 vezes nas páginas 14, 15, 17, and 21.

SCHUMACHER, M. et al. *Security Patterns: Integrating Security and Systems Engineering*. [S.l.]: *John Wiley & Sons*, 2013. Citado na página 17.

SILVA-RODRÍGUEZ, V. et al. Classifying Design-Level Requirements Using Machine Learning for a Recommender of Interaction Design Patterns. *IET Software*, Wiley Online Library, v. 14, n. 5, p. 544–552, 2020. Citado na página 18.

SINGLETON, A. The Economics of Microservices. *IEEE Cloud Computing*, IEEE, v. 3, n. 5, p. 16–20, 2016. Citado na página 20.

SOARES, M. S.; FRANCA, J. M. Characterization of the Application of Service-Oriented Design Principles in Practice: A Systematic Literature Review. *Journal of Software*, Academy Publisher, v. 11, n. 4, p. 403–418, 2016. Citado na página 14.

SOUSA, L. et al. Identifying Design Problems in the Source Code: A Grounded Theory. *Proceedings of the 40th International Conference on Software Engineering*, p. 921–931, 2018. Citado 3 vezes nas páginas 15, 16, and 21.

SOUSA, L. et al. How do Software Developers Identify Design Problems? A Qualitative Analysis. *Proceedings of the XXXI Brazilian Symposium on Software Engineering*, p. 54–63, 2017. Citado 3 vezes nas páginas 14, 15, and 21.

TAIBI, D.; LENARDUZZI, V.; PAHL, C. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. *IEEE Cloud Computing*, IEEE, v. 4, n. 5, p. 22–32, 2017. Citado 2 vezes nas páginas 20 and 21.

TAIBI, D.; LENARDUZZI, V.; PAHL, C. Architectural Patterns for Microservices: A Systematic Mapping Study. In: SCITEPRESS. *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*. [S.l.], 2018. Citado na página 20.

THÖNES, J. Microservices. *IEEE Software*, IEEE, v. 32, n. 1, p. 116–116, 2015. Citado 2 vezes nas páginas 15 and 20.

THYMELEAF. Home. [S.l.], 2023. Disponível em: <<https://www.thymeleaf.org>>. Acesso em: September 12, 2023. Citado na página 45.

TIANGOLO. Home. [S.l.], 2023. Disponível em: <<https://fastapi.tiangolo.com>>. Acesso em: September 18, 2023. Citado na página 46.

UYSAL, A. K. An Improved Global Feature Selection Scheme for Text Classification. *Expert Systems with Applications*, Elsevier, v. 43, p. 82–92, 2016. Citado 2 vezes nas páginas 18 and 31.

VMWARETANZU. Home. [S.l.], 2023. Disponível em: <<https://spring.io>>. Acesso em: August 29, 2023. Citado na página 45.

ZHOU, X. et al. A Map of Threats to Validity of Systematic Literature Reviews in Software Engineering. *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, p. 153–160, 2016. Citado na página 72.