INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Implementação do Algoritmo Apriori num Sistema de Recomendação de Grupo baseado em Agentes

RUI FAUSTO MARTINS ALVES
Setembro de 2024

POLITÉCNICO
DO PORTO

ISEP

Apriori Algorithm Implementation in an Agent based Group Recommendation System

Rui Fausto Martins Alves

Dissertation submitted as a partial fulfillment for the

Master's Degree in Computer Science

Area of Specialization in Software Engineering

Supervisor (ISEP): Goreti Marreiros

Co-supervisor (ISEP): Patrícia Alves

September 2024

To my grandmother Manuela

Acknowledgments

I would like to deeply thank everyone involved in my academic and professional journey, particularly those who directly contributed to my success over the adversities and obstacles that appeared.

A special and unique thank you to my grandmother Manuela for all the love she gave me for almost 24 years. It is with deep sadness and desolation that I will not be able to thank you in person, however, the affection you gave me will remain in my thoughts with each passing day. Thank you, grandmother, for having such pure love, for the constant smile when you greeted me, for all the games, conversations, and all the laughs you brought out of me.

A thank you is not enough for my family, who are part of my daily life and have had the biggest impact on my journey as a human being. To my mother for her effort, affection and dedication to get me this far, to Pedro for being a plus since he came into our lives, and to my sister Leonor for taking me out of my mind, but also for making me laugh with her way of being.

Thank you to my grandparents, Fausto and Fátima, for all the love, to my aunts and uncles who are present, and to my cousins who I interact with and adore.

To my uncle Joaquim, for being such a great person that reentered my life. You and aunt Laura played such an important role in my childhood, filling it with love, laughter, and unforgettable moments.

Thank you to my friends for all the moments of fun and enjoyment, and to my girlfriend Alice for the love and for putting up with me all these years.

Finally, I cannot forget to thank GECAD for the opportunity and the invaluable experience it provided. I am also deeply grateful for the cooperation of the GECAD members and colleagues. To my supervisor, Goreti Marreiros, and my co-supervisor, Patrícia Alves, I extend my sincere thanks for their constant support and availability throughout this project. Without their guidance, it would not have been possible to achieve the desired objectives.

Declaration of Integrity

I declare that I have conducted this academic work with integrity.

I have not plagiarized or applied any form of misuse of information or falsification of results throughout the process that led to its elaboration.

Therefore, the work presented in this document is original and my own, and does not previously been used for no other purpose.

I further declare that I am fully aware of the P.PORTO Code of Ethical Conduct.

ISEP, Porto, September 15, 2024

Resumo

Os sistemas de recomendação (RS) são utilizados em vários setores para prever, filtrar ou dar recomendações aos utilizadores com base em comportamentos, padrões ou preferências anteriores. Porém, quando é necessário fazer essas previsões para grupos, em que vários utilizadores com preferências diferentes estão envolvidos, os RS tradicionais têm mais dificuldade em fornecer recomendações que atendam às suas necessidades. Os sistemas de recomendação para grupos (GRS) procuram resolver este problema ao fornecer recomendações personalizadas, geralmente com base nas preferências agregadas de cada membro do grupo. A área do turismo é um exemplo de uma área em que este tipo de recomendações é complexa. Com o objetivo de melhorar as sugestões dadas aos grupos turísticos, este projeto sugeriu a integração de regras de associação, através do algoritmo Apriori, no atual Microserviço Multiagente (MAMS) de um protótipo de GRS de turismo, Grouplanner.

O algoritmo Apriori é uma importante ferramenta de mineração de regras de associação para identificar padrões e associações entre objetos num conjunto de dados e pode ser usado para descobrir associações entre locais de interesse (POI) frequentemente visitados, examinando as preferências e experiências de viagem dos membros do grupo.

Os principais objetivos do projeto foram melhorar a seleção de POI, incentivar uma abordagem mais personalizada às recomendações e modificar dinamicamente as recomendações de acordo com as preferências individuais dos membros dos grupos, através de previsões resultantes do algoritmo Apriori, integrando-o num dos microserviços desenvolvidos em .NET e alojados no Azure, o MAMS, existente no protótipo de GRS.

Em particular, foi desenvolvido um algoritmo Apriori personalizado, em C#, assim como a geração das regras de associação em conformidade com o sistema multiagente. Seguidamente, foram conduzidos testes de modo a demonstrar a eficácia do Apriori na geração de regras de associação e estabelecer padrões de preferências de POI entre grupos turísticos.

Os resultados dos testes mostram que o algoritmo Apriori é capaz de incentivar uma abordagem mais personalizada de POI para incluir ou excluir da recomendação, através dos padrões que encontra entre as preferências e características dos turistas, melhorando assim as sugestões personalizadas, conscientes do contexto em que se enquadram, apesar de conferir limitações no que toca à estabilidade do algoritmo em grandes conjuntos de dados. Assim, estas descobertas mostram um desempenho confiável na criação de listas de pontos de interesse (POI) para incluir ou excluir das recomendações, e definem o caminho para futuras melhorias nas tecnologias de sistemas de recomendação.

Palavras-chave: Algoritmo Apriori, Regras de Associação, Sistemas de Recomendação para Grupos, Sistemas Multiagente, Microserviços Multiagente.

Abstract

Recommendation systems (RS) are used in various sectors to predict, filter, or provide recommendations to users based on previous behaviors, patterns, or preferences. However, when it is necessary to make these predictions for groups, where multiple users with different preferences are involved, traditional RS face more difficulty in providing recommendations that meet their needs. Group recommendation systems (GRS) aim to solve this problem by offering personalized recommendations, usually based on the aggregated preferences of each group member. The tourism sector is an example of an area where this type of recommendation is complex. To improve suggestions given to tourist groups, this project proposed integrating association rules, through the Apriori algorithm, into the current Multi-Agent Microservice (MAMS) of a tourism GRS prototype, Grouplanner.

The Apriori algorithm is an important tool for mining association rules to identify patterns and associations between objects in a dataset. It can be used to discover associations between frequently visited points of interest (POI) by examining the travel preferences and experiences of group members.

The main goals of the project were to improve POI selection, encourage a more personalized approach to recommendations, and dynamically modify recommendations according to individual group members' preferences. This was achieved through predictions resulting from the Apriori algorithm, integrating it into one of the microservices developed in .NET and hosted on Azure, MAMS, which is part of the GRS prototype.

A customized Apriori algorithm was developed in C#, as well as the generation of association rules in line with the multi-agent system. Subsequently, tests were conducted to demonstrate the effectiveness of Apriori in generating association rules and establishing POI preference patterns among tourist groups.

The results show that the Apriori algorithm can encourage a more personalized approach to which POI to include or exclude from recommendations, based on the patterns it identifies among tourists' preferences and characteristics. This improves personalized suggestions, aware of the context in which they are framed, although the algorithm presents limitations regarding stability with large datasets. These findings show reliable performance in creating lists of points of interest (POI) to include or exclude from recommendations and pave the way for future improvements in recommendation system technologies.

Keywords: Apriori Algorithm, Association Rules, Group Recommendation Systems, Multi-Agent Systems, Multi-Agent Microservices

Table of Contents

Figures List

Tables List

Code Snippets List

Abbreviations

API	Application Programming Interface
CATS	Collaborative Advisory Travel System
DTO	Data Transfer Objects
Eclat	Equivalence Class Transformation
FP	Frequent Pattern
FURPS	Functionality Usability Reliability Performance Supportability
GAMA	Generic Agent-based Data Mining
GQM	Goal Question Metric
GRS	Group Recommender Systems
GRUD	Create Read Update Delete
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
ID	Identity
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
MAMS	Multi-Agent Microservice
MAS	Multi-Agent Systems
MCDA	Multi-Criteria Decision Analysis Architecture
MDPI	Multidisciplinary Digital Publishing Institute
OLAP	Online Analytical Process
POI	Points Of Interest
POIMS	Points Of Interest Microservice
QFD	Quality Function Deployment
RE	Recommendation Engine
REMS	Recommendation Engine Microservice
RS	Recommender Systems

SOA	Service-Oriented Architecture
UI	User Interface

1. Introduction

In this chapter, the foundation for the dissertation is laid. It begins with the context of the research, followed by a description of the problem that the study aims to solve. The objectives of the work are then described, focusing on the primary aims to be achieved. The chapter finishes with an overview of the document's structure, which serves as a road map for the issues discussed in the next sections.

1.1. Context

Within the scope of the curricular unit “Dissertation (DIMEI)”, master’s in Computer Engineering, specialization in Software Engineering, from the Instituto Superior de Engenharia do Porto (ISEP), this research project aimed to improve recommendations of points of interest (POI) in a prototype of a Group Recommender System (GRS) for Tourism, Grouplanner, that is currently under development at the Research Group on Engineering and Intelligent Computing for Innovation and Development (GECAD) (GECAD, 2024b), also at ISEP. This work is part of the initiative proposed by GECAD and fully developed in GECAD facilities, whose mission is to develop scientific research and innovation for the incorporation of Intelligence in Complex Engineering and Computing Systems.

Grouplanner (GECAD, 2024a) is a Group Recommender System (GRS) developed to help Chinese tourist groups plan tours and choose Points of Interest (POI) in Northern Portugal. It allows groups to select from a list of options depending on individual preferences, improving security and happiness throughout their stay. Its goal was to perform investigations, research, and experiments under the topic of GRS. Primarily, the system ought to assist groups in choosing one (hotel, city, etc.) or several (points of interest, POI, etc.) from a range of options (based on the interests and preferences of each group member). Next, it ought to provide each group member with assistance during their stay, thereby fostering a stronger sense of security. Consequently, the system aims to optimize group satisfaction across multiple aspects, such as point of interest recommendations.

Grouplanner is an GRS mobile application based on a microservices architecture developed in .NET, C#, consisting of five microservices, hosted in Azure, one of which is the Multi-Agent Microservice (MAMS). In MAMS, multiple autonomous agents communicate and collaborate based on their unique profiles to achieve complicated tasks, such as dynamically grouping individuals with similar qualities or interests (Alves et al., 2024).

Grouplanner gives continuity to ATT - Accelerate and Transform Tourism – initiative (ATT, 2021), as it fits in with its aim of taking advantage of innovative technologies to improve the tourist business with processes, backed by technology and human resource qualification, to help to revolutionize the business landscape and tourist management organizations, increasing their ability to respond to changing market requirements and improving operational efficiency.

The prototype brings new programming concepts that complement knowledge acquired in the master's degree related to software engineering, but also allows exploring modern technologies and research approaches, namely the implementation of a data mining algorithm in a multi-agent microservice (MAMS).

1.2. Problem Description

Recommendation systems (RS) are becoming increasingly significant tools for assisting users in making decisions across a range of domains (Alves, Martins, Saraiva, et al., 2023). However, when there are multiple participants, preference conflicts and heterogeneity take place, and group recommendation systems (GRS) are being proposed as a solution to these problems.

Serving a group of users with varying preferences and characteristics presents a special challenge for GRS (Wooldridge, 2002). Because of this preference heterogeneity, which often results in conflicts, traditional RS finds it challenging to offer consistently satisfying recommendations for every member of the group.

By offering contextual and personalized recommendations that take into consideration the preferences of each group member, GRS are intended to address these issues (Alves et al., 2019). Traveling as a group can be made much more enjoyable and satisfying by GRS, as it takes into account the varied interests and preferences of its members.

There are still a number of limitations with GRS, including the lengthy setup and excessive intrusiveness needed to generate recommendations and create user profiles (Lillis, 2020). To overcome these restrictions, researchers have suggested using multi-agent systems (MAS), which proactively make recommendations based on user profiles and context.

1.3. Objectives and Research Questions

The goal of this project is to enhance suggestions made for Points of Interest (POI) in an under-development prototype GRS for Tourism. A data mining algorithm will be incorporated into the current multi-Agent Microservice in order to accomplish this goal. To improve the selection of recommended points of interest, the algorithm considers individual or each group's profile and prior travel experiences and offer tailored lists of POI to include or exclude to the recommendation, either to a group or individual preference. Therefore, to respond to this goal, the objectives were proposed as:

1. To study and review association rules data mining algorithms to find preference patterns between tourists, namely Apriori and its variants and the operational mechanism, considering the computational efficiency and performance between variants with different datasets, and identify their challenges and limitations, to find the more suitable algorithm to implement.
2. To study Recommendation Systems, namely Recommendation Systems for Groups and the implications of association rule mining algorithms in RS and GRS. This includes understanding the fundamentals of RS and applications with data mining algorithms across industries and evaluate the challenges around it.
3. To study and delve into Multi-Agent Systems and their features, analyzing the core differences between MAS and regular systems, including their architecture and coordination mechanisms, and how can association rules data mining algorithms fit with MAS.
4. To study and develop the suitable association rules data mining algorithm, triggered automatically or manually, for the GRS prototype in the MAS microservice (MAMS). This means to identify the requirements and constraints of the prototype withing the

MAMS framework, selecting and adapting the chosen algorithm and evaluate, with testing, the performance in terms of accuracy, efficiency, and scalability within the MAS environment.

Based on the proposed objectives and existing research, the following research questions take place:

- **RQ1** - How accurate is the data mining algorithm when integrated into a multi-agent system at generating rules between tourists with different preferences and traits?
- **RQ2** - What impact does the use of a data mining algorithm generating rules have on the personalization of POI lists to include or exclude for group members with different preferences and/or traits?
- **RQ3** - To what extent does variable thresholds of the data mining algorithm, if any, influence the accuracy of association rules in predicting group behavior?
- **RQ4** - What are the limitations of the data mining algorithm when applied to larger datasets in a multi-agent tourism recommendation system?

1.4. Report Structure

The report is divided into several chapters. The first chapter, introduction, focuses on defining the project's context, outlining the stated problem, objectives, and the Apriori algorithm's critical role in improving point-of-interest recommendations inside a Group Recommendation System for Tourism.

The third chapter describes the research methodology, including how the Apriori algorithm, Group Recommendation Systems, and Multi-Agent Systems were ethically and methodical researched. Also, it’s presented the Context and Related Work, with:

- Apriori algorithm introduction, along with information on how it works and its variations.
- Recommendation Systems and the use of the Apriori algorithm in RS, its limitations and applications across industries.
- Group Recommendation Systems, putting a focus on group preferences.

- Multi-Agent Systems, empathizing Agent and MAS features, and the implementation of the Apriori algorithm within MAS.
- Multi-Agent Microservices, with an emphasis on Microservices integrated into MAMS.
- Related works across different industries.

The following chapter describes the solution analysis on the current prototype and the design, documenting all artifacts for the implementation solution and offers the comprehension of the business rules around the prototype, as well the requirements to meet.

Next chapter focuses on the implementation of the requirements described in the previous chapter, englobing all details for the clustering of users, generate Association Rules and provide POI Inclusion/Exclusion lists, followed by the respective implementation tests.

The subsequent chapter focuses on assessing the implementation through the Goals, Questions, Metrics approach, consequently defining hypothesis tests to evaluate the solution and measure if the functionalities met the expected criteria.

Finally, the last chapter draws conclusions from the study based on the degree of achievement of the objectives described, and comments on possible limitations and threats to the solution implementation proposal and a final overall statement about the dissertation project.

This document and developed work also adheres to established ethical norms of the P.Porto Code of Good Practices and Conduct (E da Fonseca Margato, 2020) and the Code of ethics Portuguese Board of Engineers (Ordem dos Engenheiros, 2016).

1.5. Work Planning

A Gantt chart is a project management tool used to plan and schedule projects of various kinds (Geraldi & Lechter, 2012). It is especially useful for project visualization. A Gantt chart is a graphical representation of activities against time that aids project managers in monitoring progress.

Table 1 displays a Gantt chart with the sequential and parallel evolution of assignments, offering an in-depth understanding of the project's roadmap.

Table 1 - Project Timeline and Milestones

	2023-2024										
Task Description	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Problem Identification	■	■									
Objectives		■									
Research Methodology		■									
Context and Related Work		■	■								
Codebase Analysis Pre-Implementation				■	■						
Functionality Implementation					■	■	■				
Integration and Functional Testing						■	■	■	■		
Results Conclusion										■	■
Writing Report	■	■	■	■	■	■	■	■	■	■	■

In November and December, the focus was on identifying the main problem regarding the lack of customization on recommending POI in the GRS prototype. Also in December, clear objectives, reported in section 1.3, were set to define what the project aims to achieve.

Between December and January, the research methods were developed to guide how the problem would be approached. Next in the timeline, the context and related work solutions was written.

During February and March, the codebase was analyzed before any implementation begins. From March to April, the core functionalities of the system were built.

Between April and June, the different parts of the system were integrated and then tested to ensure everything meets the acceptance criteria.

In June and July, conclusions were drawn from the results of the system. Finally, during all the other assignments, a detailed report was written to document the entire process and findings.

2. Context and Related Work

Research methodology is the theoretical examination of procedures used in a field of study. It consists of a theoretical examination of the body of methods and principles linked with a particular field of knowledge (Williams, 2007). This way of research was intended to build a foundation for reliable and trustworthy information to guide the main topics of the study.

2.1. Literature Review

Literature reviews are carried out to identify gaps in the available literature, opportunities for more research, and to provide a summary of the current state of knowledge on a specific issue (Kraus et al., 2022). This can also be defined as research that analyzes and synthesizes an existing body of literature by identifying, criticizing, and developing the theoretical building blocks through an assessment of a body (or bodies) of preceding work. It serves as the intellectual framework for future study. It enables researchers to connect with current literature, participate in ongoing discussions, and define the direction of their own study within a larger academic framework (Paul & Criado, 2020).

2.1.1. Inclusion and Exclusion Criteria

Researchers employ inclusion and exclusion criteria to choose which studies to include or exclude from a systematic review or meta-analysis (Meline, 2006). Inclusion criteria are factors that a study must have in order to be included in the review, whereas exclusion criteria are factors that would prohibit a study from being included. These inclusion and exclusion criteria, when combined, form a structured approach to the literature review process, allowing the researcher to filter through an abundance of sources and select the most relevant ones that contribute notably to the project.

Table 2 illustrates the inclusion criteria to include the literature study, and the exclusion criteria to exclude from the literature study.

Table 2 - Criteria Framework for Literature Selection

Criteria	Content
Inclusion	Articles discussing the implementation the Apriori algorithm
	Research of Recommendation Systems (RS)
	Practical cases of implementing the Apriori algorithm in GRS
	Implementation of the Apriori algorithm in GRS
	Materials discussing MAS and their role in recommendation systems
	Content exploring the integration of MAMS
Exclusion	Commercial publications or content primarily intended for promotional purposes
	Articles not written in English
	Publications unrelated to Software Engineering, Association Rules Data Mining Algorithms, GRS, MAS, MAMS, or RS
	Articles lacking clear evidence of the author's perspective or research contribution

In order to gain an extensive understanding of the issue, the literature was not limited to studies published within the recent five years. Some foundational investigations, which may go back further, give insights and provide the groundwork for future study. By integrating earlier works, listed in Table 3, the review acquires a broader perspective, allowing for the investigation of the evolution and application of algorithms such as Apriori in GRS, MAS, and MAMS.

2.1.2. Information Sources

Google Scholar, IEEE Xplore, and the Clarivate InCites Journal Citation Reports are all used in the research. These platforms provide access to a wide range of scholarly articles, technical reports, and books, all of which contribute to an accurate and well-informed literature assessment.

2.1.3. Search Terms

The following search terms were used to acquire relevant literature:

- Apriori Algorithm
- Apriori Algorithm Variants
- Recommendation Systems
- Group Recommendation Systems
- Multi-Agent Microservices
- Multi-Agent Systems
- Apriori Algorithm in Multi-Agent Systems

2.1.4. Selection Process

A thorough methodology was used during the systematic review and selection process to ensure the inclusion of the most relevant studies. The selection procedure is outlined in the following steps:

- **Initial Gathering:** The project's scope was initially covered in an in-depth collection of papers, blog entries, and magazine pieces, gathering a number of 11439 results.
- **Exclusion based on Criteria:** A review was conducted to eliminate content that did not meet the inclusion requirements.
- **Final Selection:** After the previous steps, a final list of 52 results was presented in accordance with the criteria.
- **Final Utilization:** Of the final list, the used papers or articles that were chosen as being basis for the state of art were 34.

Therefore, the final list that contributed to the state of art is presented in Table 3.

Table 3 - Search Outcome

Title	Author(s)	Year
A Novel Recommender System Based on Apriori Algorithm for Requirements Engineering	S. Alzu'Bi, B. Hawashin, M. Eibes, and M. Al-Ayyoub	2018
A State-of-the-Art Survey on Various Domains of Multi-Agent Systems and Machine Learning	A. H. Barrientos and A. N. Luevano	2022
A Survey of Recommender Systems: Approaches and Limitations	M. Sharma and S. Mann	2013
A Systematic Mapping Study in Microservice Architecture	N. Alshuqayran, N. Ali, and R. Evans	2016
An Improved Apriori Algorithm For Association Rules	M. Al-Maolegi and B. Arkok	2014
An improved Apriori algorithm for mining association rules	X. Yuan	2017
An improved Apriori-based algorithm for association rules mining	H. Wu, Z. Lu, L. Pan, R. Xu, and W. Jiang	2009
An Introduction to MultiAgent Systems	M. Wooldridge	2002
Application of Data Mining in e-Commerce	M. Chajri and M. Fakir	2016
Apriori Algorithm – Frequent Pattern Algorithms	S. Seela	2023
Apriori Algorithm for the Data Mining of Global Cyberspace Security Issues for Human Participatory Based on Association Rules	Z. Li, X. Li, R. Tang, and L. Zhang	2021
Apriori Algorithm in Data Mining: Implementation With Examples	Sruthy	2024
Association rules algorithms for data mining process based on multi agent system	I. Belabed, M. T. Alaoui, J. El Miloud, and A. Belabed	2019
Association Rules Mining: A Recent Overview	S. Kotsiantis and D. Kanellopoulos	2006
Automatic Rule Generation for Decision-Making in Context-Aware Systems Using Machine Learning	R. Jabla, M. Khemaja, F. Buendia, and S. Faiz	2022

Comparative Study of Apriori-variant Algorithms	S. Mutalib, A. Azri, A. Subar, S. Abdul-Rahman, and A. Mohamed	2014
Comparative Study on Apriori Algorithm and Fp Growth Algorithm with Pros and Cons	M. M. Kavitha and S. T. Tamil Selvi	2016
Conference on Local Computer Networks	A. Paricio García, J. Oliver, and D. Gosch	2010
Conformal Group Recommender System	V. R. Kagita, A. Singh, V. Kumar, P. K. R. Neerudu, A. K. Pujari, and R. K. Bondugula	2023
Delivering Multi-Agent MicroServices using CArtAgO	D. Lillis	2020
Evaluation of Apriori, FP growth and Eclat association rule mining algorithms	V. Srinadh	2022
Extensible Multi Agent System for Heterogeneous Database: Knowledge Discovery in Database, Multi Agent Systems, Extendible Multi Agent Data mining System	A. Ramar and M. Elamparithi	2016
Fast Algorithms for Mining Association Rules	R. Agrawal and R. Srikant	1994
FP Growth Algorithm Implementation	S. Sidhu, U. K. Meena, A. Nawani, H. Gupta, and N. Thakur	2014
Grid implementation of the Apriori algorithm	C. Aflori and M. Craus	2007
Grouplanner: A Group Recommender System for Tourism with Multi-agent MicroServices	Alves P, Gomes D, Rodrigues C, Carneiro J, Novais P, Marreiros G	2022
Microservices: How To Make Your Application Scale	N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina	2017
Microservices: The evolution and extinction of web services?	L. Baresi and M. Garriga	2019
Modeling a Mobile Group Recommender System for Tourism with Intelligent Agents and Gamification	P. Alves, J Carneiro, G Marreiros, and P Novais	2019
Multi-agent Systems for Distributed Data Mining Techniques: An Overview	M. H. Qasem, A. Hudaib, N. Obeid, M. A. Almaiah, O. Almomani, and A. Al-Khasawneh	2022

Multi-Agent Systems: A Survey	A. Dorri, S. S. Kanhere, and R. Jurdak	2018
Multi-agent-based modeling for extracting relevant association rules using a multi-criteria analysis approach	A. Ait-Mlouk, F. Gharnati, and T. Agouti	2016
Online Store Product Recommendation System Uses Apriori Method	C. S. Fatoni, E. Utami, and F. W. Wibowo	2018
Recommendation System using Apriori Algorithm	K. Singh Talwar, A. Oraganti, N. Mahajan, and P. Narsale	2015
Research of an Improved Apriori Algorithm in Data Mining Association Rules	J. Yabing	2016

2.2. Introduction to the Apriori Algorithm

Recognized as one of the classic algorithms in the field of association rule mining, the Apriori algorithm was developed by R. Agrawal and R. Srikan in 1994 (Agrawal & Srikant, 1994). The name of the algorithm was Apriori because it uses prior knowledge of frequent itemset properties.

The Apriori algorithm is based on the idea that all subsets of a set of frequent items are also frequent itemsets, and it functions according to fundamental data mining theory (Li et al., 2021). It has been the foundation for frequent itemset mining in databases since its inception, using an iterative process to find patterns and associations. The base for later developments and improvements in this area has been established by its contributions (Yuan, 2017).

The goal of association rule mining is to find interesting correlations, frequent patterns, associations, or casual structures among groups of items in transaction databases or other data repositories (Kotsiantis & Kanellopoulos, 2006). Usually, the problem can be divided into two smaller issues. One is to find itemsets in the database whose occurrences are above a predetermined threshold. These itemsets are referred to as large or frequent itemsets. The second challenge is to use the constraints of minimal confidence to generate association rules from those huge itemsets.

As information technology advances and the need to extract valuable business information from datasets grows, data mining algorithms, such as Apriori or variants, and related techniques seem to be the answer to achieving the objective of finding hidden patterns in vast amounts of data

stored in databases, data warehouses, OLAP (online analytical process), and other information repositories is the crucial process of data mining (Al-Maolegi & Arkok, 2014).

2.2.1. Operational Mechanism of the Apriori Algorithm

This Apriori algorithm discovers frequent item sets and, subsequently, association rules (Seela, 2023). Apriori algorithm is a sequence of steps to be followed to find the most frequent itemset in the given database. This set of operations requires pre-defined values like the minimum of support and confidence thresholds.

Support is calculated as the proportion of transactions in the database containing a particular itemset (Li et al., 2021). For an itemset X, the support is given by the number of transactions containing the itemset X divided by the total number of transactions in the database.

For a rule from itemset X to Y (X => Y), the confidence is the support of the union of X and Y divided by the support of X (Li et al., 2021). This measures how often items in Y appear in transactions that contain X, indicating the likelihood or confidence of Y occurring in instances where X occurs.

Therefore, an item X is considered not frequent if its probability of occurrence in the dataset is less than the minimum support threshold (Li et al., 2021). This concept extends to combinations of items, by means, if an itemset containing X and another item Y also falls under the minimum threshold, is it considered infrequent.

The Apriori algorithm, illustrated in Figure 1, is employed to discover all frequent itemsets within a dataset.

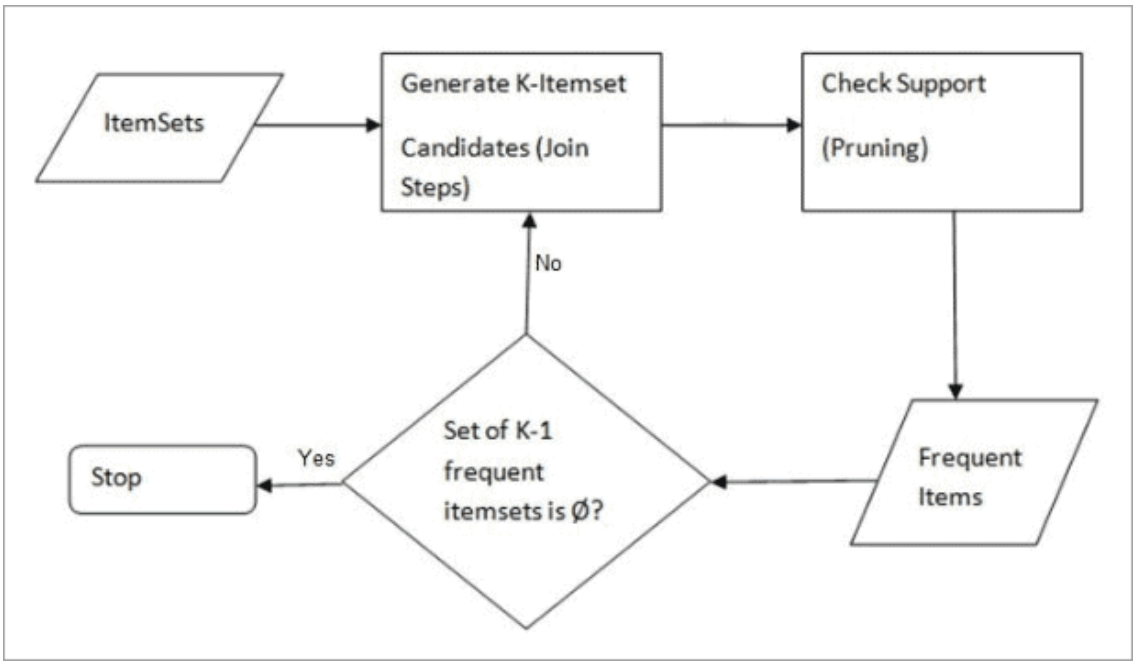

Figure 1 - Apriori Algorithm Flowchart, retrieved from (Sruthy, 2024)

It operates iteratively with the identification of frequent itemsets with the minimum size and counts the occurrences of each item (Yabing, 2016). Only candidates which satisfy the minimum support are taken ahead for the next iteration. From the itemsets with bigger size, the algorithm combines the frequent itemsets with themselves to create bigger candidates. On the other hand, it removes infrequent subsets (pruning), based on the principle that every subset from a frequent itemset also should be frequent. The database then is scanned to count the support of generated candidates. Candidates not meeting the minimum support threshold are discarded.

The process repeats iteratively, increasing set size in each iteration until no more frequent sets are found (Aflori & Craus, 2007). Finally, the algorithm generates association rules from identified frequent itemsets. Confidence measures the rule strength, guiding a selection of significant rules.

Therefore, with the following notations and definitions (Aflori & Craus, 2007; Agrawal & Srikant, 1994; Yabing, 2016), there are the following definitions of the Apriori algorithm:

- I = $\{I_1, I_2, ..., I_m\}$ – set of items.
- D = set of transactions; each transaction *t* is included in I.
- X = set of items from I, *t* contains X.
- An association rule is a pair $X \rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, $X \cap Y = \emptyset$.

- Confidence of the rule X → Y is "C", if C% of the transactions in D that contain the set X, also contain the set Y.
- Support of rule X →Y is "S", if S% of the transactions in D contains the set X ∪ Y.

Given a set of transactions D, generate all the association rules (or a certain number of them) with more support and confidence than the user-specified minimum support and minimum confidence (Aflori & Craus, 2007). Data mining for association rules can be divided in two tasks:

- find all big itemsets with transaction support higher than the minimum support.
- for all large itemsets, for each itemset *l*, find all nonempty subsets of *l* for every such subset *a*, the rule is "$a \rightarrow l \rightarrow a$" if $\frac{support(l)}{support(a)}$ > minimum confidence (Aflori & Craus, 2007; Agrawal & Srikant, 1994).

The Apriori Algorithm in pseudocode is presented in Figure 2.

```
L_l = {frequent 1-itemsets}:
For (k = 2; L_{k-l} <> 0;  k++) {
    C_k= set of new candidates;
    for all transactions t ⊂ D
            for all k-subsets m of t
                    if (m ⊂ C_k) m.count++
    L_k = {n ⊂ C_k | n.count >= minsupp}
}
Set of all frequent itemsets = ∪_k L_k
```

Figure 2 - Apriori Algorithm Pseudocode

A single transaction is characterized as *t*, from the dataset D, while *m* represents a k-subset of transaction *t* and the *n.count* keeps track of how often a candidate itemset *n* appears in the transactions.

Explaining the algorithm, starts by finding all frequent 1-itemsets, by means, sets of individual items that meet the minimum support threshold. The loop iterates to find itemsets of increasing size and stops when are no more frequent itemsets of size *k-1*.

For each transaction *t* in the dataset D, the algorithm checks whether any k-subset of *t* matches a candidate in C_k. If so, the count for that candidate itemset is increased. After counting, L_k represents the set of frequent k-itemsets that are identified as those candidates in C_k that meet the minimum support threshold. The algorithm continues until no more frequent itemsets can be found, after which the set of all frequent itemsets is $\cup_k L_k$.

The following algorithm, displayed in Figure 3, must be used to find the association rules as the Apriori algorithm only finds the frequent itemsets:

```
For (each frequent itemset l) {
    generate all non-empty subsets of l
    for (each non-empty subset a of l) {
    output_rule: a → (l-a) if support(l)/support(a) > = min confidence}
}
```

Figure 3 - Association Rules Pseudocode

Every time the Apriori Algorithm is used, the database is searched to find support for fresh candidates (Aflori & Craus, 2007). It takes a lot of processing power, storage capacity, and input/output (I/O) communications to find frequent itemsets. There is a significant I/O overhead for scanning in each iteration if the database cannot fit in memory.

The process repeats iteratively, increasing set size in each iteration until no more frequent sets are found (Aflori & Craus, 2007). Finally, the algorithm generates association rules from identified frequent itemsets. The confidence previously defined measures the rule strength, guiding a selection of significant rules.

2.2.2. Apriori Algorithm Variants

Spite the Agrawal and Srikan algorithm revolutionized the way large sets of data are interpreted, it suffers from scanning time problem while generating candidates of frequent itemsets (Mutalib et al., 2014). The numerous modifications and enhancements over time increased its effectiveness and solved its drawbacks. They address issues like memory usage, computational complexity, scalability, and flexibility regarding various data formats and data types.

Each variation has a distinct methodology and set of benefits, which makes them essential tools in world of data mining (Wu et al., 2009). Next, the two main variants will be studied to determine if there is one of them that is better suited than the regular Apriori algorithm for the intended application.

Equivalence Class Transformation

Different from the Apriori algorithm, the Equivalence Class Transformation (Eclat) algorithm is intended for mining frequent itemsets using a distinct methodology (Yu & Wang, 2014). It uses a vertical database layout instead of explicitly listing all transactions. This variation arranges data vertically, using a matrix intended for representing transactions as rows and items in the columns. If an item is in a transaction, it's marked as 1, otherwise as 0.

Eclat does not generate candidates itemsets, instead it relies on a tree structure to discover items combinations, starting with the most frequent item and adds related items to form a tree, and the paths represent itemsets (Srinadh, 2022). This algorithm takes a depth first search approach to traverse the tree set and discover frequent itemsets. It requires less space than Apriori if item sets are small. It is suitable for small datasets and requires less time for frequent pattern generation than Apriori.

Eclat, although more efficient than regular Apriori, by using a vertical layout, may consume more memory when dealing with larger datasets, which can be a challenge (Agrawal & Srikant, 1994; Srinadh, 2022). Apriori tends to be more simple, understandable and perform well in a variety of datasets, so it is preferable to Eclat for dealing with tourism datasets due to its versatility.

Frequent Pattern Growth

Comparing to the Apriori algorithm, Frequent Pattern (FP) Growth discovers the frequent itemset without the candidate generation (Sidhu et al., 2014). It mainly consists of a compact data structure called FP-Tree and extracts directly the frequent itemsets.

The FP-Tree is built by traversing the data set twice. FP-Tree is built, first, scanning the data and then finding support for each item, discarding the infrequent itemsets and in decreasing order based on the support. Nodes have a counter and match an itemset. Pointers are kept between nodes that contain the same item, resulting in a linked list. The more paths that overlap, the higher the compression (Kavitha & Tamil Selvi, 2016).

The path nodes are arranged in decreasing order of frequency and brings the advantage of offering overlapping itemsets that share the same prefix path within the tree nodes. This means that the information of the data set is highly compressed, and it is only needed two database scans and no candidate generation. First, it maps each transaction to a path by reading one at a time. Paths can overlap when a transaction shares items because of the fixed order used. In this instance, counters are raised (Kavitha & Tamil Selvi, 2016).

FP Growth is more efficient because of the FP-Tree compact data structure, however, compared to Apriori, it can be memory-intensive for large datasets with a large number of unique items, where Apriori is more versatile and simpler. Also, with a high level of support, the regular Apriori will prune most of the infrequent itemsets, therefore, the no candidate generation is no longer a major advantage (Agrawal & Srikant, 1994; Kavitha & Tamil Selvi, 2016; Sidhu et al., 2014; Srinadh, 2022).

2.3. Apriori in Recommendation Systems (RS)

In the context of Recommendation Systems (RS), the Apriori algorithm, traditionally used in data mining for discovering association rules, is applied to find patterns and associations among big datasets. The algorithm therefore seeks to identify items frequently co-selected or co-viewed by users in the process of analyzing their transaction databases, such as purchase history or viewing records. The basic premise is based on the application of these identified patterns to suggest the items, hence offering the recommendations that fit along the tastes of the given users or rather those behaviors exhibited by others with similar kinds of preferences (Singh Talwar et al., 2015).

The Apriori algorithm is effective in RS since it manages to uncover hidden associated relations and patterns amid items. Such patterns are normally never visible yet of great importance

in the process of designing the appropriate recommendations that depict context plus other pertinent disputes. More reasons for the popularity of this algorithm are contributed by its simplicity to apply and use. The truth is, practically everybody carrying out some development can just apply the Apriori model into different RS systems for better results, at any level of development expertise, therefore increasing the applicability of the technology (Fatoni et al., 2018).

2.3.1. Recommendation Systems Limitations

However, there are challenges that arise from using the Apriori algorithm in RS. One of the main drawbacks will arise when applying the approach to large datasets (Sharma & Mann, 2013). Apart from the requirement that the algorithm needs multi scans of extensive databases, slow performance is a big factor which was a critical issue in the age of big data and real-time recommendation engines. Moreover, the algorithm is not robust to data sparsity scenarios - which often crops up in all systems where there exist few interactions from users towards items. In addition, much of the online environment such as e-commerce and social networking is characterized by user preferences that are highly dynamic and changing fast, which makes the less flexible structure that is facilitated by the Apriori algorithm sub-optimal for these applications.

2.3.2. Apriori in RS Applications Across Industries

Practical implementations of the Apriori algorithm in RS are observable across different sectors (Chajri & Fakir, 2016). In the realm of e-commerce, the algorithm plays a pivotal role in enhancing customer shopping experiences. Beyond the "frequently bought together" recommendations, it is also employed to analyze customer purchasing patterns over time. This longitudinal analysis allows e-commerce platforms to anticipate future customer needs and preferences, thereby enabling more targeted marketing strategies and personalized product suggestions.

The entertainment industry, particularly in streaming services for movies and TV shows, also capitalizes on the strengths of the Apriori algorithm (Alzu'Bi et al., 2018). Platforms analyze

viewers' watching habits, identifying common sequences and combinations of content consumed. This data is instrumental in curating more personalized content libraries for users, increasing viewer engagement and satisfaction. Additionally, these insights assist in content acquisition strategies, guiding platforms towards genres or types of content that are likely to be successful among their user bases.

In music streaming services, the Apriori algorithm is used similarly to analyze listening patterns (Alzu'Bi et al., 2018). It helps in identifying songs, artists, or genres that are frequently listened to together, enabling the service to create more accurate and appealing playlists for its users. This not only enhances user engagement but also aids in discovering new artists or tracks, thus enriching the overall user experience.

A prototype for agricultural output is placed utilizing a collaborative multi-agent system intended for data mining (Belabed et al., 2019). The approach combines two data mining model functions: clustering variables to form homogeneous groups of attributes and extracting association rules within each group. It's used a multi-agent framework to incorporate these data mining techniques, using both the Apriori and genetic algorithms to extract association rules.

2.4. Recommendation Systems for Groups (GRS)

When it comes to suggesting a movie, restaurant, or travel destination to a group of people, Group Recommender Systems play a crucial role in identifying pertinent items from an almost limitless inventory based on group preferences rather than individual preferences (Kagita et al., 2023).

The methods currently in use for individual recommendations, however, are inadequate to handle the dynamics of group decision-making in scenarios that call for recommendations for larger groups (Alves et al., 2024; Alves, Martins, Saraiva, et al., 2023). To expand personalized recommendations to group recommendation systems, several strategies have been put forth recently, with the main goal being to increase recommendation accuracy.

Research has recently moved its focus to developing transparent group recommendation models that prioritize accountability and comprehensibility considering recent advancements in algorithmic science (Kagita et al., 2023).

2.4.1. Strategies for Incorporating Individual Preferences in GRS

These models seek to be accurate while offering extra benefits like sensitivity, confidence measures, or explanations (Kagita et al., 2023). One particularly interesting aspect of these improvements is the addition of a confidence measure to the recommendation set. The system's reliability is improved, and users are assisted in accurately and quickly recognizing the products of their choice by the confidence measures, which express the system's confidence that the desirable items are included in the recommendation set.

Some of the strategies involves individual preferences to achieve group recommendations. Table 4 presents some of the strategies of individual preferences in groups. (Boratto et al., 2015; Kagita et al., 2023).

Table 4 - GRS Strategies

Strategy	**How it works**
Plurality Voting	The item with the most votes is chosen
Average	Averages individual ratings
Multiplicative	Multiples individual ratings
Borda Count	Calculates points based on how items are ranked on each person's preference list; the item at the bottom receives zero points, the item next to it receives one point, and so on.
Copeland Rule	Calculates the frequency with which an item wins (by majority vote) less the frequency with which it loses.
Approval Voting	The number of people who have given the item ratings higher than a certain level of approval
Least Misery	Takes the minimum of individual ratings
Most Pleasure	Takes the maximum of individual ratings

Average without Misery	Averages individual ratings after eliminating products whose individual ratings fall below a predetermined level.

2.4.2. GRS across Industries

Group Recommendation Systems (GRS) are being used across several domains to improve shared experiences and decision-making. In areas such as music, movies, travel, and restaurants, GRS seeks to integrate individual interests to provide recommendations that are most appropriate for the entire group.

For example, MusicFX uses advanced algorithms and collaborative filtering techniques to provide tailored music suggestions to groups of users, such as friends, family, and coworkers (J. F. Mccarthy & Ha~ost, 1998). MusicFX generates a playlist that fits to the group's collective preferences by assessing each member's individual preferences and recognizing shared interests.

PolyLens is a group recommender expansion to the MovieLens system (Pnnz et al., 2001). Users create groups, and the system suggests movies from the MovieLens system to these groups, aiming to satisfy less satisfied members.

Collaborative Advisory Travel System (CATS) is a collaborative group recommender system designed to help a group of up to 4 friends plan and arrange their skiing vacation (K. Mccarthy et al., 2006). This GRS uses the concept of a shared collaborative area for a group of users with access to their own personal spaces. Individual user feedback is used to update explicit user models, both per individual and globally.

Of course, we also have the Grouplanner system, a group recommender system that provides travel recommendations using the tourists' personality to predict their preference for tourist attractions and applying intelligent agents with microservices (Alves et al., 2022).

These systems improve the group experience by measuring individual preferences and detecting shared interests, whether in music, movies, vacation planning or travel recommendations, resulting in a fulfilling and personalized experience for all members.

2.5. Multi-Agent Systems

According to Wooldridge, agents are computer systems that are placed in an environment and have the ability to act autonomously within it to achieve their intended goals (Wooldridge, 2002).

A multi-agent system (MAS) is a network of independent agents, each having the capability to perceive, reason, and act, without being directly controlled by the others, in an environment. MAS is gaining increasing importance in Recommendation Systems because it has the capability to emulate complex human behaviors and interactions in dynamic environments (Barrientos & Luevano, 2022).

Agents are distinct, specialized entities that solve problems within defined parameters (Qasem et al., 2022). They also exhibit adaptable and proactive behavior in the pursuit of a specific goal. In MASs, agents are autonomous when it comes to their ability to operate on their own without supervision. These agents oversee their own behavior and internal states. An agent has the autonomy to decide whether to carry out a request. Agents use adaptable problem-solving techniques to accomplish the goals that are intended for them.

MAS focusses on the problem of recommendation scenarios that include multiple users, changing preferences and changing contexts (Barrientos & Luevano, 2022). This flexibility makes MAS a useful tool in the dexterity of improving recommendation accuracy and customization. According to studies carried out in respect to the subject, there is an increasing interest in employing MAS in the advanced and situational-based recommendation approaches.

Warkentin, Sugumaran, and Sainsbury undertake a study on the use of intelligent agents and data mining to manage electronic partnerships (Warkentin et al., 2012). They identify the intricacies of information flows between economic actors in electronically mediated marketspaces and emphasize data mining's potential for revealing useful management insights with the use of intelligent agents. The study introduces a Generic Agent-based Data Mining Architecture (GAMA) that can be customized to help with decision-making and problem-solving in a networked economy.

2.5.1. Agent Features

Usually, agents work in a complex, dynamic, non-deterministic environment (Qasem et al., 2022). There are no presumptions about global control, data centralization, or synchronization in MAS environments. As a result, it is assumed that agents in a MAS performs with incomplete knowledge or constrained resources to address the issue. Agents coordinate their actions and improve interoperability by exchanging information through communication. Requests for information, specific services, or an action to be carried out by other agents, as well as matters pertaining to cooperation, coordination, and/or negotiation to set up related activities, can all occur during interactions between the agents.

Agents must handle interactions flexibly, making decisions at runtime about when and how to initiate interactions as well as the type and extent of these interactions, since they are flexible problem solvers with only limited control and knowledge of the environment in which they operate (Paricio García et al., 2010). Agents can solve complex tasks and have wide applicability with the following features:

- **Sociability**: To improve performance in reaching their goal, agents can ask other agents for information and share their own knowledge.
- **Autonomy**: Every agent has the ability to make decisions on its own and carry out the necessary actions.
- **Proactivity**: Each agent uses its past, sensed parameters, and information from other agents to predict potential future actions. This allows agents to take efficient actions that achieve their goals. This capability leads us to believe that, depending on the surroundings in which it is placed, the same agent may adopt dissimilar behaviors.

Even though autonomous agents are capable of acting, their true potential can only be realized in cooperative efforts with other agents (Jabla et al., 2022). MASs are groups of agents that work together to solve a challenging task. This is especially difficult given the decentralized nature of MAS and the fact that the effective coordination among agents is warranted during the data mining process.

2.5.2. MAS Features

MAS features are outlined in this section, along with a discussion of the various classifications that result from taking each feature into account (Dorri et al., 2018):

- **Heterogeneity**: Based on agent heterogeneity, MAS can be classified into two groups: homogeneous and heterogeneous. Uniform MASs consist of agents with identical features and functionalities, whereas heterogeneous MASs are made up of agents with different features.
- **Topology**: The location and relationships between agents make up topology. There are two types of MAS topology: static and dynamic. Over the course of an agent's lifetime, the position, and relationships of an agent in a static topology remain constant. An agent's position and relationships alter in a dynamic MAS topology when it travels, enters, or exits the MAS, or forms new connections, or relations, with other agents.
- **Data transmission frequency**: Agents sense their surroundings and communicate with other agents via time- or event-triggered data sharing. The agent continuously senses its surroundings in the time-triggered approach, gathers data, and communicates all newly sensed data to other agents at predetermined intervals. The agent in the event-triggered approach only detects its surroundings when a specific event occurs.
- **Agreement parameters**: In some MAS applications, agents must reach a consensus on specific metrics, or parameters. MASs can be categorized as first, second, or higher order based on how many metrics they have. Agents cooperate in the first order to settle on a single metric.
- **Leadership**: We consider the possibility of a leader, or an agent that assigns tasks and sets objectives for the other agents based on a single global goal. MAS can be classified as leaderless or leader follow, depending on whether such a leader is present.

2.5.3. Apriori Algorithm Implementation in MAS

In a MAS environment, Apriori can be used to analyze behavior amongst agents, such as capturing common purchase decisions by the users while predicting similar future actions or preferences (Jabla et al., 2022). Some of the tasks within Apriori implementation entail managing data distribution across agents and ensuring information exchange to facilitate efficient mining of association rules. This is especially difficult given the decentralized nature of MAS and the fact that the effective coordination among agents is warranted during the data mining process.

The combination of MAS with the Apriori algorithm provides Recommender Systems with a distinct set of advantages (Ait-Mlouk et al., 2016). One of the primary advantages is its flexibility in changing contexts. This versatility is critical in today's RS, when user preferences and situations are always changing. The flexibility of MAS enables the modeling of complicated, real-world scenarios involving several users with varying demands and preferences.

Another key advantage is the capacity to analyze data in parallel and distributed fashion. MAS are built to spread jobs across several users (Belabed et al., 2019; Ramar & Elamparithi, 2016). When combined with the Apriori technique, this capability allows for the efficient processing of big datasets, which is frequent in RS. This parallel processing power not only accelerates computing but also enables more complicated and nuanced data analysis, which is required for deriving valuable insights from user data.

These benefits, however, come with their own set of disadvantages. The intricacy of implementation is one of the key obstacles (Ramar & Elamparithi, 2016). The Apriori algorithm must be integrated into MAS by coordinating the activities and data exchange of several autonomous agents. To guarantee that the data mining process is precise and efficient, a high degree of synchronization is required. The decentralized nature of MAS adds to this complexity because each agent functions autonomously, making coordination difficult.

Finally, the work by Addi Ait-Mlouk, Fatima Gharnati, and Tarik Agouti addresses the public health issue of road accidents by using data mining techniques to uncover the key characteristics that contribute to crash severity (Ait-Mlouk et al., 2017). The study focuses on using association rule mining to forecast future accidents, allowing drivers to avoid potential hazards. For that, it's used a multi agent system to manage and model the quality measurement. The problem of

managing a large number of decision rules created by this technique is solved by incorporating a multi-criteria decision analysis (MCDA) approach.

2.6. MAS, Microservices, and Multi-Agent Microservices

2.6.1. Microservices

Software applications can be designed as a collection of independently deployable services, which are known as microservices (Baresi & Garriga, 2019). Services become micro in terms of their contribution to the application, different versions can coexist, and the system's actual topology can be changed at runtime as needed. Every single microservice component needs to be able to be modified independently of the others' functionality and performance.

Microservices is an architectural style that is an evolution of the classic service-oriented architecture style that emphasizes breaking the system up into small, lightweight services that are specifically designed to carry out a very cohesive business function (Alshuqayran et al., 2016). Increased agility, developer productivity, resilience, scalability, dependability, maintainability, separation of concerns, and ease of deployment are among the generally acknowledged advantages of this style.

A few fundamental concepts form the foundation of the microservice architecture (Dragoni et al., 2017):

- **Limited Framework**: Combining of related functionalities into a single business capability, which is implemented by each microservice. The system structure and business capabilities are perfectly aligned in this way, which makes it simple to update or fix certain features, such as knowing where they are.
- **Size**: One of the main differences between microservices and earlier service-oriented architectures (SOAs) is their emphasis on small size. The conventional application of microservice architectures proposes that a service that has grown too big should be divided into two or more smaller services in order to maintain focus on offering a single business capability. A small service can be quickly modified and, if necessary,

rebuilt from scratch with limited resources and in a short amount of time, which is a major benefit of its small size in terms of extendibility and maintenance.

- **Independency**: This idea states that every microservice in a microservice architecture is operationally independent of every other microservice and that the only way for services to communicate with each other is through their published interfaces, which promotes loose coupling and high cohesion. This is essential because, as long as the interfaces are maintained, it is possible to modify, repair, or upgrade a microservice without jeopardizing the accuracy of the system.

2.6.2. Multi-Agent Microservices

Multi-Agent Micro-Services and microservices are similar in many ways, to the extent that both approaches can be broadly defined as being concerned with the creation of loosely coupled distributed systems composed of small independent components with internal state (Lillis, 2020). This similarity suggests that approaches that are at least somewhat consistent with the MAS perspective are beginning to emerge in the industry, even though the two approaches differ in many other ways, one of which is the application of practical reasoning. Table 5 compares Microservices principles to MAS.

Table 5 - Microservices principles vs MAS (Collier et al., 2019; Lillis, 2020)

Principle	Microservices	MAS
Bounded Context	A single piece of business functionality is represented by a microservice	An agent can take on one or more roles
Size	Microservices ought to be sufficiently compact to guarantee extensibility and maintainability	Size and complexity are not problems and rely on the target domain
Isolated State	There is minimal state information shared across services	For an agent, the state is local and private, essential for their autonomy
Distribution	Services are dispersed among multiple nodes	Although agents are dispersed logically, it is expected that they will be over a number of nodes

Elasticity	The application is made to enable the addition and deletion of necessary resources in runtime	Adding and removing agents during runtime is a key component
Automated Management	Management tasks like scaling and addressing failures	Agents do not focus on management tasks, although they are occasionally taken into consideration
Loose Coupling	Systems break down into highly coherent collections of loosely connected situated services	Agents are independent, loosely connected problem solvers

There are a few similarities and differences between Microservices and MAS. Regarding the isolated state, distribution, elasticity, and loose coupling, both strategies are similar. The notion of agents as independent decision-makers gives rise to the isolated state and loose coupling, whereas the concept of agents as social entities creating a loosely coupled network of problem solvers gives rise to distribution. Table 6 compares MAS principles to microservices.

Table 6 - MAS principles vs Microservices (Collier et al., 2019)

Principle	Microservices	MAS
Autonomy	Microservices have some degree of internal state control and function without direct human interaction	Agents have some degree of agency over their behavior and internal states and function without human involvement
Social Ability	Microservices communicate with one another over HTTP and RESTful API messages	Agents interact with other agents using some kind of Agent Communication Language
Reactivity	Microservices promptly reply to incoming queries made via HTTP	Agents observe their surroundings and react quickly to changes that take place in them
Proactivity	Microservices do not take the initiative	Agents take the initiative rather than passively react

Multi-Agent Systems agents take the initiative and react to changes in their environment, while microservices concentrate on handling HTTP requests (Lillis, 2020). The ability to balance

the autonomy and proactive qualities of Multi-Agent Systems with the modularity and automation strengths of Microservices would be necessary to implement multi-agent microservices.

Some real examples related to MAMS are found in a rule-based eCommerce methodology for the IoT using Intelligent Agents and Microservices (Kravari, 2018). This study combines Agents with the microservice architecture to deal with Internet of Things heterogeneity while it adopts the use of a social agent-based trust model, implemented in Emerald (Kravari et al., 2010).

Other example presents a multi-Agent system solution to energy management in a microgrid based on distributed hybrid renewable energy generation and distributed consumption (Boudoudouh & Maâroufi, 2018). The real model of each element connected is needed, enabling microgrid modeling and control.

In the field of recommendation systems, the integration of advanced algorithms with multi-agent systems (MAS) has grown in significance. The problem of giving individualized recommendations to groups of users with varying preferences has prompted researchers to investigate novel ways. This chapter examined the relevant literature, focusing on prior research that used intelligent agents and association rule mining in a variety of fields, and places our project within that perspective.

3. Solution analysis and design

This chapter illustrates the process of understanding the problem, which includes an examination of the domain of the system built. In this chapter, we present an overview of the solution design, business rules and the actors of the system, as well as the requirements, including the main technologies employed, which played an important role in the development process and was constantly integrated with Azure technologies.

3.1. Grouplanner Architecture

Grouplanner was built using .NET technologies, since it is a prominent object-oriented programming framework for developing web services and compatible cross-platform applications (Alves et al., 2022; Microsoft, 2024). Figure 4 presents the Grouplanner Microservices Architecture composed of 5 microservices.

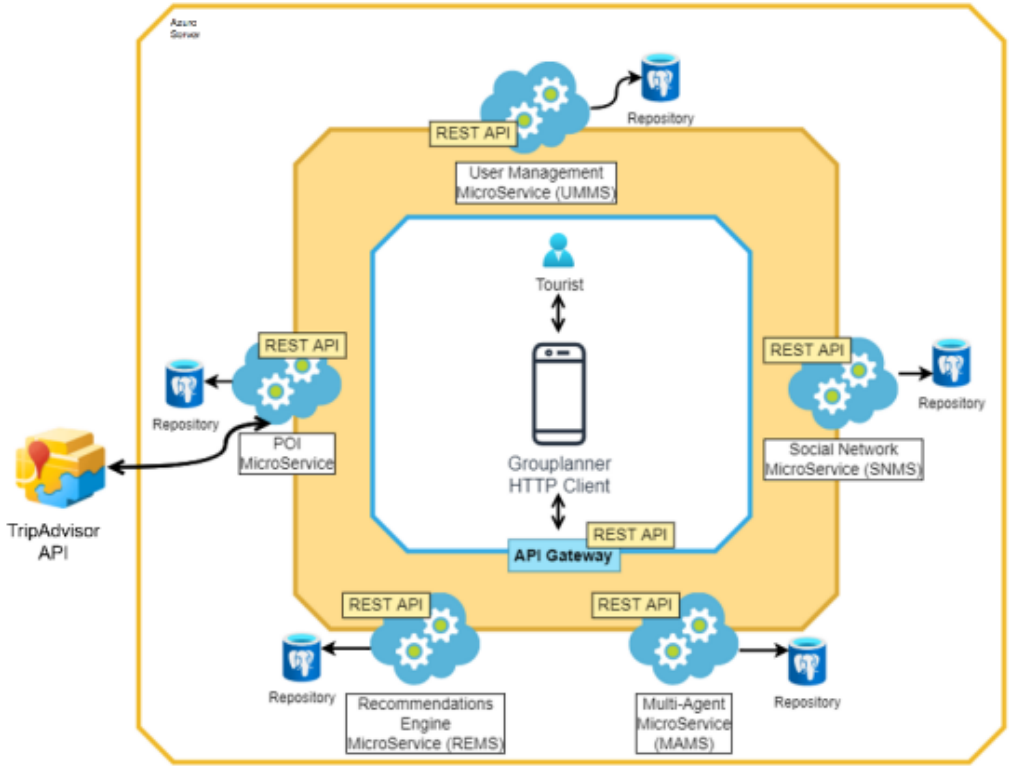

Figure 4 - Grouplanner Microservices Architecture, retrieved from (Alves et al., 2022)

The User Management Microservice (UMMS) handles and stores the data of registered users. This covers demographic information, personality traits, and physical disabilities. It also handles tourist attraction choices, travel preferences and concerns, records of visited and unvisited points of interest (POI), rated POI, and other profile-related data.

The Multi-Agent Microservice generates agents in the MAMS environment whenever a new tourist user registers or a new excursion group form. Each user is represented by an intelligent agent modeled with their profile.

MAMS is responsible for creating dynamic clusters of tourists with similar personalities and, in the case of excursion groups, subdividing the main group into smaller subgroups depending on these personality clusters and the individual personalities of the tourists on the excursion. Also, it is expected to offer the ability to suggest or discourage points of interest to the Recommendation Engine Microservice (REMS) based on individual or group characteristics and preferences.

REMS aims to discover the best points of interest (POI) to recommend to tourists, using the Weather API. This is accomplished by using a POI ontology and considering aspects such as tourist attraction preferences, travel-related preferences and concerns, disabilities, fears and phobias, and MAMS-identified rules.

The POI Microservice (POIMS) provides all the POI that can be proposed to users, and these are obtained using the TripAdvisor API, which currently is limited to POI in northern Portugal. This microservice handles all POI-related actions to display updated POI information.

Finally, the Social Network Microservice (SNMS) is responsible for organizing group and subgroup chats. It makes ensuring that data messages between groups and subgroups is accessible and displayed in the user interface.

All microservices are already developed in .NET with C# as programming language and hosted in ElephantSQL. MAMS is the focus of the implementation and was upgraded from .NET 3.1 to .NET 8 to be in long term support by Microsoft.

3.2. Domain Model

This section introduces the domain model that explains the MAMS Service system. This model outlines the important elements and their interactions to fully understand the system's business rules and behavior. The **Rule** concept was the only concept added to the existing domain model to represent the rules created by the Apriori algorithm. Some other representations were slightly modified, such as the **Cluster** now having the average personality traits of the users that it represents, and some **User** characteristics, such as the birthday, professional situation, formation area and education level. These concepts are illustrated in Figure 5.

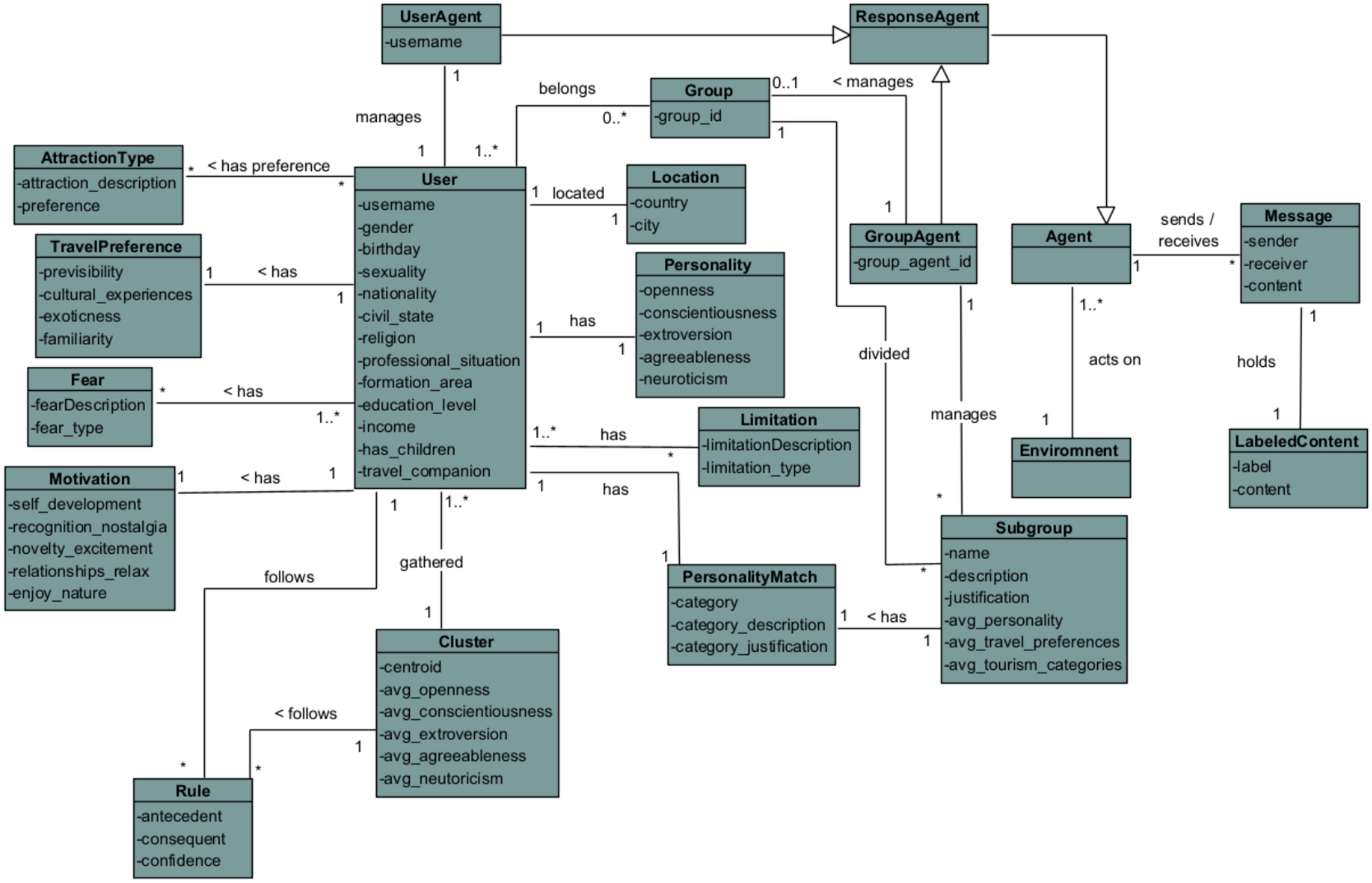

Figure 5 - MAMS Domain Model

A tourist is represented by a **User** in search of recommendations of POI, with the necessary attributes for points of interest recommendation requests. A **User** has also some characteristics for better and personalized requests such as the **Motivation**, **TravelPreferences** and **AtractionType** that represent, respectively, **User** motivations on travelling, travel preferences scores aspects and the preference of attraction types, which is the category of a set of POI. Also, it has a **Location** representing the country and city where the **User** lives.

Furthermore, a set of **Users** can be gathered in a **Cluster** based on their **Personality** scores based on the Big Five model (Alves, Martins, Saraiva, et al., 2023; Soto & Jackson, 2013). Either the **Cluster** or **Users** can follow a set of **Rules**, which are relative to the **User** attributes, their **Fears**, **Limitations** and their **AttractionTypes** preferences. **AttractionTypes** are categories of a list of POI. The antecedent in a **Rule** provides the condition or objects found together ('if' part), and the consequent explains what is predicted or related ('then' part).

A **Group** is a collective of **Users** who participate in excursions and can be divided into **Subgroups** based on their personality. **PersonalityMatch** provides data on the characteristics of **Users** or **Subgroups** based on the average of their **Personality**, **Motivations** and **TravelPreferences**.

Regarding the multi-agent functionalities, some domain concepts are adopted and managed using the **ActressMAS** framework (Alves et al., 2022). **ActressMAS** is a .NET open-source framework for MAS development that aims to reduce the complexity of language configuration and learning for agents with varying technologies, allowing communication by direct message exchanges between agents (Leon, 2022).

An **Agent** is an entity tasked with managing information and executing assigned tasks. The **Environment** represents the context in which the **Agents** operate. A **ResponseAgent** is a type of **Agent** that can send or receive messages inside or outside of its current **Environment**. **UserAgent** and **GroupAgent** are a type of **ResponseAgent** that represents a **User** and the **Group** or **Subgroup**, respectively, regarding the functionalities of communication between them. Internal and external communication among **Agents** is carried out via **Messages**, which subscribe to a standardized format defined by **LabeledContents**.

To detail each entity more accurately, in Table 7 is presented the glossary of the domain model.

Table 7 - Domain Model Glossary

Entity	Description
Agent	Manages information and executes tasks within the agent ecosystem.
AttractionType	Categories of Points of Interest (POI) preferred by tourists
Cluster	Tourists are categorized together depending on their personality metrics
Environment	The context in which agents act
Fears	Specific fears or phobias of a tourist.
Group	Collective of tourists participating in excursions
GroupAgent	Manages information and tasks related to excursions of a group or subgroup
LabeledContent	Labels the messages to ensure an organized communication format
Limitations	Physical limitations or constraints of a tourist
Location	Country and city where the tourist reside
Message	Communication channel among agents
Personality	Traits based on the Big Five model (Soto & Jackson, 2013) (Openness, Conscientiousness, Extraversion, Agreeableness, Neuroticism)
PersonalityMatch	Characteristics of tourists or subgroups of tourists based on their average personality, motivations, and travel preferences.
ResponseAgent	Type of agent that can transmit and receive messages both within and outside of its environment.
Rule	Norm followed by clusters or a set of tourists (users), based in their attributes and preferences as a whole
Subgroup	Subset of a group made up of tourists with similar personalities.

User	Tourist seeking recommendations on Points of Interest (POI)
UserAgent	Represents a user so that he can communicate with other agents.

Ultimately, this domain model defines the components of the MAMS system, including tourist preferences and personality measurements inside a multi-agent framework, and uses rules to improve user recommendations and **ActressMAS** to ensure efficient communication and task management among agents.

3.3. System Actors

An actor describes the role that a user or other system plays when interacting with a subject (Rosenberg & Stephens, 2007). It may reflect the roles of human users, external hardware, or other subjects. Actors are always outside of the system and interact with it directly by initiating a use case, providing input, and/or receiving response. While a single physical instance can play the role of separate actors, an actor does not always represent specific physical entities.

In MAMS prototype (Alves et al., 2022), the actors include the System Administrator and the System (Alves, Martins, Novais, et al., 2023). The system administrator is an actual user of the application with administrator roles to and the 'System' represents automated use cases that are automatic and needs no influence of a concrete actor. Table 8 presents the MAMS actors that interact with the new functionalities.

Table 8 - MAMS Actors Overview

Actor	Description
System Administrator	Requests the generation of association rules by the Apriori algorithm.
System	Generates of association rules by the Apriori algorithm
	Clusters users based on personality
	Creates POI Inclusion/Exclusion List of an individual or a group

Although these are the only actors that are able to cause the new features, some other entities interact with these new MAMS functionalities.

For context, the Agent, representing a Tourist/User, is the main user of the Grouplanner application and is able to register in the MAMS application, where the 'System' automatically will cluster the users based on personality automatically. Also, the Tourist can request a POI recommendation, where the Recommendation Engine will take advantage of the POI Inclusion/Exclusion list generated by the 'System' to tailor their final recommendations. MAMS will request the recommendation to RE and then provide to the UI the tailored recommendations of a group or an individual.

MAMS had already implemented the feature of requesting recommendations to RE but didn't send any POI to include or exclude based on patterns found. Also, POIMS provides all the POI that MAMS requests based on the attraction type categories. That communication by endpoints between entities is also already implemented.

3.4. Requirements engineering

Understanding and defining the system's purpose of software development involves conducting an extensive investigation of the problem to gather and document the relevant requirements. This approach ensures that the final software product meets the needs and expectations (De Lucia & Qusef, 2010).

During this step, the requirements are separated into functional and non-functional categories. Functional requirements are deducted trough a Quality Function Deployment (QFD) to then specify the behaviors and functionalities that the system must accomplish, whereas non-functional requirements establish the system's operating features and restrictions.

3.4.1. Functional Requirements

QFD is an approach for translating client needs into technological features and procedures. Its goal is to guarantee that customer demands are properly incorporated into product and service design, development, and delivery. QFD assists firms in aligning their goods and services with

market expectations by breaking down client requests into actionable aspects such as quality, dependability, technology, and pricing (Zare Mehrjerdi, 2010).

A discussion with the project client (supervisor and co-supervisor) resulted in the practical implementation of the QFD model as a means of documenting the client's demands, ensuring that the expectations are translated into quality characteristics. Furthermore, during the same meeting, the importance of each demand was assessed and specified, resulting in a prioritizing of the client's requirements based on relevancy, presented in Figure 6.

1: low, 5: high	Desired direction of improvement (↑,0,↓)	↑	↑	↑	0	
Customer importance rating	Functional Requirements (How's) → / Customer Requirements - (What's) ↓	Implementation of Apriori Algorithm	Cluster Users Integration	POI selection for matching Individual preferences	POI selection for matching Group preferences	
3	Provide Inclusion/Exclusion list for Individual	7	6	9		
5	Provide Inclusion/Exclusion list for Group	7	6		9	
5	System scalability	4	6			
5	System reliability	8	5	5	8	
4	Error handling	1	1	6	9	
5	Optimized response times	9	8	6	6	
	Technical importance score	165	147	106	151	569
	Importance %	29%	26%	19%	27%	100%
	Priorities rank	1	3	4	2	
	Difficulty	5	4	2	2	1: very easy, 5: very difficult
	Cost and time	5	4	1	2	1: low, 5: high

Figure 6 - QFD Model

The importance in each User Story defined was then translated into the MoSCoW prioritization technique (Jahan et al., 2019). It categorizes requirements as Must have (essential), Should have (important but not critical), Could have (desirable but not necessary), and Won't have (not a priority). If above or equal the average importance of 25%, it is considered a MUST, 15-24% a Should, between 5-15% a Could and below 5% a Won't.

Therefore, based on QFD, it covers the MAMS use cases. A use case shows how a system interacts with other users or systems to accomplish specified tasks (Rosenberg & Stephens, 2019). It outlines the processes or activities required for the system to complete a service to a user. Use

cases assist to explain system functionality by exhibiting user-system interactions, making them critical for understanding requirements and ensuring that all functions are managed during development. Based on that, the following Use Case diagram is represented in Figure 7.

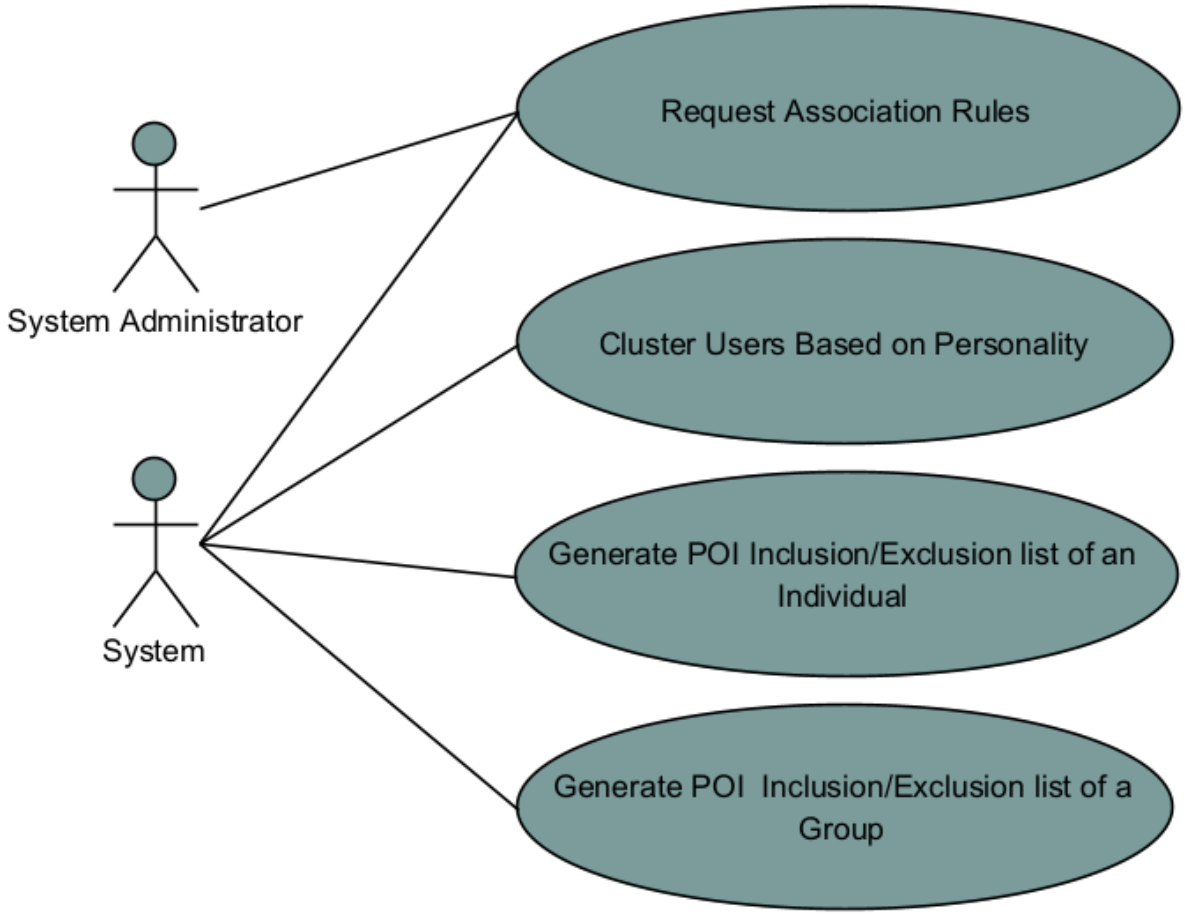

Figure 7 - Use Case Diagram

The System Administrator requests MAMS for association rules using Apriori. This feature can also be automatically done by the System, representing MAMS itself. This means that any time MAMS needs updated rules, this use case will generate context-specific rules based on the agents.

Cluster Users based on Personality, where the system clusters users based on their personality, using the d-means dynamic algorithm (Alves et al., 2024). Even though this functionality was already developed, the algorithm was found not optimal for the client requirements, therefore, the

feature was planned to be reimplemented by scratch and is essential for users to be clustered in clusters and, consequently, will have impact following use cases to develop.

Finally, whenever a recommendation is requested by the Tourist, the system automatically requests POI Inclusion/Exclusion lists, either to group or individual preferences, send it to REMS. Based on the rules that Apriori generated, REMS receives from MAMS POI to include, in case of a positive POI preference of a group or individual, or exclude, in case of a negative POI preference of a group or individual. The request of the recommendation to REMS is already implemented; thus, the use case focuses exclusively on supplying POI to include or exclude based on the rules found.

The functional requirements were written using the User Story model (Raharjana et al., 2021). A User Story is a brief, semi-structured statement that describes needs from the actor's perspective. A user story can clarify consumer desires or product descriptions. It has three aspects: who, what, and why. The "who" refers to the system user or actor, "what" is the actor's desire, and "why" is the reason (optional in user stories).

For each user story, the actors and/or external systems are identified with how they interact with the application's functions, following the IEEE/ANSI 830 standard (IEEE Computer Society. Software Engineering Standards Committee. & IEEE-SA Standards Board., 1998). Documenting these functional requirements created a roadmap for the build and operation of the functional requirements. In more detail, the following requirements are presented in Table 9.

Table 9 - Use Case Overview

Functional Requirements	Overview
MAS-US-01 Cluster Users Based on Personality	As a **System**, I want to automatically group and/or regroup users into clusters based on their personality using the Big Five model, detailed in section 3.2, so that the users are clustered with similar personality
MAS-US-02 Request Association Rules	As a **System Administrator** or the **System**, I want to request the Apriori algorithm to analyze user characteristics and preferences so it can develop association rules by cluster or globally to find preference patterns between tourists

MAS-US-03 Generate POI Inclusion/Exclusion list of an Individual	As the **System**, I want to provide a POI inclusion/exclusion list for a single user based on association rules so RE can improve POI recommendations
MAS-US-04 Generate POI Inclusion/Exclusion list of a Group	As the **System**, I want to provide a POI inclusion/exclusion list for a group of users based on association rules so RE can improve POI recommendations

MAS-US-01 Cluster Users Based on Personality:

- **Description** - MAS-US-01 is a system automated functionality, since the clustering is processed automatically after a new tourist is registered and all users are rearranged to the better fitting cluster based on the personality traits, using the specific Dynamic personality-based clustering, d-means (Alves et al., 2024). This means the system is autonomous on balancing the new user to a more suitable personality cluster while also considering potential changes to clustering of existing users.
- **Acceptance Criteria** – Each user shall be assigned, using d-means, to their closest cluster by estimating the Euclidean distance between the cluster average personality values and the user (minimum Euclidean distance should be set as 80%); existing users shall be reassigned into a better fitting cluster if it has a new one more suitable to the user; each user shall be assigned to exactly one cluster; clustering process should be automated for every new user registered in MAS.
- **Priority** – Based on the QFD in previous subsection, the priority to cluster users based on personality is a M (MUST), since is crucial to be included in the application in the implementation time interval estimated.

MAS-US-02 Request Association Rules:

- **Description** – In MAS-US-02, the System Administrator or System will request the generation of association rules using Apriori based on all tourist attributes and preferences, either in a cluster or globally, defined by a minimum cluster threshold. The preferences to be considered in the algorithm should focus on the categories of

the POIs, referred to as Attraction Types, rather than the individual POIs themselves. If the number of users in a cluster surpasses or matches the threshold, the rules will be applied individually to that user clusters data, while the remain users will have global rules generated. These association rules are also generated based support and confidence bound percentage values and it becomes the landscape to suggest POI to individual or groups based on the rules created. Both the minimum cluster threshold, the support and confidence values, in case of no specific input of the System Administrator, shall be retrieved by a configuration file with predefined values. The System itself shall use the configuration file with the predefined values.

- **Acceptance Criteria** – The system must available an endpoint so the System Administrator can initiate the rules generation; rules must be created separately for each cluster that has the number of users associated equal or higher than the minimum cluster threshold defined; the global rules must account for all users that their associated cluster doesn't threshold the minimum cluster threshold; all rules must threshold both support and confidence values defined by the System Administrator; the system shall display the generated association rules with the respective confidence values for each one.
- **Priority** – Based on the QFD in previous subsection, the priority to generate association rules is a M (MUST), since is the landscape to the desired POI inclusion/exclusion lists.

MAS-US-03 Generate POI Inclusion/Exclusion list of an Individual:

- **Description** – The Recommendation Engine will request POI inclusion/exclusion lists to a single user, requested by its username. The goal is to verify possible association rules and match with the user attraction type preferences. If there is a match of characteristics regarding the preference of attraction types, the system should provide the corresponding POI to include, in case of a positive preference, or POI to exclude, in case of a negative preference.
- **Acceptance Criteria** – The POI list must have POI of attraction types of the matching rules with the user preferences or characteristics, either to include or exclude; the rules to prioritize are the user cluster, if any, otherwise, global rules should be followed; if

the user username is not found within MAS, a custom error handling should be displayed; if the rules were not found, a custom error handling should be displayed; if there is no rules, an empty list of POI to include or exclude should be returned; if there is no matching rules with the user traits and preferences, an empty list of POI to include or exclude should be returned.

- **Priority** – Based on the QFD in previous subsection, the priority to generate association rules is a S (Should), since is the primary focus is POI include/exclude lists for a group of users.

MAS-US-04 Generate POI Inclusion/Exclusion list of a Group:

- **Description** – The System will provide POI inclusion/exclusion lists to a group of users, requested by their usernames and an optional cluster identifier. This identifier serves to specify the cluster rules to follow, if any, otherwise, the global association rules shall be followed. Then, the goal is to verify possible association rules and match with the user attraction type preferences. If there is a match of characteristics regarding the preference of attraction types of at least half of the group, the system should provide the corresponding POI to include, in case of a positive preference, or POI to exclude, in case of a negative preference.
- **Acceptance Criteria** – The system must retrieve the POI list to include or exclude for a group of users with the following rules of a specific cluster, optionally; the POI list must have POI of attraction types of the matching rules with at least half the group preferences or characteristics, either to include or exclude; the rules to prioritize are the cluster identifier specified, if any, otherwise, the first user cluster should be followed, and if there is also no cluster rules, global rules should be followed; if group usernames are not all found within MAS, a custom error handling should be displayed; if the rules were not found, a custom error handling should be displayed; if there is no rules, an empty list of POI to include or exclude should be returned; if there is no matching or less than half of users match rules with the group traits and preferences, an empty list of POI to include or exclude should be returned.
- **Priority** – Based on the QFD in previous subsection, the priority to generate association rules is a M (MUST).

3.4.2. Non-Functional Requirements

The FURPS+ framework is a comprehensive model created exclusively for evaluating software quality (Samadhiya & Wang, 2010). This evaluative paradigm, which includes assessments of both functional and non-functional criteria, has five critical components. In this section, the focus will be on the non-functional requirements. The inclusion of the symbol '+' in the original FURPS model implies additional developments that go beyond its initial scope. The essential components include:

- **Functionality**: This aspect examines the software's capabilities and overall functionality. It verifies that the program consistently and precisely carries out its assigned responsibilities.
- **Usability**: Usability focuses on enhancing the user experience by addressing human-centered design, interface attractiveness, and coherence.
- **Reliability**: This dimension assesses the software's dependability by considering factors such as the frequency and severity of failures, recovery strategies and accuracy.
- **Performance**: Performance factors include the software's response time, speed, efficiency, availability, accuracy rate, and resource consumption. These components specify how well the software performs in various scenarios.
- **Supportability**: Supportability encompasses the ease with which the software can be tested, extended, adapted, maintained, and supported.

The '+' in FURPS+ signifies the inclusion of other essential requirements that impact the software's overall quality (Al-Qutaish & Ain, 2010):

- **Design Constraints**: These are limitations on the design of the software, such as regulatory requirements or specific technology choices.
- **Implementation Constraints**: These include considerations related to the coding and construction of the software, such as standards and guidelines for development.
- **Interface Constraints**: This involves specifications for how the software interacts with other systems or components.
- **Physical Constraints**: These pertain to the physical environment in which the software operates, such as hardware specifications and environmental conditions.

To evaluate the system's quality and integrity, the FURPS+ model was used to categorize non-functional requirements and ensure that the system met the needs of the consumer and client.

Functionality

- The authentication is done on the server side using JSON Web Tokens (Jánoky et al., 2018).
- The secure HTTPS protocol is used in the integration between the systems.
- A list of roles with permissions is defined in the system.

Usability

- Not applicable (n/a).

Reliability

- .NET Error handling (Cabral & Marques, 2007) with error messages.
- Validation mechanisms for input data, such as Data Annotations and Model Validation.

Performance

- Optimized response times, detailed in 5.1.

Supportability

- Functional features comply with the correspondent acceptance criteria.
- Documentation of new features, detailed in section 3.5.
- .NET version updated to long support version.

Design Constraints

- Not applicable (n/a).

Implementation Constraints

- MAMS must be developed in C# and .NET.
- The data model to be used must be generated through Entity Framework Core.

Physical Constraints

- The microservices must be hosted on the Microsoft Azure platform.
- The database used by each microservice must be hosted on ElephantSQL.

Interface Constraints

- Not applicable (n/a).

3.5. Design

The C4 model is a paradigm for visualizing software system architecture that was developed by Simon Brown to bridge the gap between high-level system overviews and low-level implementation details (Brown, 2015). The term "C4" refers to the four layers of detail it offers: context, container, component, and code. Each level provides a unique viewpoint on the system's architecture, allowing for a thorough understanding of how it is constructed and operated.

The **Context** level provides a high-level overview of the system, depicting it as a black box and demonstrating its interactions with external users, systems, and stakeholders (Brown, 2015; Vázquez-Ingelmo et al., 2020). This identifies the system's boundaries and dependencies.

The **Container** level divides the system into key containers, such as apps, databases, and microservices, and demonstrates how they interact and their roles (Brown, 2015; Vázquez-Ingelmo et al., 2020).

The **Component** level gives a thorough view of each container, dividing it into primary components and highlighting their interactions. This illustrates how components work together to accomplish the container's obligations (Brown, 2015; Vázquez-Ingelmo et al., 2020).

Finally, the **Code** level focuses on the implementation of a given component, indicating the actual implementation details (Brown, 2015; Vázquez-Ingelmo et al., 2020).

The C4 model, with its four layers of abstraction, provides an organized way to documenting and understanding software design (Brown, 2015). It guarantees that all stakeholders, including

non-technical business executives and technical developers, understand the system's design at the proper degree of detail.

3.5.1. Level 1 – Setting

The context diagram presented shows the high-level architecture of Grouplanner, including how it interacts with external systems and users. Figure 8 allows understanding the key elements involved and how they interact with one another, offering an overview of the system's environment.

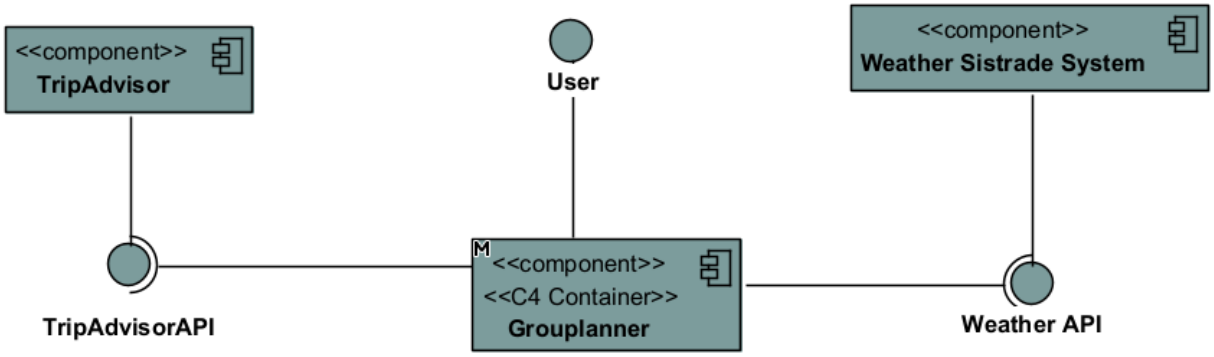

Figure 8 - Context Diagram of Grouplanner

The context diagram consists of three major components: TripAdvisor, Weather Sistrade, and Grouplanner, as well and a user representing a tourist that will interact with the system. Grouplanner will communicate with external APIs to obtain information for recommendation purposes. It connects to the TripAdvisor API to retrieve travel and destination information to help users plan their vacations. Furthermore, the Grouplanner retrieves weather-related data via the Weather Sistrade System API, giving users with live weather information relevant to their plans as well as personalized weather recommendations. The illustration illustrates the major interactions and data flow inside the system, highlighting its dependence on external services and the user's role in the overall architecture.

3.5.2. Level 2 – Container

In the context of the C4+1 model, this level shows the architectural view of the Grouplanner application's components and interactions. Figure 9 represents structural aspects that determine how different components work together to fulfill the application's objective.

Logic View

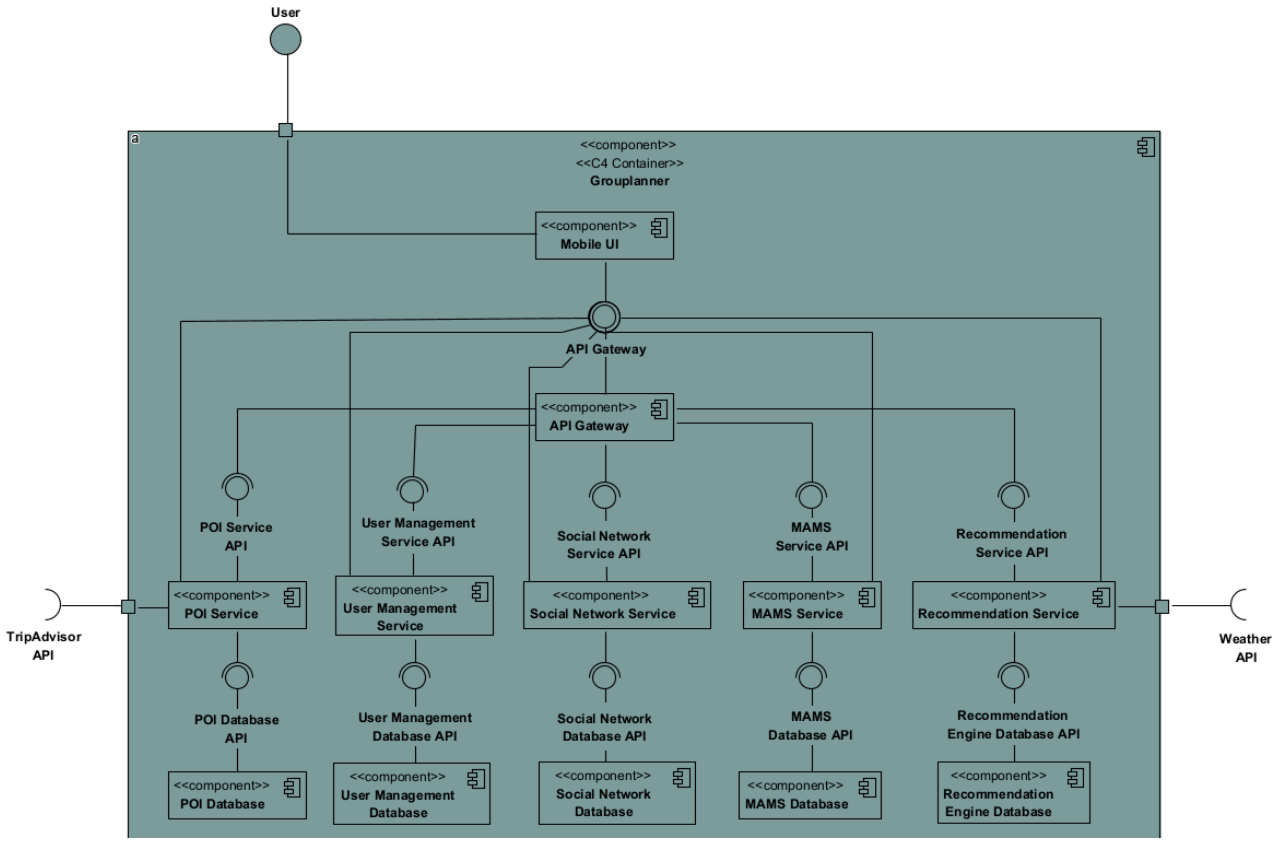

Figure 9 - Level 2 Logic View of Grouplanner

The diagram shows the many components of the Grouplanner system along with their relationships, providing a more in-depth look. The Mobile UI component works as the front-end application for user interaction with the system. All requests, both internal and external, enter through the API Gateway, which directs them to the proper services. All microservices were previously described in section 3.1 for context.

Process View

The following process views are related to the MAS-US-03 and MAS-US-04, explained in section 3.4.1 and detailed in chapter 4. Figure 10 represents a sequential diagram with the POI Inclusion/Exclusion list of an individual.

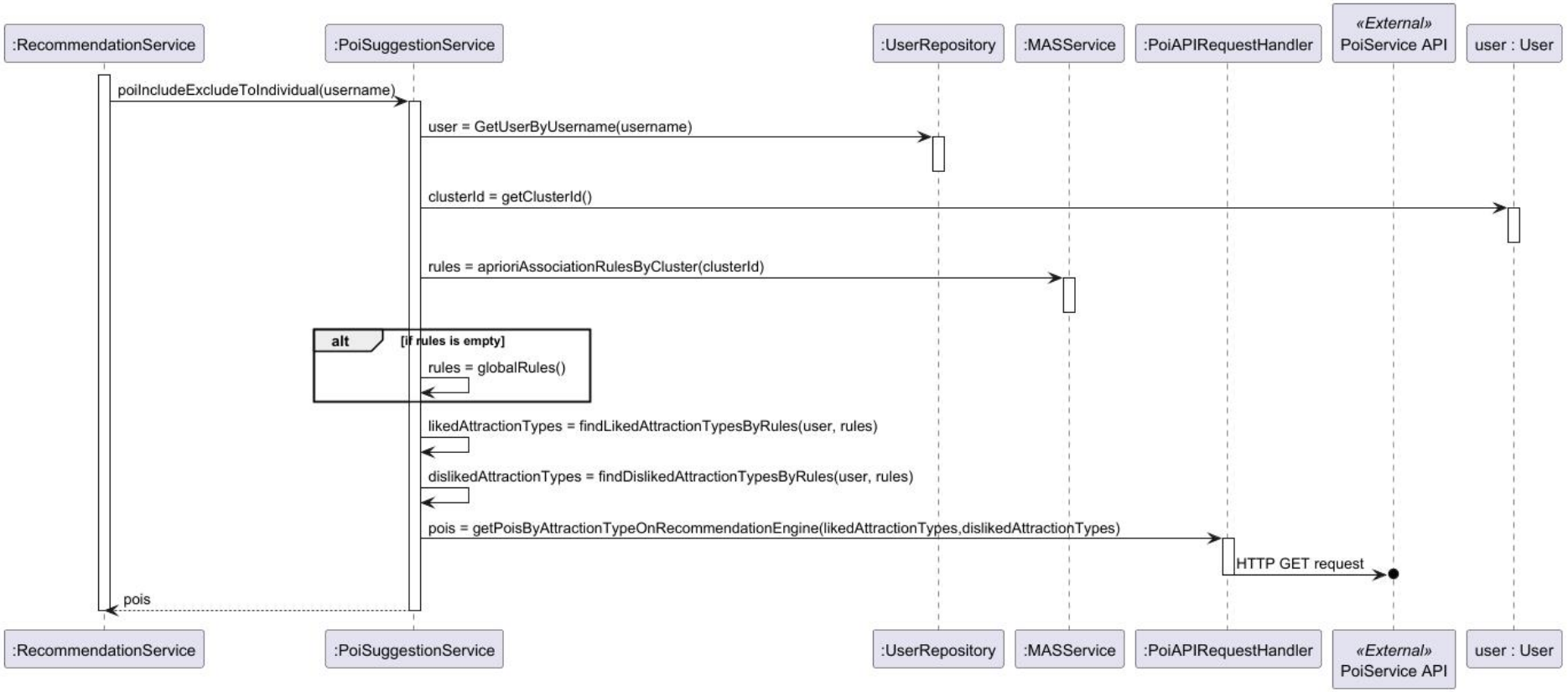

Figure 10 - MAS-US-03 Sequence Diagram

This process view corresponds to the creation of liked and/or disliked POI that will be send to REMS based on the rules related to characteristics of a single tourist. The procedure invokes a service function including a username of the respective tourist. The service retrieves user information, including the user's cluster identifier, and uses this identifier to obtain the cluster association rules. If no specific cluster rules are found, global rules are applied. According to these guidelines, liked and disliked attraction types are identified. The service then gets from POIMS the POI of the respective attraction types and are redirected to the ‘Recommendation Service’ already implemented that consequently will send the POI found to REMS.

Figure 11 corresponds to the sequence diagram of the creation of liked and/or disliked POI that will be send to REMS based on the rules related to characteristics of a group of tourists.

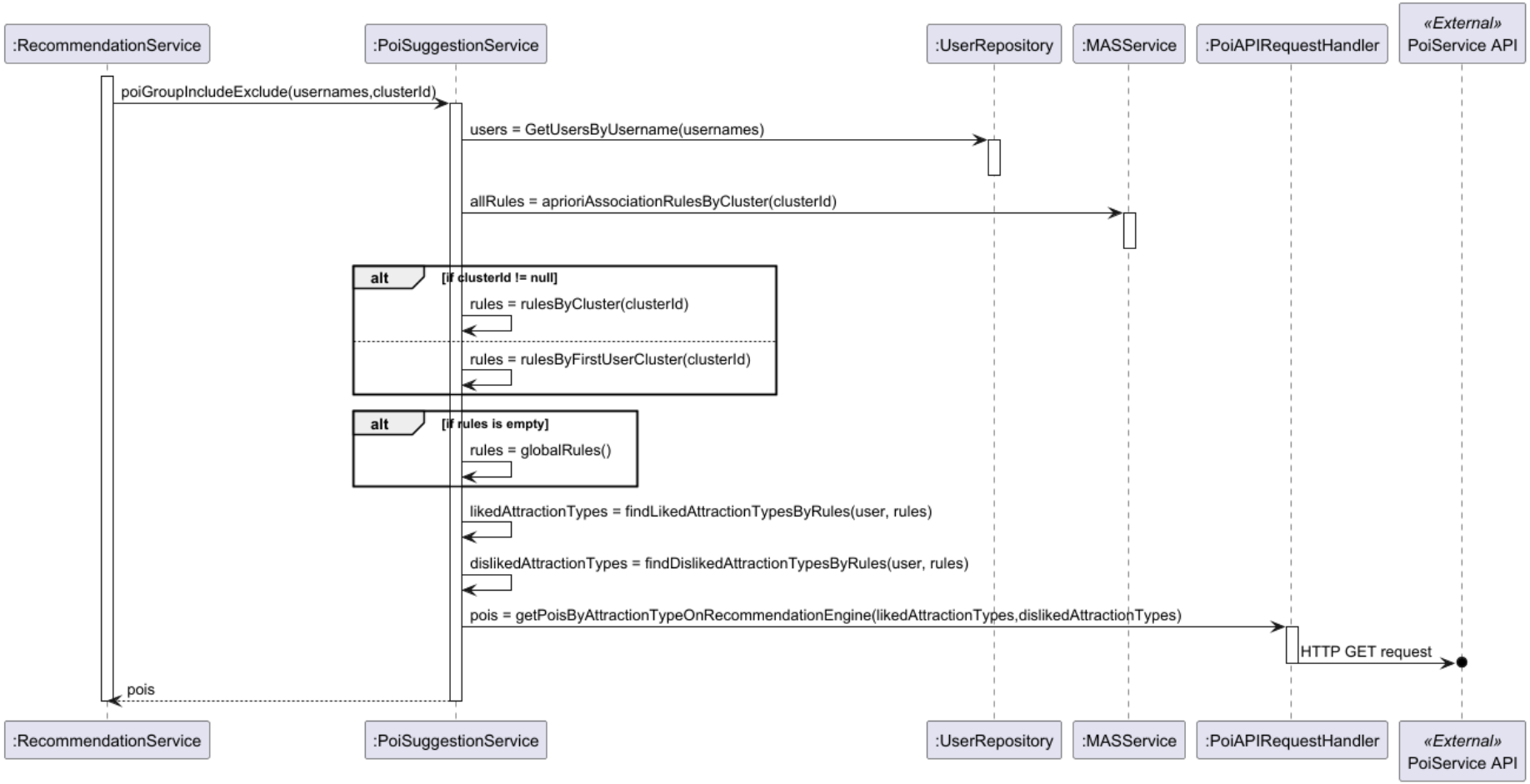

Figure 11 - MAS-US-04 Sequence Diagram

The process invokes a service function using usernames and a cluster identifier of the respective group. The service retrieves cluster-specific user information and association rules. If the cluster identifier is not null, cluster-specific rules are applied; otherwise, the first user's cluster rules are applied. If no rules are identified, global rules are used. According to these guidelines, liked and disliked attraction types are identified. The service then requests the POI of the respective attraction types to POIMS.

Implementation View

Figure 12 illustrates the main dependencies between the Grouplanner microservices architecture.

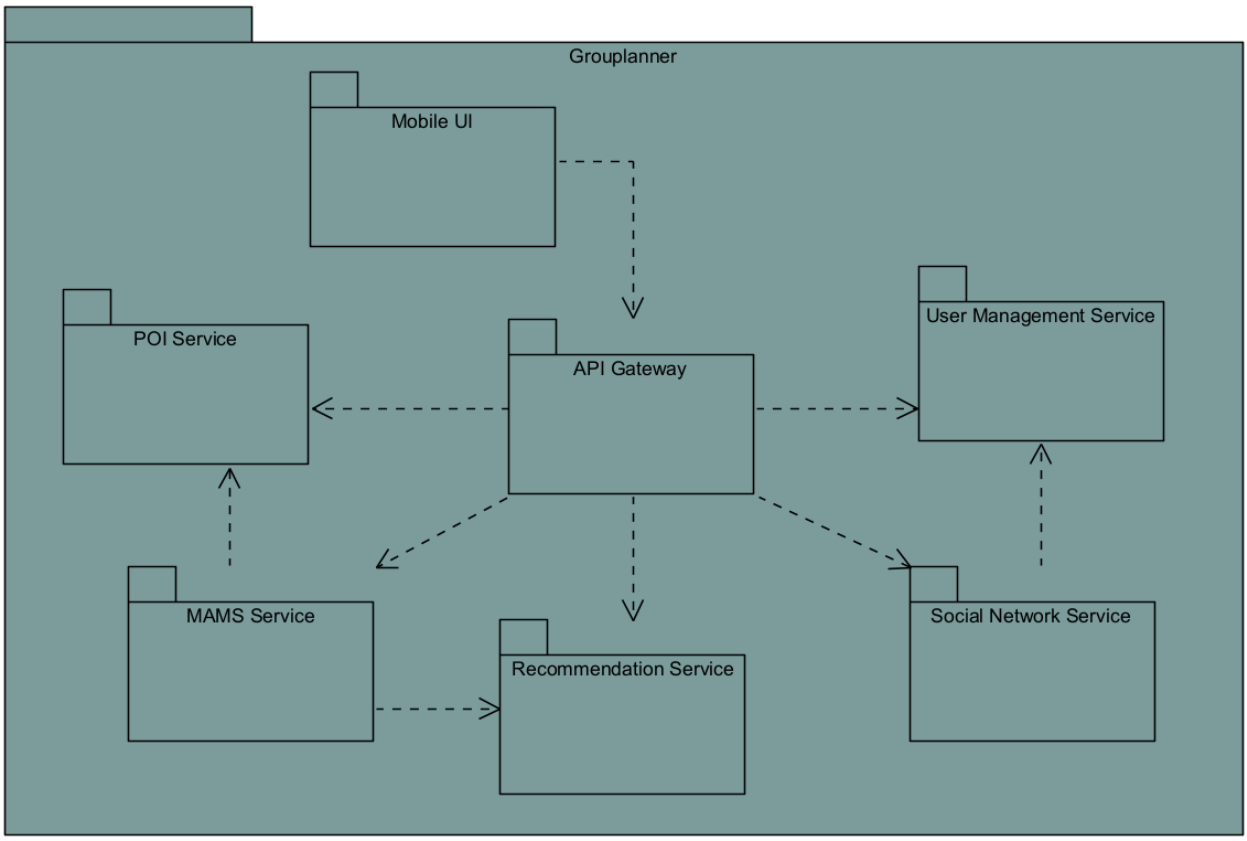

Figure 12 - Level 2 Implementation View of Grouplanner

The Mobile UI depends exclusively on the API Gateway to connect to the backend services. The API Gateway relies on the User Management Service for user operations and links with the

Social Network Service for social chat functionalities. The Recommendation Service makes suggestions via the API Gateway. MAMS Service manages agent-related data, communicating with the POI Service to retrieve POI suggestions and depends on Recommendation Service to send the POI inclusion/exclusion list. Finally, the POI Service gives points of interest to the MAMS Service. Despite all communication rely on the API Gateway, the microservices depend on each other, as illustrated, to provide the overall services of the Grouplanner system.

Physical View

The physical view diagram shown in Figure 13 reveals how the microservices architecture of the Grouplanner system is set up on Azure, highlighting the links and protocols (HTTPS) for communication between the different components.

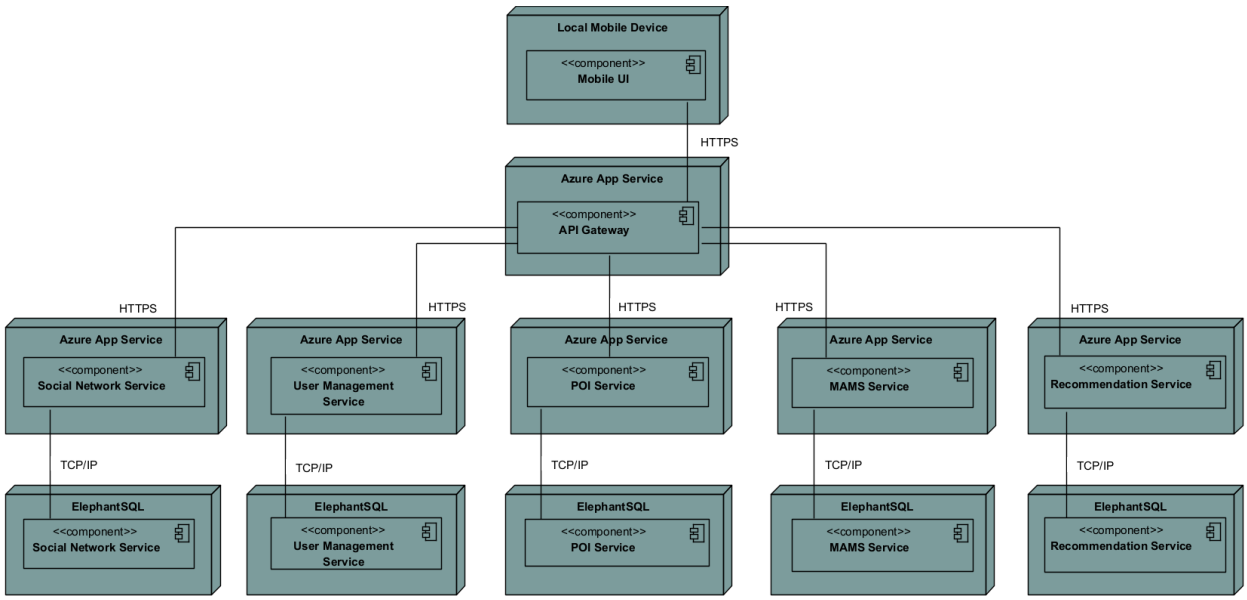

Figure 13 - Level 2 Physical View of Grouplanner

Azure was selected for this deployment because of its support for .NET technologies. Each microservice is connected to its respective database, all hosted in ElephantSQL and communicates with its database via TCP/IP, which ensures reliable data exchange.

3.5.3. Level 3 - Components

The Level 3 views analyze the internal workings of MAMS architecture components, providing an in-depth review of its logic, implementation, and processes. Figure 14 provides an understanding of the internal procedures and dependencies that constitutes the MAMS Service.

Logic View

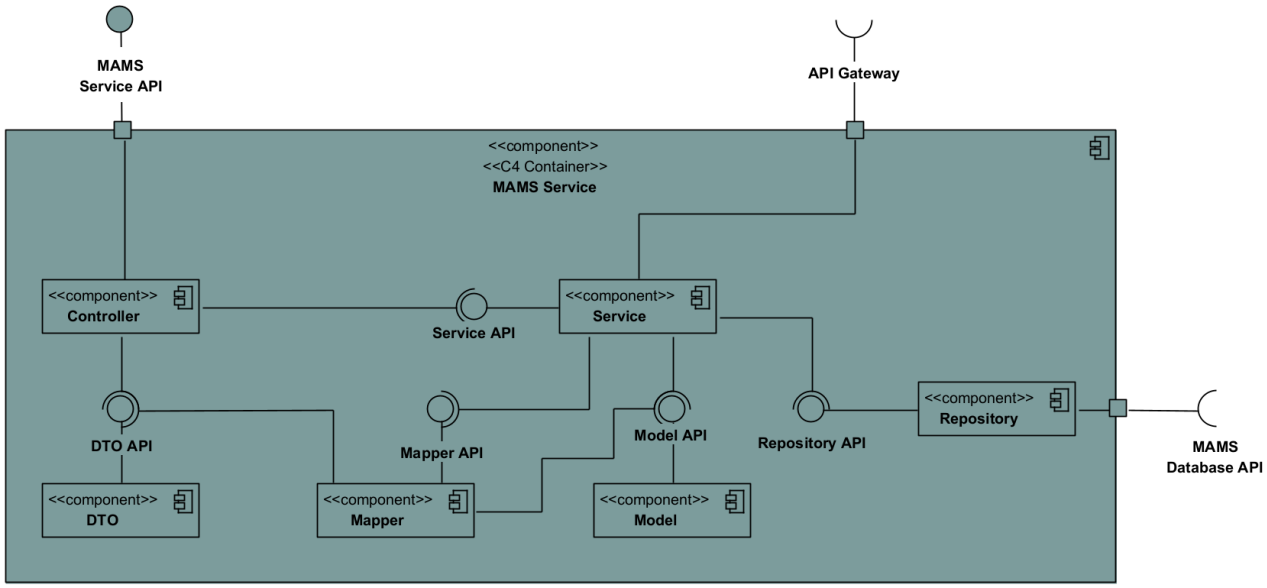

Figure 14 - Level 3 Logic View of MAMS

This level 3 logic view diagram details MAMS's internal structure within the Grouplanner architecture. MAMS's REST API acts as an access point for external requests, linked to the API Gateway.

The **API Gateway** manages and directs external requests, with the ability to consume other specific microservices within the Grouplanner application via the gateway that connects them, described in the logic view in section 3.5.2.

The **Controller** component handles incoming requests from the **MAMS Service API** and works with other internal components to fulfill them: **Service API**, providing an interface for the business logic enclosed within the **Service** component, which processes requests, performs operations, and orchestrates interactions across other MAMS components; **DTO API**, representing

the Data Transfer Objects (DTOs), which used to move data across different layers of an application. Therefore, the **DTO component** serves as an interface for these objects, ensuring that data is consistent and correctly formatted.

Consuming the **DTO API**, the **Mapper component** transforms data across formats or layers, ensuring that DTO data is properly mapped to domain models and vice versa. The **Model API** serves as an interface for domain models, which represent business entities in the MAMS service, and it is consumed by the **Service component** so business logic can be handled with the necessary data manipulation of the **Mapper API**.

The **Repository API** offers an interface for data-management tasks related to the business domain in **Model component**. The **Repository component** manages CRUD (Create, Read, Update, and Delete) and other data persistence operations, while the MAMS Database API connects the Repository component to the database where the data is kept, allowing MAMS to save and access data as needed.

The overall flow begins with the **Controller**, then moves on to the **Service**, which involves the **Repository** and **Mapper** manipulating the **Model** before returning data to the **Controller**.

Process View

Since the association rules with Apriori is the most complex implementation and interaction between components, a more detailed process view was created to guide the implementation process presented in section 4. Figure 15, Figure 16 and Figure 17, respectively, the diagram and the two subdiagrams represent the process view of MAS-US-02.

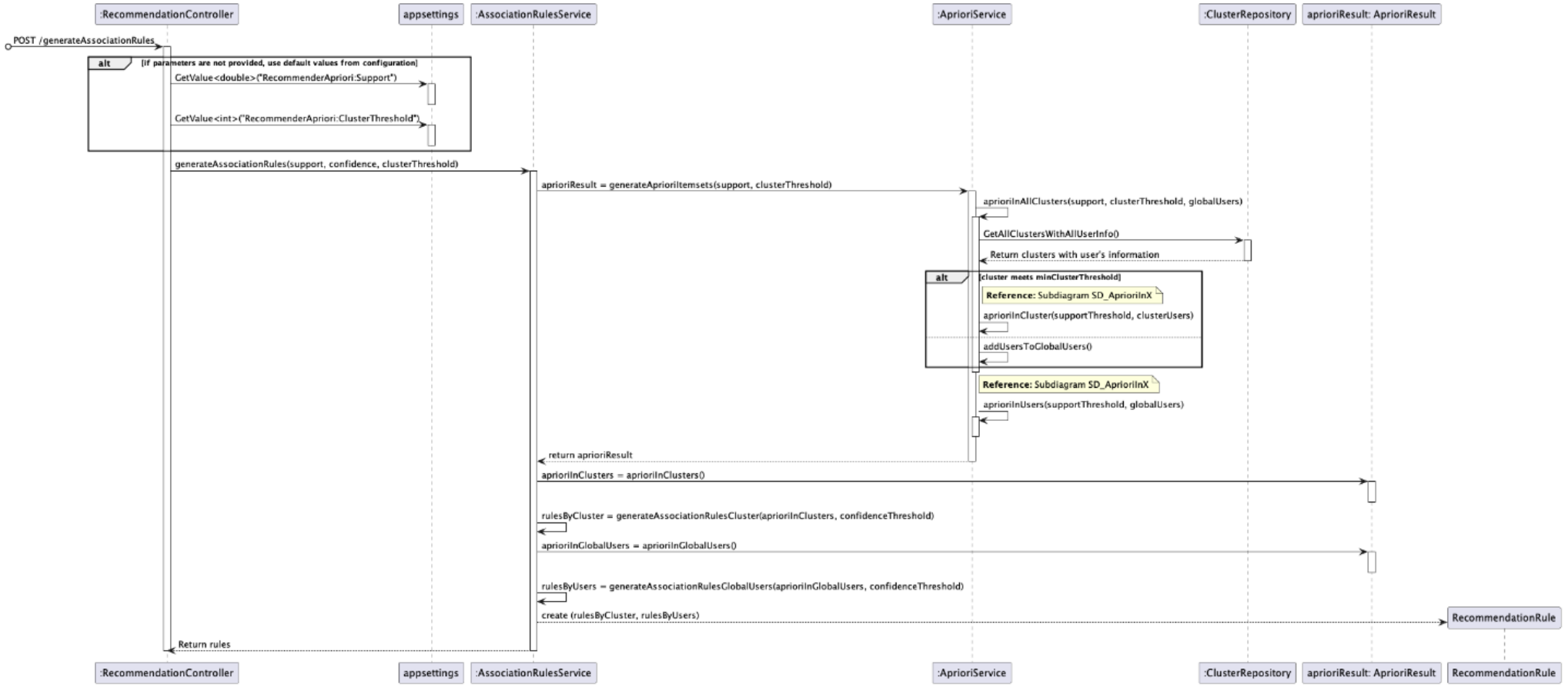

Figure 15 - MAS-US-02 Level 3 Process View – Association Rules

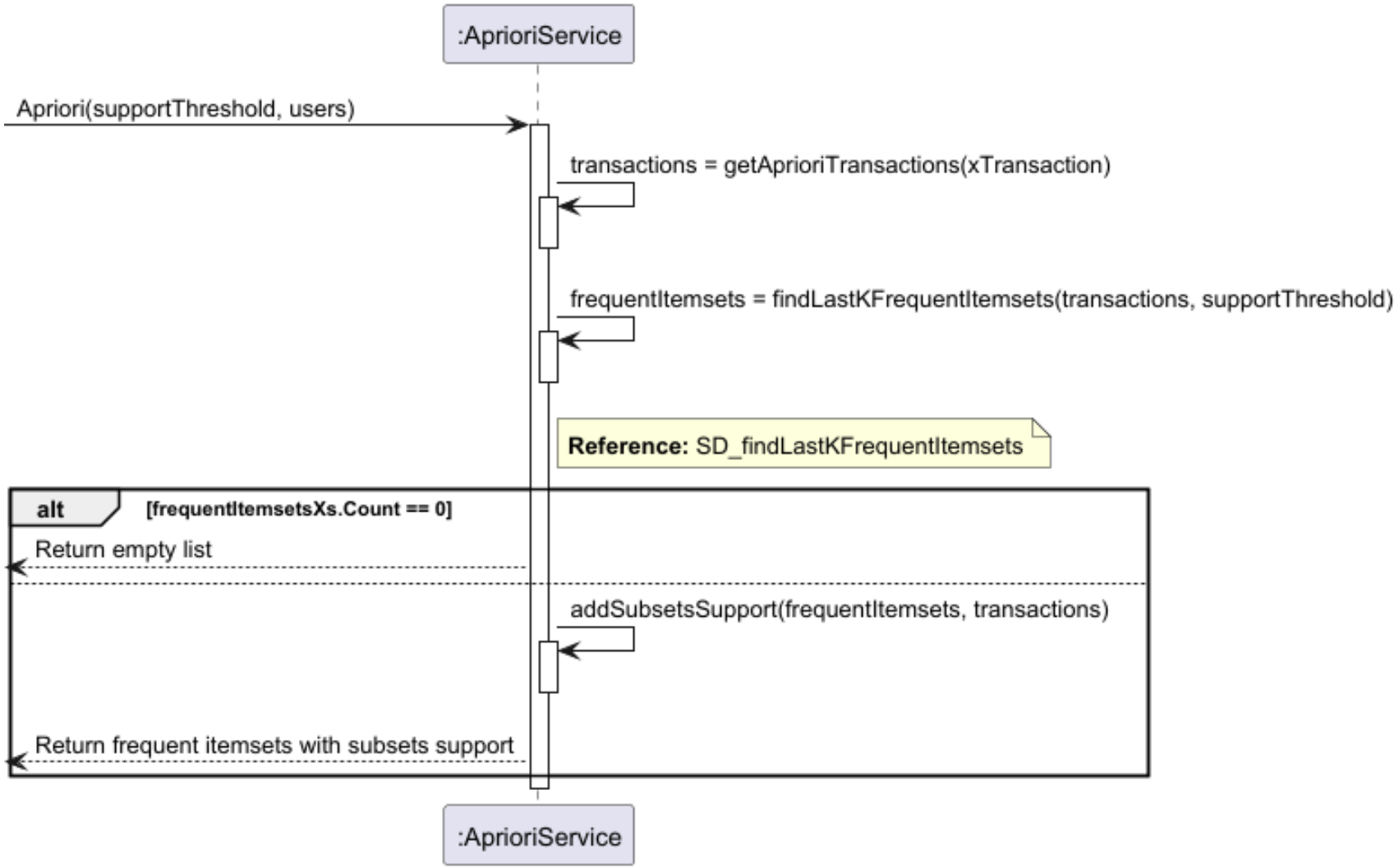

Figure 16 - MAS-US-02 Level 3 Process View – Apriori Algorithm

Figure 17 - MAS-US-02 Level 3 Process View – Candidate Itemsets within Apriori

Figure 15 represents the process of applying association rules for clusters and globally based on the Apriori itemsets discovered, which is illustrated on the subdiagram in Figure 16, that describes the Apriori algorithm's phases, including detecting frequent itemsets. Finally, Figure 17 describes the development of candidate itemsets within Apriori, including how initial itemsets are combined and filtered to identify frequent ones.

Implementation View

As illustrated in Figure 18, the MAMS implementation represents the internal dependencies between components.

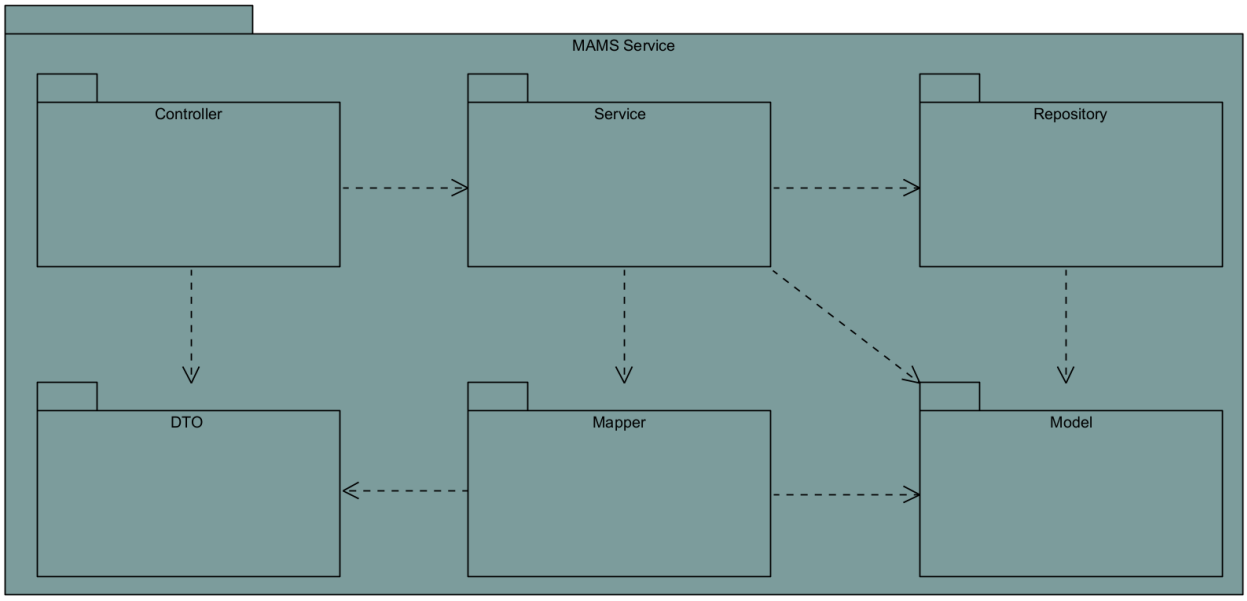

Figure 18 - Level 3 MAMS Implementation View

The **Controller** depends on the **Service** to handle business rules and the independent **DTO component** to maintain the integrity of the data. The **Service** depends on the **Mapper** to convert data between **DTO** and **Model** formats and depends on the **Repository** for persistence matters. Finally, the **Repository** works with the independent **Model component** to make data management operations.

3.5.4. Level 4 – Code

In the code level, a more refined and concrete code-view is illustrated. Therefore, MAS-US-02 is described in the class diagram illustrated in Figure 19, not only showing the main components, but also the details and dependencies of the software system.

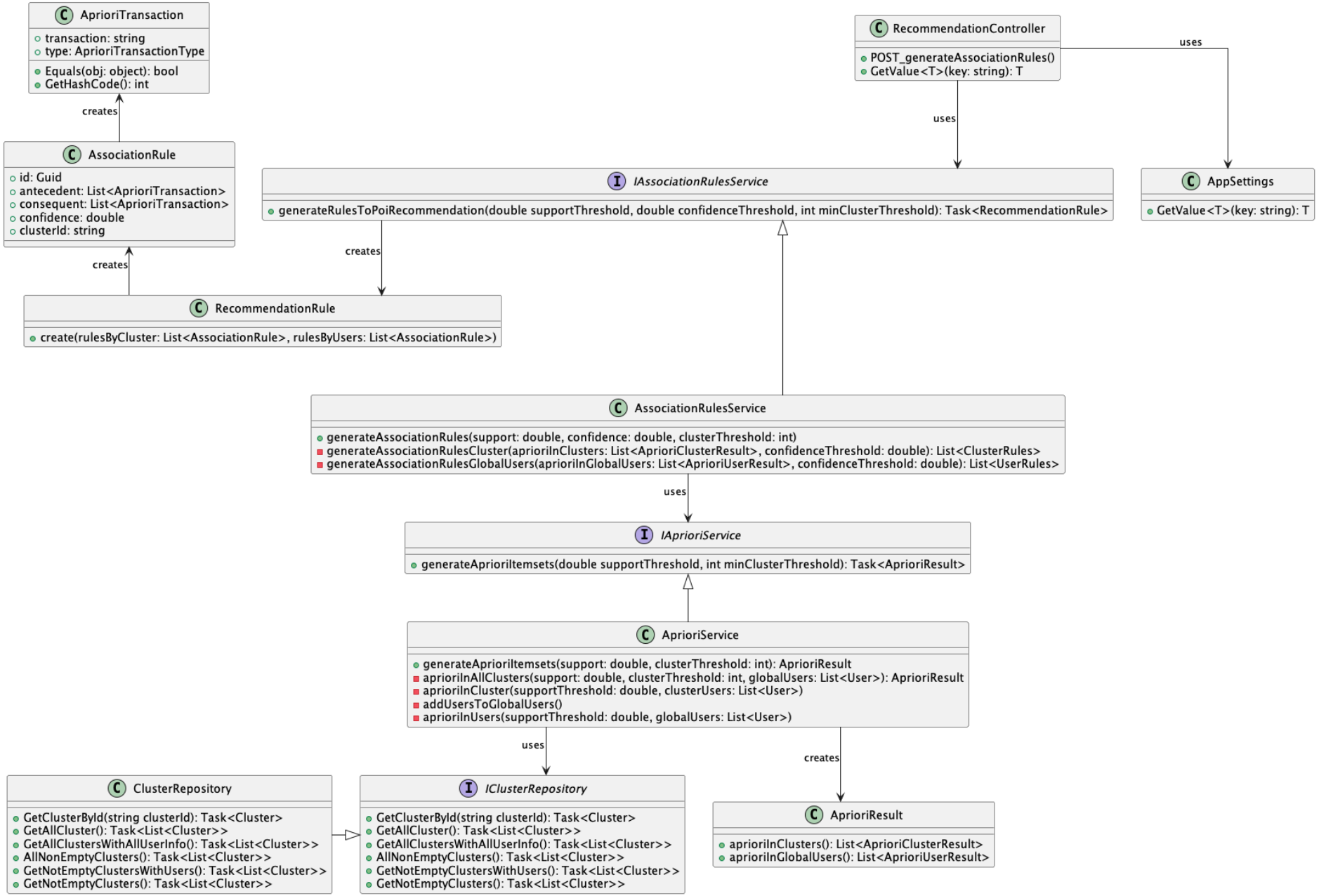

Figure 19 - MAS-US-02 Class Diagram

This diagram demonstrates the architecture of a recommendation system with association rules using Apriori. The AprioriTransaction class stores transactions, which are then used by AssociationRule to produce rules. ClusterRepository contains the users grouped into clusters. The RecommendationController generates suggestions using the AssociationRulesService. AssociationRulesService works with AprioriService to build frequent itemsets, which are then utilized to create the association rules into the RecommendationRule. Finally, the RecommendationRule is saved in the MASService so that the patterns found stay in the agents MASService instance.

4. Implementation Solution

This section provides a complete overview of the solution's implementation, beginning with a description of the MAMS microservice's general functionality only necessary to the context of the new implementations. It contains a thorough examination of the user stories described in section 3.4.1 that were implemented. The section wraps up with a summary of the tests designed to value the system's functionality and reliability.

4.1. Implementation Description

This section describes the implementation of the user stories. Firstly, it shows the multi-agent service current implementation and how it can be used to preserve the new rules of the Apriori algorithm. Then, the focus is on how the system cluster users on their personality metrics, how the Apriori algorithm generates association rules for user characteristics and preferences, and lastly, how these rules are used to create personalized and group-based POI inclusion/exclusion lists.

4.1.1. MAMS Multi-Agent Service Component

This section focusses on the MAMS service component that coordinates context-specific actions between agents. The Multi-Agent Service aims to enable a network of autonomous agents, each representing a user tourist, a group or a subgroup, to function effectively inside the software environment. Its function is to manage and coordinate these agents, allowing them to complete tasks autonomously or collectively. This architecture enables agents to interact, make decisions, and carry out actions depending without blocking the service itself by using a Singleton type of service. This allows the service to be the same for all agents, since they are registered in the same instance of the multi-agent system and is useful to register their respective rules generated by the Apriori algorithm since it can be re-generated.

Code Snippet 1 shows the main aspects of the MASService class. It initializes various components, including a dictionary for agent responses that holds replies from various agents in the system and allows them to communicate. Also, an environment representing the container in which agents conduct their actions, and a timer agent to keep the environment active and prevent

it from terminating. The constructor sets up these components and starts the environment in a separate thread to ensure it runs concurrently with other processes. The ‘StartEnv’ method is responsible for starting the environment on this separate thread, ensuring that it operates without blocking other operations. This environment is run by the ‘envThread’, which ensures that the agents within it are active and capable of carrying out their assigned tasks.

```
//Apriori association rules for POI Inclusion/Exclusion lists
private RecommendationRule aprioriRules { get; set; }

public MASService()
{
    //Initializing the response dictionary dictionary
    agentResponseDictionary = new Dictionary<String,
ConcurrentDictionary<Guid, Message>>();
    //Initializing the environment
    environment = new EnvironmentMas();
    //Adding the timer agent
    //This also ensures that the environment doesn't stop due to lacking
any agents
    environment.Add(new TimerAgent(), TIMER_AGENT_NAME);
    //Creating the thread for the environment
    envThread = new Thread(StartEnv);
    //Starting the environment thread
    envThread.Start(environment);
}

//Method used for starting the environment in a separate thread
private void StartEnv(object env)
{
    Console.WriteLine("Starting environment");
    EnvironmentMas envT = (EnvironmentMas)env;
    envT.Start();
    Console.WriteLine("Environment started");
}
```

Code Snippet 1 - MASService Overview

All components were already implemented, except the new variable ‘aprioriRules’. This variable includes both cluster-specific rules and global rules, where the global rules apply to users who are part of a cluster but do not have specific cluster rules. This variable will start as ‘null’ to distinguish if the rules have been generated or if they are empty and no patterns were found. The Apriori algorithm generates association rules, which are stored in the MAS Service component, allowing the system to make suggestions about agents based on the patterns observed.

4.1.2. MAS-US-01 Cluster Users Based on Personality

The cluster of users' integration involves group and regrouping users based on their personality traits utilizing the Big Five framework. This model measures five major personality dimensions: openness, conscientiousness, extraversion, agreeableness, and neuroticism. By examining these variables, the system creates groups of individuals with similar personality features in clusters. This use case is critical for then developing rules using clusters. This means the patters will be searched by processing users inside each cluster and not only the whole dataset of users in MAMS.

Code Snippet 2 shows the 'Reclustering' method, which cluster users based on personality traits using the Big Five framework.

```
public async Task Reclustering()
{
    //By default no changes Happened
    bool centroidHasChanged;

    //Initial values to centroid
    var allUsers = await userRepo.AllUsers();
    var allClusters = await clusterRepo.AllNonEmptyClusters();
    var allClustersCopy = new List<Cluster>();

    do
    {
        centroidHasChanged = false;
        var allUsersCopy = new List<User>(allUsers);

        for (var i = 0; i < allUsersCopy.Count; i++)
        {
            var u = allUsersCopy[i];
            allClustersCopy = [..allClusters];

            bestCluster = FindBestClusterForUser(allClustersCopy, u);

            centroidHasChanged = UpdateUserClusterAssignment(bestCluster,
u, centroidHasChanged, allUsers, i, allClusters);
        }

        if (centroidHasChanged) continue;

        centroidHasChanged = RecalculateUpdateCentroids(allClustersCopy,
allUsers, centroidHasChanged, allClusters);
    } while (centroidHasChanged);

    var usersToUpdate = allUsers.Where(u => u.hasClusterChanged).ToList();

    if (usersToUpdate.Count != 0)
    {
        await userRepo.UpdateObjects(usersToUpdate);
    }
}
```

Code Snippet 2 - Reclustering Users Functionality

The method is based on the Dynamic personality-based clustering, *d*-means (Alves et al., 2024), and it begins by setting a flag, ‘centroidHasChanged’, to indicate whether any centroids (average points representing each cluster) change during the procedure. It then retrieves all users and non-empty clusters from the corresponding repositories. The procedure enters a loop that lasts as long as the centroids change.

Code Snippet 3 accounts for the ‘FindBestClusterForUser’ inner method, which identifies the most suitable cluster for a given user.

```
private static Cluster FindBestClusterForUser (List<Cluster>
allClustersCopy, User user, double minDistance)
{
    Cluster bestCluster = null;
    var minDistance = double.MinValue;

    foreach (var c in allClustersCopy)
    {
        //Calculate normalized euclideanDistance between the actual user
and the actual cluster centroid
        var euclideanSimilarity = Distance.EuclideanSimilarity(user, c);


        if (!(euclideanSimilarity > minDistance) || !(euclideanSimilarity
>= MIN_DISTANCE)) continue;
        minDistance = euclideanSimilarity;
        bestCluster = c;
    }

    return bestCluster;
}
```

Code Snippet 3 - Best Cluster for a specific User

It iterates over each cluster, calculating the Euclidean similarity between the user and the centroid. The ‘centroid’ represents the average scores of the Big Five personality traits for all users in that cluster. If the estimated similarity exceeds the current minimal distance and reaches a given threshold, it changes both the best cluster and the minimum distance. After assessing all clusters, it returns the cluster that is most similar to the user.

Code Snippet 4 defines the ‘UpdateUserClusterAssignment’ inner method, which updates a user's cluster assignment based on the identified best cluster. The method checks if the best cluster is different from the user's current cluster.

```
private static bool UpdateUserClusterAssignment(Cluster bestCluster, User
u, bool centroidHasChanged, IList<User> allUsers,
    int clusterIndex, ICollection<Cluster> allClusters)
{
    if (bestCluster != null && bestCluster != u.cluster)
    {
        // If the avg is the different, centroid changed, so change
centroidHasChanged to true to continue the loop and update the values of
the cluster
        centroidHasChanged = true;

        // Update existing cluster in the user and update the user itself
        u.cluster = bestCluster;
        u.clusterId = bestCluster.clusterId;
        u.hasClusterChanged = true;

        allUsers[clusterIndex] = u;
    }
    else if (bestCluster == null)
    {
        var newCluster = new Cluster
        {
            centroidUserName = u.UserName,
            clusterId = Guid.NewGuid().ToString(),
            openness = u.openness,
            conscientiousness = u.conscientiousness,
            extraversion = u.extraversion,
            agreeableness = u.agreeableness,
            neuroticism = u.neuroticism,
            users = []
        };

        u.cluster = newCluster;
        u.clusterId = newCluster.clusterId;
        u.hasClusterChanged = true;
        allClusters.Add(newCluster);
        centroidHasChanged = true;
    }

    return centroidHasChanged;
}
```

Code Snippet 4 - User Cluster Assignment

If the best cluster is different from the user's current cluster, it sets the ‘centroidHasChanged’ flag to true, indicating that the cluster centroids need to be updated. It then updates the user's cluster and cluster identifier, marks the user as having changed clusters, and reflects the changes in the allUsers list.

If no suitable cluster is found (‘bestCluster’ is null), the method creates a new cluster with the user's personality traits as its centroid, assigns this new cluster to the user, and adds the new cluster

to the list of all clusters. The ‘centroidHasChanged’ flag is set to true to ensure that the centroids will be recalculated.

Code Snippet 5 describes the ‘RecalculateUpdateCentroids’ inner method, which updates the centroids of clusters based on the latest user assignments.

```
private static bool RecalculateUpdateCentroids(IReadOnlyList<Cluster>
allClustersCopy, IReadOnlyCollection<User> allUsers, IList<Cluster>
allClusters)
{
    var centroidHasChanged = false;
    for (var i = 0; i < allClustersCopy.Count; i++)
    {
        var cluster = allClustersCopy[i];
        // Recalculate cluster centroid
        var usersInCluster = allUsers.Where(user => user.clusterId ==
cluster.clusterId).ToList();

        // Only users with cluster assigned
        if (usersInCluster.Count == 0)
            continue;

        // Average each users traits
        var centroid = CalculateCentroid(usersInCluster);

        // Update cluster centroid if it changes
        if (AreEqual(cluster, centroid)) continue;

        cluster.openness = centroid.Openness;
        cluster.conscientiousness = centroid.Conscientiousness;
        cluster.extraversion = centroid.Extraversion;
        cluster.agreeableness = centroid.Agreeableness;
        cluster.neuroticism = centroid.Neuroticism;
        centroidHasChanged = true;

        // Update existing cluster in the allClusters list
        allClusters[i] = cluster;
    }

    return centroidHasChanged;
```

Code Snippet 5 - Cluster Centroids Recalculation

After all users have been processed, if no centroids have changed, the procedure recalculates the centroids of each cluster. It achieves this by averaging the personality metrics of all users in each cluster. If the recalculated centroid varies from the existing one, the cluster is updated with the new centroid values, and ‘centroidHasChanged’ is set to true, resulting in another loop iteration. This flag indicates that changes have occurred, which means further adjustments to the

clusters may be necessary. The updated clusters are then reflected in the ‘allClusters’ list. This method ensures that cluster centroids accurately represent the average traits of their members.

Finally, after all clusters are stable, the approach identifies users whose cluster assignments have changed and does the changes only on their records in the repository, illustrated back in the Code Snippet 1, dynamically refining the users and the clusters based on personality metrics.

4.1.3. MAS-US-02 Request Association Rules

The "Generate Association Rules" use case involves creating association rules to detect relevant patterns and relationships in user data. Using the Apriori algorithm, the system analyzes user characteristics and preferences to build frequent itemsets based on a percentage of specified support. These itemsets are then used to generate association rules based on given confidence criteria, which are also expressed in percentage.

The rules created are based on the 'minClusterThreshold'. If this minimal value is equal to or less than the number of users in a cluster, the rules will be written for that cluster individually, while the remaining users will have rules generated as a whole group.

Code Snippet 6 defines the ‘GenerateAssociationRules’ controller method, which is responsible for generating association rules for POI recommendations based on configurable parameters. This method was designed for use by the System Administrator to request the generation of rules.

```
[HttpPost("generateAssociationRules")]
public async Task<IActionResult> GenerateAssociationRules(double? support =
null, double? confidence = null, int? minClusterThreshold = null)
{
    try
    {
        // Check if parameters are provided, if not, use default values
from configuration
        support ??=
configuration.GetValue<double>("RecommenderApriori:Support");
        confidence ??=
configuration.GetValue<double>("RecommenderApriori:Confidence");
        minClusterThreshold ??=
configuration.GetValue<int>("RecommenderApriori:ClusterThreshold");

        var result = await
associationRulesService.GenerateRulesToPoiRecommendation(support.Value,
confidence.Value, minClusterThreshold.Value);

        return Ok(result);
    }
    catch (Exception x)
    {
        return StatusCode(500, x.Message);
    }
}
```

Code Snippet 6 - Rules Generation Controller

The 'GenerateAssociationRules' serves as an HTTP POST method for the System Administrator to request the generation of association rules. It provides optional parameters for support, confidence, and minimum cluster threshold, which default to values from the standard configuration file 'appsettings' if not specified.

In case of the system itself, Code Snippet 7 represents the automated process of generating rules.

```
private void StartAprioriTimerTask()
{
    // Read configuration settings
    var hoursInterval =
configuration.GetValue<int>("RecommenderApriori:HoursInterval");
    var supportThreshold =
configuration.GetValue<double>("RecommenderApriori:Support");
    var confidenceThreshold =
configuration.GetValue<double>("RecommenderApriori:Confidence");
    var clusterThreshold =
configuration.GetValue<int>("RecommenderApriori:ClusterThreshold");

    // Convert hours to TimeSpan for the timer
    var interval = TimeSpan.FromHours(hoursInterval);

    // Set up the timer
    aprioriTimer = new Timer( _ =>
    {
    // Call the generateRulesToPoiRecommendation method
AssociationRulesService.GenerateRulesToPoiRecommendation(supportThreshold,
confidenceThreshold, clusterThreshold);
    }, null, TimeSpan.Zero, interval);
}
```

Code Snippet 7 - Automated Apriori Timer

The rules are automatically generated in MAS Service Component, more detailed in section 4.1.1. The ‘StartAprioriTimerTask’ method sets up a timer to call the ‘GenerateRulesToPoiRecommendation’ method of the ‘AssociationRulesService’ at a regular interval specified in the configuration settings. It first reads the interval, support threshold, confidence threshold, and cluster threshold values from the configuration.

Then, it creates rules using these arguments by invoking the 'AssociationRulesService' service method. If successful, it returns an ‘OK’ status. If an exception occurs, the status code ‘500’ is returned, along with the exception message.

Code Snippet 8 defines the ‘GenerateRules’ method, which is responsible for generating association rules using the Apriori algorithm.

```
public async Task<RecommendationRule> GenerateRules(double
supportThreshold, double confidenceThreshold, int minClusterThreshold)
{
    // Generate frequent itemsets using the Apriori algorithm
    var aprioriResult = await
aprioriService.GenerateAprioriItemsets(supportThreshold,
minClusterThreshold);

    // Generate association rules based on the frequent itemsets
    var rules = new RecommendationRule
    {
        rulesByCluster =
GenerateAssociationRulesCluster(aprioriResult.aprioriClusterItemsets,
confidenceThreshold),
        rulesByUsers = GenerateAssociationRulesGlobalUsers
(aprioriResult.aprioriGlobalUsersItemsets, confidenceThreshold)
    };

    // Save the generated rules in MAS
    multiAgentService.SaveAprioriRules(rules);

    return rules;
}
```

Code Snippet 8 - Generation of Rules

Inside the method in 'AssociationRulesService' service, it creates inclusion and/or exclusion rules for Points of Interest (POI) to include or exclude in the recommendations. It begins by constructing frequent itemsets using the Apriori method, which is dependent on given support and cluster thresholds. It then creates a ‘RecommendationRule’ object, which includes association rules categorized by clusters and global users. After generating these rules, the method saves them within the multi-agent the generated rules.

Code Snippet 9 defines the ‘GenerateAprioriItemsets’ method, which generates frequent itemsets using the Apriori algorithm. It initializes a list for global users and then calculates two sets of itemsets built on the predefined support threshold: one for all clusters based on the stipulated minimum cluster threshold, and one for global users.

```
public async Task<AprioriResult> GenerateAprioriItemsets(double
supportThreshold, int minClusterThreshold)
{
    var globalUsers = new List<User>();

    return new AprioriResult()
    {
        aprioriClusterItemsets = await AprioriInAllClusters(supportThresh,
minClusterThreshold, globalUsers),
        aprioriGlobalUsersItemsets = AprioriInUsers(supportThreshold,
globalUsers)
    };
}
```

Code Snippet 9 - Generation of Frequent Itemsets

The method begins by defining an empty list, ‘globalUsers', that contains users whose clusters do not satisfy the minimum cluster threshold. It then generates the Apriori itemsets using two helper methods: 'AprioriInAllCluster' and 'AprioriInUsers'.

Code Snippet 10 defines the ‘AprioriInAllClusters’ helper method, which obtains all clusters, including user information, and then processes each cluster independently. Clusters with user counts that comply with the minimal criteria are processed and generate cluster-specific itemsets. If the cluster fails to achieve the threshold, its users are added to a list for global item set creation.

```
private async Task<List<AprioriClusterItemset>> AprioriInAllClusters(double
supportThreshold, double minClusterThreshold, List<User> globalUsers)
{
    var aprioriResultsCluster = new List<AprioriClusterItemset>();
    var clusters = await clusterRepository.GetAllClustersWithAllUserInfo();

    foreach (var cluster in clusters)
    {
        // Only consider clusters that meet the minimum threshold
        if (cluster.users.Count >= minClusterThreshold)
        {
            var aprioriClusterItemsets = AprioriInCluster(supportThreshold,
cluster);
            if(aprioriClusterItemsets!= null)
                aprioriResultsCluster.Add(aprioriClusterItemsets);
        }
        else {
            // List of users below the cluster threshold
            globalUsers.AddRange(cluster.users);}
    }
    return aprioriResultsCluster;
}
```

Code Snippet 10 – Separation of Users by Cluster or Globally

Code Snippet 11 outlines the ‘AprioriInCluster’ method, which employs the Apriori algorithm to calculate frequent itemsets for a specific cluster. The method begins by creating transactions from users in the specified cluster. It then extracts frequent itemsets from these transactions based on the support threshold. If no frequent itemsets can be found, it returns ‘null’. Otherwise, it combines the itemsets with subset support information and produces an 'AprioriClusterItemset' object containing the cluster identifier and the respective frequent itemsets.

```
private static AprioriClusterItemset AprioriInCluster(double
supportThreshold, Cluster cluster)
{
    var transactions = GetAprioriTransactions(cluster.users);
    var frequentItemsetsCluster = FindLastKFrequentItemsets(transactions,
supportThreshold);

    if (frequentItemsetsCluster.Count == 0) return null;

    var frequentItemsetsWithSubsetsCluster =
AddSubsetsSupport(frequentItemsetsCluster, transactions);

    return new AprioriClusterItemset { ClusterId = cluster.clusterId,
FrequentItemsets = frequentItemsetsWithSubsetsCluster };
}
```

Code Snippet 11 - Apriori Application in a specific Cluster

Similar to ’AprioriInCluster’, the ‘AprioriInUsers’ method generates frequent itemsets for users who did not meet the cluster threshold.

Code Snippet 12 defines the ‘AprioriInUsers’ method, which calculates frequent itemsets for a set of users using the Apriori algorithm. The method generates transactions from the given users and then identifies frequent itemsets based on the provided support threshold. If no frequent itemsets are identified, it returns an empty list. Otherwise, it adds to these itemsets with subset support information and returns the list of frequent itemsets.

```
private static List<FrequentItemset> AprioriInUsers(double
supportThreshold, IEnumerable<User> usersMinThreshold)
{
    var transactionsUsers = GetAprioriTransactions(usersMinThreshold);
    var frequentItemsetsUsers =
FindLastKFrequentItemsets(transactionsUsers, supportThreshold);

    return frequentItemsetsUsers.Count == 0 ? [] :
AddSubsetsSupport(frequentItemsetsUsers, transactionsUsers);
}
```

Code Snippet 12 - Apriori Application in Global Users

From now on, the methods and inner functions of 'GetAprioriTransactions', 'FindLastKFrequentItemsets' and 'AddSubsetsSupport' are used in generating frequent itemsets for both of global users and by users of a cluster.

Code Snippet 13 defines the 'GetAprioriTransactions' method, which is simplified for better understanding. It creates transaction lists for users to be used in the Apriori algorithm. For each user, it maps various attributes (such as age, gender, civil state, and other personal details) and attraction type preferences to 'AprioriTransaction' objects. This structure, through 'enum' values, helps keep track the type of transaction is being created, either if is about the user gender with type '0', a liked attraction type preference of type '9', or other. If there is any transaction about a user, it is added to the main list to be carried out for the generation of the frequent itemsets.

```
private static List<List<AprioriTransaction>> GetAprioriTransactions
(IEnumerable<User> users)
{
    var transactions = new List<List<AprioriTransaction>>();

    foreach (var user in users)
    {
      var transaction = new List<AprioriTransaction>();

      // Mapping user attributes to Apriori transactions
      AddIfNotNull(transaction, UserAgeBinMapper.mapAgeToCategory(
user.birthday));
      AddIfNotNull(transaction, user.gender, 0);
      AddIfNotNull(transaction, user.civil_state, 1);
      AddIfNotNull(transaction, user.hasChildren, 2);
      AddIfNotNull(transaction, user.educationLevel, 3);
      AddIfNotNull(transaction, user.formation_area, 4);
      AddIfNotNull(transaction, user.professional_situation, 5);
      AddIfNotNull(transaction, user.live_with, 6);
      AddIfNotNull(transaction, user.income, 7);
      AddIfNotNull(transaction, user.religion, 8);

      // Adding attraction types, limitations, and fears if not null
      AddRangeIfNotNull(transaction, user.likedAttractionTypes, 9);
      AddRangeIfNotNull(transaction, user.dislikedAttractionTypes, 10);
      AddRangeIfNotNull(transaction, user.limitations, 11);
      AddRangeIfNotNull(transaction, user.fears, 12);

      if (transaction.Any())
        transactions.Add(transaction);
    }

    return transactions;
}
```

Code Snippet 13 - Convertion of User traits to Transactions

Code Snippet 14 illustrates the 'FindLastKFrequentItemsets' method, which seeks for the most frequent itemsets within a given collection of transactions using a predefined support threshold. First, it extracts unique 'AprioriTransaction' items from all transactions and creates a support count dictionary to track the occurrences of each itemset. It then enters a loop in which ‘k’ is incrementally increased to locate larger itemsets until no more can be found. This means that the function breaks out of the loop and returns the last ‘k’ size frequent itemsets.

```
private static List<List<AprioriTransaction>>
FindLastKFrequentItemsets(IList<List<AprioriTransaction>> transactions,
double supportThreshold)
{
    var frequentItemsets = new List<List<AprioriTransaction>>();
    var k = 1;

    var items = transactions.SelectMany(t => t)
        .GroupBy(t => t.GetHashCode())
        .Select(group => group.First())
        .ToList();

    var totalTransactions = transactions.Count;

    while (true)
    {
        var candidateItemsets = GetSubsetsOfSizeK(items, k);

        var uniqueItems = new HashSet<AprioriTransaction>(new
AprioriTransactionEqualityComparer());

        var frequentSets = FilterCandidateItemsets(candidateItemsets,
transactions, supportThreshold, totalTransactions, uniqueItems);

        if (frequentSets.Count == 0)
            break;

        items = [..uniqueItems];

        frequentItemsets = frequentSets;
        k++;
    }

    return frequentItemsets;
}
```

Code Snippet 14 - Last K Frequent Itemsets Application

Code Snippet 15 shows the ‘GetSubsetsOfSizeK’ method that generates subsets of a given size ‘k’ from a list of transactions. It uses a queue-based approach for the breadth-first discovery of potential subsets.

```
private static List<List<AprioriTransaction>> GetSubsetsOfSizeK
(List<AprioriTransaction> items, int k)
{
  var subsets = new List<List<AprioriTransaction>>();
  var queue = new Queue<List<AprioriTransaction>>();

  // Initialize the queue with individual items
  foreach (var item in items)
  {
    queue.Enqueue([item]);
  }

    // Generate subsets of size k
  while (queue.Count > 0)
  {
      var subset = queue.Dequeue();
      if (subset.Count == k)
      {
        subsets.Add(subset);
      }
      else
      {
        // Extend the subset with each remaining item and enqueue the
extended subsets
        var lastIndex = items.IndexOf(subset.Last());

        for (var i = lastIndex + 1; i < items.Count; i++)
        {
          var newSubset = new List<AprioriTransaction>(subset){items[i]};
           queue.Enqueue(newSubset);
        }
      }
   }

   return subsets;
}
```

Code Snippet 15 - Subsets of K Size Application

The method initializes a queue with single-item subsets. It then processes each subset in the queue, extending it by adding remaining items and creating new subsets of size k. The newly formed subsets are added back to the queue for further processing. Once a subset reaches the desired size, it is added to the list of subsets. The method continues this process until all possible subsets of size k have been generated and returns the list of these subsets.

Code Snippet 16 shows the ‘FilterCandidateItemsets’ method that analyzes lists of candidate itemsets to transaction data to determine which have sufficient support based on the predefined threshold. The method iterates over each candidate itemset and counts its occurrences in the transactions. It calculates the support of each itemset by dividing its count by the total number of

transactions. Itemsets that meet or exceed the support threshold are added to the list of frequent itemsets.

```
private static List<List<AprioriTransaction>>
FilterCandidateItemsets(List<List<AprioriTransaction>> candidateItemsets,
    IList<List<AprioriTransaction>> transactions, double supportThreshold,
int totalTransactions, HashSet<AprioriTransaction> uniqueItems)
{
    var frequentItemsets = new List<List<AprioriTransaction>>();

    // Evaluate each candidate itemset
    foreach (var itemset in candidateItemsets)
    {
        var count = transactions.Count(t => itemset.All(t.Contains));

        var support = (double)count / totalTransactions;

        // If support meets the threshold, add to frequent itemsets
        if (!(support >= supportThreshold)) continue;

        foreach (var item in itemset)
        {
            uniqueItems.Add(item);
        }
        frequentItemsets.Add(itemset);
    }

    return frequentItemsets;
}
```

Code Snippet 16 - Candidate Itemsets Filtering

After finding all frequent itemsets, both in ‘AprioriInCluster’ and ‘AprioriInUsers’, the ‘AddSubsetsSupport’ method takes a list of frequently used itemsets and calculates their support as well as the support of their subsets using the transactions. It iterates through each itemset, calculating its support using the given function ‘CalculateSupport’, which divides the itemset's support by the total number of transactions, and then determining the support for each of its subsets. Code Snippet 17 represents the ‘AddSubsetsSupport’ method.

```
private static List<FrequentItemset> AddSubsetsSupport
(List<List<AprioriTransaction>> frequentItemsets,
IReadOnlyCollection<List<AprioriTransaction>> transactions)
{
    var frequentItemsetsWithSubsets = new List<FrequentItemset>();

    // Add support information for each frequent itemset
    foreach (var itemset in frequentItemsets)
    {
        var itemsetInfo = new FrequentItemset
        {
            Itemset = itemset,
            Support = CalculateSupport(itemset, transactions),
            SubsetsSupport = new Dictionary<List<AprioriTransaction>,
double>()
        };

        var subsets = GetSubsets(itemset);
        foreach (var subset in subsets)
        {
            itemsetInfo.SubsetsSupport[subset] = CalculateSupport(subset,
transactions);
        }

        frequentItemsetsWithSubsets.Add(itemsetInfo);
    }

    return frequentItemsetsWithSubsets;
}
```

Code Snippet 17 - Subbests Support Calculation

This is the end of the Apriori algorithm implementation called on the 'AssociationRuleService' service when generating the POI inclusion/exclusion rules, described in Code Snippet 8.

It then constructs association rules from these itemsets, categorizing them by clusters and global users. Code Snippet 18 shows the creation of the association rules of each cluster.

```
private static List<AssociationRule>
GenerateAssociationRulesCluster(List<AprioriClusterItemset> aprioriResults,
double confidenceThreshold)
{
    var associationRules = new List<AssociationRule>();

    // Iterate through each cluster's frequent itemsets
    foreach (var aprioriResult in aprioriResults)
    {
        foreach (var itemsetInfo in aprioriResult.FrequentItemsets)
        {
            var itemset = itemsetInfo.Itemset;
            var subsets = itemsetInfo.SubsetsSupport;

            // Generate rules by iterating through subsets of the itemset
            foreach (var (antecedent, supportAntecedent) in subsets)
            {
                var consequent = itemset.Except(antecedent).ToList();

                var supportItemset = itemsetInfo.Support;

                var confidence = supportItemset / supportAntecedent;

                // Add the rule if it meets the confidence threshold
                if (confidence >= confidenceThreshold)
                {
                    associationRules.Add(new AssociationRule
                    {
                        clusterId = aprioriResult.ClusterId,
                        antecedent = antecedent,
                        consequent = consequent,
                        confidence = confidence
                    });
                }
            }
        }
    }

    return associationRules;
}
```

Code Snippet 18 – Association Rules Generation by Cluster

The 'GenerateAssociationRulesCluster' method generates association rules for clusters by processing the frequent itemsets produced by the Apriori algorithm. It iterates through each cluster's frequent itemsets, examining their subsets to generate suitable rules. It calculates the confidence of the rule for each subset, which is defined as the total support of the itemset divided by the support of the antecedent. If the confidence level matches or exceeds the stated threshold, the rule is added to the list of association rules, along with the associated cluster identifier, antecedent, consequence, and confidence.

Code Snippet 19 shows the creation of the association rules of global users.

```
private static List<AssociationRule>
GenerateAssociationRulesGlobalUsers(List<FrequentItemset> frequentItemsets,
double confidenceThreshold)
{
    var associationRules = new List<AssociationRule>();

    // Iterate through each frequent itemset
    foreach (var aprioriResult in frequentItemsets)
    {
        var itemset = aprioriResult.Itemset;
        var subsets = aprioriResult.SubsetsSupport;

        // Generate rules by iterating through subsets of the itemset
        foreach (var (antecedent, supportAntecedent) in subsets)
        {
            var consequent = itemset.Except(antecedent).ToList();

            var supportItemset = aprioriResult.Support;

            var confidence = supportItemset / supportAntecedent;

            // Add the rule if it meets the confidence threshold
            if (confidence >= confidenceThreshold)
            {
                associationRules.Add(new AssociationRule
                {
                    antecedent = antecedent,
                    consequent = consequent,
                    confidence = confidence
                });
            }
        }
    }

    return associationRules;
}
```

Code Snippet 19 - Association Rules by Global Users

The 'GenerateAssociationRules' private method serves a similar purpose, but for globally frequent itemsets rather than clusters. This strategy does not correlate rules with specific clusters, instead focuses on overall patterns. It iterates through each frequent itemset, considers its subsets to generate potential rules, and calculates their confidence. If the confidence level reaches or exceeds the stated threshold, the rule is added to the list of association rules alongside the antecedent, consequent, and confidence.

Finally, the rules are saved in the multi-agent system (MAS), and it returns the created rules as shown in Code Snippet 8. This is the starting point for then to provide context-specific inclusion/exclusion POI on MAS-US-03 and MAS-US-04.

4.1.4. MAS-US-03 Request POI Inclusion/Exclusion list of an Individual

This feature retrieves customized points of interest (POI) for a given user based on the multi-agent system's rules. This includes the POI to include in case a rule favors certain attraction type or to exclude in case a pattern is found to exclude POI of an attraction type. Code Snippet 20 illustrates the 'PoiInclusionExclusionByUser' method, which its goal is to find the POI to Include or Exclude based on the rules relative to an individual.

```
public async Task<POISuggestionsDto> PoiInclusionExclusionByUser(string
username)
{
    var user = await userRepository.GetUserByUsername(username);

    // Check if the user was found
    if (user == null)
        throw new UserNotFoundException();

    // Retrieve all association rules persisted in MAS
    var allRules = masService.AprioriAssociationRules();

    // Check if association rules were already generated
    if (allRules == null)
        throw new RulesNotGeneratedException();

    // If no rules were found, returns empty POI suggestions
    if (allRules.rulesByUsers.Count == 0 && allRules.rulesByCluster.Count ==
0)
        return new POISuggestionsDto();

    // First find rules of the user cluster - this rules are prioritized
above global rules
    var rulesByCluster = allRules.rulesByCluster.Where(c => c.clusterId ==
user.clusterId).ToList();

    // If no cluster rules were found, global rules will be addressed
instead
    var rules = rulesByCluster.Count != 0 ? rulesByCluster :
allRules.rulesByUsers.ToList();

    // If no rules were found, returns empty POI suggestions
    if (rules.Count == 0)
        return new POISuggestionsDto();

    var (likedAttractionTypes, dislikedAttractionTypes) =
ExtractIndividualAttractionPreferencesFromMatchingRules(rules, user);

    // If no attraction types were added, returns empty POI suggestions
    if (likedAttractionTypes.Count == 0 && dislikedAttractionTypes.Count ==
0)
        return new POISuggestionsDto();

    //Request POIMS all POI of the respective attraction types
    return await getPOIByAttractionTypeOnPOIervice(likedAttractionTypes,
dislikedAttractionTypes);
}
```

Code Snippet 20 - POI Inclusion/Exclusion List of Groups Application

Initially, it retrieves the user's data from the 'userRepository'. If the user cannot be found, it throws the 'UserNotFoundException'. Then it retrieves all association rules stored in MAS. If no association rules are found, indicating that they were not generated, it throws the

'RulesNotGeneratedException' exception. The approach then prioritizes rules relevant to the user's cluster. If no cluster-specific rules are detected, the system defaults to global user rules. It then requests POI data to POIMS based on the liked and disliked attraction and returns the POI to include and/or exclude from the recommendation.

Code Snippet 21 illustrates the 'ExtractIndividualAttractionPreferencesFromMatchingRules' inner method. It retrieves attraction types from the rule's consequence for each rule that matches the user's characteristics and preferences. If the rules do not provide any suitable attraction types, it produces no list of POI to include or exclude.

```
private static (HashSet<int> likedAttractionTypes, HashSet<int>
dislikedAttractionTypes)
ExtractIndividualAttractionPreferencesFromMatchingRules(List<AssociationRul
e> rules, User user)
{
    var likedAttractionTypes = new HashSet<int>();
    var dislikedAttractionTypes = new HashSet<int>();

    /* For each rule, if the rule matches with the attributes of the user,
it means that the user follows that rule
    Add both liked attraction types and disliked attraction types to the
respective lists that the rule dictates */
    foreach (var rule in rules.Where(rule => matchesTransactions
(rule.antecedent, user)))
    {
        //The consequent may not have transactions of type
"LIKES_ATTRACTION_TYPE" or "DISLIKES_ATTRACTION_TYPE"
        //Filter the type of the transaction type to only add attraction
type preferences
        likedAttractionTypes.UnionWith(rule.consequent
            .Where(at => at.type is
AprioriTransactionType.LIKES_ATTRACTION_TYPE)
            .Select(at => int.Parse(at.transaction)));

        dislikedAttractionTypes.UnionWith(rule.consequent
            .Where(at => at.type is
AprioriTransactionType.DISLIKES_ATTRACTION_TYPE)
            .Select(at => int.Parse(at.transaction)));
    }

    return (likedAttractionTypes, dislikedAttractionTypes);
}
```

Code Snippet 21 - Individual Matching Preferences based on Rules

4.1.5. MAS-US-04 Request POI Inclusion/Exclusion list of a Group

This use case focuses on collecting the inclusion or exclusion for points of interest (POI) following Apriori rules for a group of users. The functionality guarantees that includes or exclude POI based on the cluster that originated the group. In case of no cluster identifier provided, the cluster selected will be of the first user in the group.

Code Snippet 22 illustrates the 'PoiGroupInclusionExclusion' method, which its goal is to find the POI to Include or Exclude based on the rules relative to a group and send to it REMS.

```
public async Task<POISuggestionsDto> PoiGroupInclusionExclusionList
(List<string> usernames, string clusterId = null)
{
    var users = await userRepository.GetUsersByUsername(usernames);

    // Check if the users were found
    if (users.Count == 0) throw new UserNotFoundException();

    // Retrieve all association rules persisted in MAS
    var allRules = masService.aprioriAssociationRules();

    // Check if association rules were already generated
    if (allRules == null) throw new RulesNotGeneratedException();

    if (allRules.rulesByUsers.Count == 0 && allRules.rulesByCluster.Count ==
0)
        return new POISuggestionsDto();

    // First find rules of the first user cluster (in case of no cluster ID
provided) - this rules are prioritized above global rules
    var rulesByCluster = clusterId == null
        ? allRules.rulesByCluster.Where(c => c.clusterId ==
users[0].clusterId).ToList()
        : allRules.rulesByCluster.Where(c => c.clusterId ==
clusterId).ToList();

    // If no cluster rules were found, global rules will be addressed instead
    var rules = rulesByCluster.Count != 0 ? rulesByCluster :
allRules.rulesByUsers.ToList();

    // If no rules were found, returns empty POI suggestions
    if (rules.Count == 0) return new POISuggestionsDto();

    var (likedAttractionTypes, dislikedAttractionTypes) =
ExtractGroupAttractionPreferencesFromMatchingRules(rules, users);

    // If no attraction types were added, returns empty POI suggestions
    if (likedAttractionTypes.Count == 0 && dislikedAttractionTypes.Count ==
0)
        return new POISuggestionsDto();

    //Request POIMS all POI of the respective attraction types
    return await getPOIByAttractionTypeOnPOIervice(likedAttractionTypes,
dislikedAttractionTypes);
}
```

Code Snippet 22 - Group POI Inclusion/Exclusion Application

Similar to MAS-US-03, however, it attempts to find cluster-specific rules for a group of users, prioritizing the first user's cluster if no cluster identifier is provided. If no cluster-specific rules are found, it uses global rules. The method extracts the group's attraction type preferences from the matching rules. If no preferences are found, it returns an empty 'POISuggestionsDto'. After, it

requests POI data to POIMS based on the liked and disliked attraction types and returns the POI to include and/or exclude from the recommendation.

Code Snippet 23 shows the extraction of the group attraction type preferences from the matching rules.

```
private static (HashSet<int> likedAttractionTypes, HashSet<int>
dislikedAttractionTypes)
    ExtractGroupAttractionPreferencesFromMatchingRules(List<AssociationRule>
rules, List<User> users)
{
    var likedAttractionTypes = new HashSet<int>();
    var dislikedAttractionTypes = new HashSet<int>();

    foreach (var rule in rules)
    {
        // Count the number of users who follow this rule
        var usersFollowingRule = 0;
        foreach (var user in users)
        {
            if (matchesTransactions(rule.antecedent, user))
            {
                usersFollowingRule++;
            }
        }

        // Calculate the percentage of users following this rule
        var percentage = (double)usersFollowingRule / users.Count;
        // Check if at least 50% of users follow this rule
        if (!(percentage >= 0.5)) continue;

        /*The consequent may not have transactions of type
"LIKES_ATTRACTION_TYPE" or "DISLIKES_ATTRACTION_TYPE"
        Filter the type of the transaction type to only add attraction type
preferences */
        likedAttractionTypes.UnionWith(rule.consequent.Where(at => at.type
is AprioriTransactionType.LIKES_ATTRACTION_TYPE)
            .Select(at => int.Parse(at.transaction)));

        dislikedAttractionTypes.UnionWith(rule.consequent.Where(at =>
at.type is AprioriTransactionType.DISLIKES_ATTRACTION_TYPE)
            .Select(at => int.Parse(at.transaction)));
    }

    return (likedAttractionTypes, dislikedAttractionTypes);
}
```

Code Snippet 23 - Group Matching Preferences based on Rules

The key distinction between the group and individual POI suggesting approaches is how they handle user preferences based on association criteria.

The group method processes rules for multiple users rather than just one. It iterates through the rules, counting how many users in the group follow each one by comparing the rule's antecedent to the user's traits and preferences. If at least half of the users follow a rule, the liked and/or disliked attraction types are added to the corresponding sets of attraction types. Finally, it also communicates with POIMS to retrieve points of interest based on the attraction types to include and/or exclude.

4.2. Functional Tests

This section describes the testing procedures and case scenarios used to assure the reliability of the developed functionalities. We describe the numerous test cases used to validate each use case, such as clustering users based on personality measurements, developing association rules, and producing POI inclusion/exclusion lists. Code tests were designed using the .NET *Moq* and *Xunit* frameworks to ensure that the system appropriately groups users, uses the Apriori algorithm, and delivers personalized and group-based inclusion/exclusion POI.

All test case scenarios were also evaluated with Postman to ensure validation of the API endpoints and their behaviors. Each scenario was created to test the integration of various aspects of the functionalities implemented while reflecting all the case situations specified.

Finally, all these test case scenarios were replicated and assessed as acceptance tests by the product owner for all use cases. This thorough assessment guaranteed that every feature and functionality fulfilled the specifications given in the requirements acceptance criteria.

4.2.1. IT-MAS-US-01 Cluster Users Based on Personality

These tests guaranteed that users were accurately reassigned to clusters based on their personality traits. First, Table 10 documents several testing case scenarios for the clustering of users.

Table 10 - UC1 Test Case Scenario Overview

Test Case Scenario	Purpose	Assertion
Clustering Should Update User Clusters	This test method intends to test if different users are updated/assigned to the closest cluster	Users are accurately assigned to the closest clusters based on their personality characteristics.
Clustering with Empty User List	Tests the service's behavior when no users are provided to cluster them.	The method executes with no user assignment or updates on the clusters and without throwing any exceptions.
Clustering User with no matching Cluster	This test method intends to test if user is assigned to a new cluster since hasn’t a matching cluster	User is accurately assigned to a newly created cluster.

Code Snippet 24 shows in more detail the implementation of the first case scenario described, where the clustering should assign clusters to users, as one of the main expected behaviors.

```
[Fact]
public async Task Reclustering_ShouldUpdateUserClusters()
{
    // Arrange
    var users = new List<User>
    {
        new() { Id = "1", openness = 0.1, conscientiousness = 0.2,
extraversion = 0.3, agreeableness = 0.4, neuroticism = 0.5 },
        new() { Id = "2", openness = 0.2, conscientiousness = 0.3,
extraversion = 0.4, agreeableness = 0.5, neuroticism = 0.6 }
    };

    var clusters = new List<Cluster>
    {   new() { clusterId = "c1", openness = 0.1, conscientiousness = 0.2,
extraversion = 0.3, agreeableness = 0.4, neuroticism = 0.5 },
        new() { clusterId = "c2", openness = 0.2, conscientiousness = 0.3,
extraversion = 0.4, agreeableness = 0.5, neuroticism = 0.6 } };

    _mockUserRepo.Setup(repo => repo.AllUsers()).ReturnsAsync(users);
    _mockClusterRepo.Setup(repo =>
repo.AllNonEmptyClusters()).ReturnsAsync(clusters);

    // Act
    await _service.Reclustering();

    // Assert
    _mockUserRepo.Verify(repo => repo.UpdateObjects(It.Is<List<User>>(u =>
            u.Count == 2 &&
            u.First(user => user.Id == "1").clusterId == "c1" && // User 1
should be assigned to cluster 1
            u.First(user => user.Id == "2").clusterId == "c2"    // User 2
should be assigned to cluster 2
    )), Times.Once);
}
```

Code Snippet 24 - Valid Clustering Test Should Update User's Clusters

The setup are two users with specific personality traits and two clusters with matching personalities for each user, respectively. This means the first user should be assign to the first cluster and the second user to the second cluster. All test cases described succeeded.

Figure 20 shows a scenario in MAMS prototype. This representation reflects how the users were arranged in 3 clusters created in MAMS database.

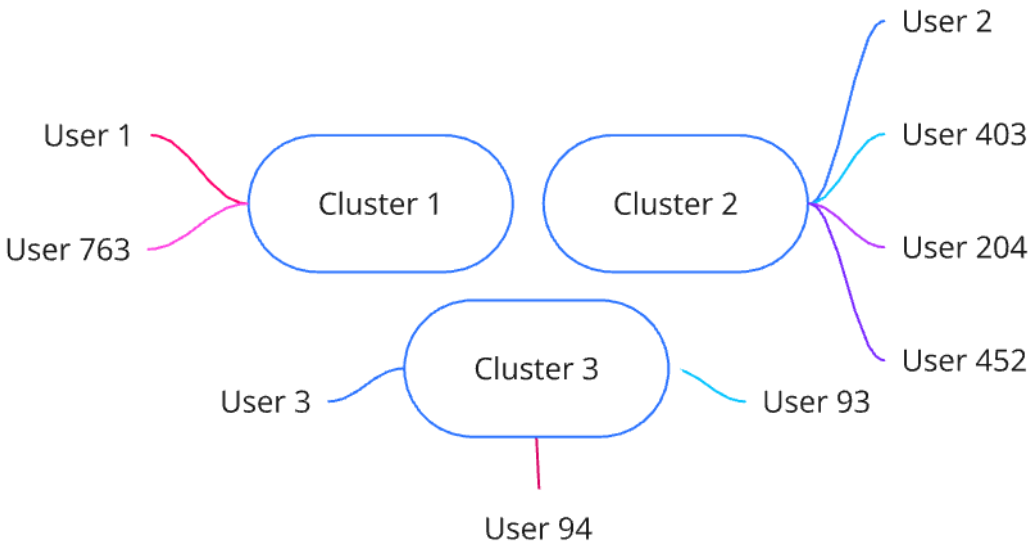

Figure 20 - MAMS Users and Respective Clusters

This assisted in testing if the clustering was being stored in MAMS database. This is a partial representation of all clusters in the database since there was a total of 158 clusters organizing 1035 users by their personality. Users are represented by masked names to protect sensitive data.

4.2.2. IT-MAS-US-02 Generate Association Rules

This part focused on testing the development of association rules implemented in section 4.1.3. Table 11 entails the testing of the Apriori algorithm and the resulting association rules.

Table 11 - UC2 Test Case Scenarios Overview

Test Case Scenario	Purpose	Assertion
Generate Association Rules with users	Tests the generation of association rules based on user attributes and preferences, both by cluster and globally	Association rules are generated accurately based on user data and preferences, either by cluster or globally
Generate Rules with No User Data	Tests the behavior of the service when no user data is provided	The method executes without generating any rules or throwing exceptions

The first case scenario, illustrated in Code Snippet 25, explores two users with two same attraction type preferences, but different gender and travel companions. This test expects the correlation between the attraction type preferences only, therefore, two rules.

```
[Fact]
public async Task
GenerateAssociationRulesWithUsers_DifferingCLusterThreshold_ShouldGenerateR
ules()
{
    // Arrange
    mockClusterRepository.Setup(repo =>
repo.GetAllClustersWithAllUserInfo())
        .ReturnsAsync([
            new Cluster
            {
                clusterId = "1",
                users = [new User { Id = "1",  UserName = "Fausto",
                        attractionTypes = new List<UserAttractionType>()
                        { new() { attractionType = new AttractionType() {
attractionArea = 1 }, preference = true },
                                new() { attractionType = new AttractionType() {
attractionArea = 2 }, preference = true } },
                        gender = "M",
                        travelCompanion = "Friends" },
                    new User   { Id = "2", UserName = "Alice",
                        attractionTypes = new List<UserAttractionType>()
                        { new() { attractionType = new AttractionType() {
attractionArea = 1 }, preference = true },
                                new() { attractionType = new AttractionType() {
attractionArea = 2 }, preference = true }
                        },
                        gender = "F",
                        travelCompanion = "Family" }]}]);

    // Act
    var resultGlobal = await
_associationRules.generateRulesToPoiRecommendation(supportThreshold:0.99,
confidenceThreshold:0.99, minClusterThreshold:10);
    // Act
    var resultCluster = await
_associationRules.generateRulesToPoiRecommendation(supportThreshold:0.99,
confidenceThreshold:0.99, minClusterThreshold:1);

    // Assert
    Assert.Equal(2 ,resultGlobal.rulesByUsers.Count);
    Assert.Equal(2 ,resultCluster.rulesByCluster.Count);
}
```

Code Snippet 25 - Valid Rules Differing Minimum Cluster Threshold Test

This test method acts in two ways, one creating rules with a cluster threshold above the one cluster that exists, and other with the cluster threshold same as the number of clusters, so one can create and test the global rules and the other to create cluster rules, respectively.

Code Snippet 26 describes valid rules differing the minimum support threshold, with a low support, so every item will be considered as frequent, therefore, the number of rules is expected increase. With high support levels, the rules will be more restrictive and only the attraction types are considered frequent.

```
[Fact]
public async Task
GenerateAssociationRulesWithUsers_DifferingSupportThreshold_ShouldGenerateRul
es()
{
    // Arrange
    mockClusterRepository.Setup(repo => repo.GetAllClustersWithAllUserInfo())
        .ReturnsAsync([
            new Cluster {
                clusterId = "1",
                users = [new User  { Id = "1", UserName = "Fausto",
                        attractionTypes = new List<UserAttractionType>()
                        { new() { attractionType = new AttractionType() {
attractionArea = 1 }, preference = true },
                             new() { attractionType = new AttractionType() {
attractionArea = 2 }, preference = true } },
                        gender = "M",
                        travelCompanion = "Friends" },
                    new User { Id = "2", UserName = "Alice",
                        attractionTypes = new List<UserAttractionType>()
                        { new() { attractionType = new AttractionType() {
attractionArea = 1 }, preference = true },
                             new() { attractionType = new AttractionType() {
attractionArea = 2 }, preference = true }},
                        gender = "F",
                        travelCompanion = "Family" }]}]);

    // Act
    var resultHighSup = await
_associationRules.generateRulesToPoiRecommendation(supportThreshold:0.99,
confidenceThreshold:0.1, minClusterThreshold:10);
    // Act
    var resultLowSup = await
_associationRules.generateRulesToPoiRecommendation(supportThreshold:0.01,
confidenceThreshold:0.1, minClusterThreshold:10);

    // Assert
    Assert.Equal(2 , resultHighSup.rulesByUsers.Count);
    Assert.Equal(22 , resultLowSup.rulesByUsers.Count);
}
```

Code Snippet 26 - Valid Rules Differing Minimum Support Threshold Test

Code Snippet 27 illustrates valid rules differing the minimum confidence threshold, with low confidence thresholds, the algorithm is also less restrictive making relationships between the frequent itemsets, therefore, it's expected more rules. With high confidence values, the relationship between frequent itemsets must be stronger, consequently, it's estimated less rules.

```
[Fact]
public async Task
GenerateAssociationRulesWithUsers_DifferingConfidenceThreshold_ShouldGenera
teRules()
{
    // Arrange
    mockClusterRepository.Setup(repo =>
repo.GetAllClustersWithAllUserInfo())
        .ReturnsAsync([
            new Cluster
            {   clusterId = "1",
                users = [new User { Id = "1", UserName = "Fausto",
                        attractionTypes = new List<UserAttractionType>()
                        { new() { attractionType = new AttractionType() {
attractionArea = 1 }, preference = true },
                                new() { attractionType = new AttractionType() {
attractionArea = 2 }, preference = true } },
                        gender = "M",
                        travelCcompanion = "Friends" },
                    new User {Id = "2", UserName = "Alice",
                        attractionTypes = new List<UserAttractionType>()
                        { new() { attractionType = new AttractionType() {
attractionArea = 1 }, preference = true },
                                new() { attractionType = new AttractionType() {
attractionArea = 2 }, preference = true }
                        },
                        gender = "F",
                        travelCompanion = "Family" }]}]);

    // Act
    var resultHighCon = await
_associationRules.generateRulesToPoiRecommendation(supportThreshold:0.001,
confidenceThreshold:0.99, minClusterThreshold:10);
    // Act
    var resultLowCon = await
_associationRules.generateRulesToPoiRecommendation(supportThreshold:0.001,
confidenceThreshold:0.001, minClusterThreshold:10);

    // Assert
    Assert.Equal(2 , resultHighCon.rulesByUsers.Count);
    Assert.Equal(28 , resultLowCon.rulesByUsers.Count);
}
```

Code Snippet 27 - Valid Rules Differing Minimum Confidence Threshold Test

4.2.3. IT-MAS-US-03 Request POI Inclusion/Exclusion list of an Individual

Table 12 describes the test case scenarios focused on the feature which involves establishing POI to include or exclude based on association rules of an individual user.

Table 12 - UC3 Test Case Scenarios Overview

Test Case Scenario	Purpose	Assertion
POI to Include/Exclude by User prioritizing cluster rules	Tests the generation of POI inclusion/exclusion list based on cluster user associated rules	A POI list is accurately generated based on the user's associated rules.
POI to Include/Exclude by User prioritizing global rules	Tests the generation of POI inclusion/exclusion list based on global user associated rules	A POI list is accurately generated based on the user's associated rules.
User Not Found	Tests the behavior when the user specified does not exist	Throws the custom exception 'UserNotFoundException'
Rules Not Generated	Tests the behavior when no association rules are available	Throws the custom exception 'RulesNotGeneratedException'
No rules found	Tests the scenario when no rules are available	Returns empty list of POI to include or exclude
No matching rules found	Tests the scenario when no rules match the user's attributes or preferences	Returns empty list of POI to include or exclude

Code Snippet 28 details the first case scenario described that accurately generates POI inclusion or exclusion lists based on specific cluster rules associated.

```
[Fact]
public async Task PoitoIncludeExcludeByUser_ValidUser_ReturnsPoiList()
{
    var associationRules = new RecommendationRule
    {
        rulesByUsers = [],
        rulesByCluster =
        [
            new AssociationRule
            {
                clusterId = "1",
                antecedent = [
                    new AprioriTransaction
                        { type =
AprioriTransactionType.LIKES_ATTRACTION_TYPE, transaction = "Museums" }
                ],
                consequent =
                [
                    new AprioriTransaction
                        { type =
AprioriTransactionType.LIKES_ATTRACTION_TYPE, transaction = "Beaches" }
                ]
            }
        ]
    };

    _mockMasService.Setup(service => service.AprioriAssociationRules())
        .Returns(associationRules);

    // Act
    var result = await _POIuggestionService.PoiInclusionExclusionByUser
(_user.UserName);

    // Assert
    Assert.NotNull(result);
    Assert.Equal("Beaches",
result.likedPOI[0].attractionTypeCategory.ToString());
}
```

Code Snippet 28 - Valid Individual POI List Test Returns Non-Empty POI

The rule example is given users that like ‘Museums’, also like ‘Beaches’. Given a valid user with the positive preference in ‘Museums’, the rule must include POI of the category ‘Beaches’ and no POI to exclude.

Code Snippet 29 details test the custom exception ‘UserNotFoundException’ test case scenario in case a user is not found.

```
[Fact]
public async Task PoiIncludeExcludeByUser_UserNotFoundException()
{
    // Arrange
    const string username = "non_existing_user"; // Username that does not
exist in the repository
    mockUserRepository.Setup(repo => repo.GetUserByUsername(username))
        .ReturnsAsync((User)null); // Simulate user not found

    // Act + Assert
    var exception = await Assert.ThrowsAsync<UserNotFoundException>(() =>
_POIuggestionService.PoiInclusionExclusionByUser (username));

    // Assert
    Assert.NotNull(exception);
}
```

Code Snippet 29 - Invalid POI Include/Exclude list with “No User Found” Exception

This test ensures that when requesting POI to include/exclude a non-existent user, the method rightly returns a custom 'UserNotFoundException' exception. It utilizes a mock repository to simulate the user's absence.

4.2.4. IT-MAS-US-04 Request POI Inclusion/Exclusion list of a Group

Table 13 emphasis the MAS-US-04 test case scenarios regarding generation of POI inclusion or exclusion lists for a group.

Table 13 - UC4 Test Case Scenarios Overview

Test Case Scenario	Purpose	Assertion
POI to Include/Exclude by group of users prioritizing cluster rules	Tests the generation of POI inclusion/exclusion list based on a specific cluster associated rules	A POI list is accurately generated regarding the group based on the specific cluster associated rules
POI to Include/Exclude by group prioritizing global rules	Tests the generation of POI inclusion/exclusion list based on global user associated rules	A POI list is accurately generated regarding the group based on the global associated rules.

POI to Include/Exclude by group of users prioritizing cluster rules without cluster identifier	Tests the generation of POI inclusion/exclusion list based on the first user of the group cluster associated rules	A POI list is accurately generated regarding the group based on the first user cluster associated rules
POI to Include/Exclude by group of users with less than 50% matching on the rules	Tests the generation of POI inclusion/exclusion list on associated rules, but less than half of users match the rule	Returns empty list of POI to include or exclude
Users Not Found	Tests the behavior when no user of the group was found	Throws the custom exception 'UserNotFoundException'
Rules Not Generated	Tests the behavior when no association rules are available	Throws the custom exception 'RulesNotGeneratedException'
No rules found	Tests the scenario when no rules are available	Returns empty list of POI to include or exclude
No matching rules found	Tests the scenario when no rules match the group attributes or preferences	Returns empty list of POI to include or exclude

Code Snippet 30 illustrates a similar method to the test case scenario implementation in Code Snippet 28, with some nuances for groups, such having a specific cluster identifier to match the group preferences with the cluster associated rules.

```
[Fact]
public async Task
PoitoIncludeExcludeByGroup_ValidUsers_ReturnsPOItoIncludeExclude()
{
    var associationRules = new RecommendationRule
    {
        rulesByUsers = [],
        rulesByCluster =
        [
            new AssociationRule
            {
                clusterId = "2",
                antecedent = [
                    new AprioriTransaction
                        { type =
AprioriTransactionType.DISLIKES_ATTRACTION_TYPE, transaction = "Museums" }
                ],
                consequent =
                [
                    new AprioriTransaction
                        { type =
AprioriTransactionType.DISLIKES_ATTRACTION_TYPE, transaction = "Beaches" }
                ]
            }
        ]
    };

    _mockMasService.Setup(service => service.AprioriAssociationRules())
        .Returns(associationRules);

    // Act
    var result = await
_POIuggestionService.PoiGroupInclusionExclusionList([_group[0].UserName,
_group[1].clusterId]);

    // Assert
    Assert.NotNull(result);
    Assert.Equal("Beaches", result.
dislikedPOI[0].attractionTypeCategory.ToString());
}
```

Code Snippet 30 - Valid Group POI List Test Returns Non-Empty POI

Now, the example illustrates a group of two users that dislikes ‘Museums’ and the rules are prioritized by the cluster of the second user, cluster with identifier “2”. The only rule of this cluster is that users that dislikes ‘Museums’ also dislikes ‘Beaches’, therefore, is expected that the list of POI to include to be empty and the list to exclude to be POI of attraction type ‘Beaches’.

Finally, Code Snippet 31 displays a valid group POI Inclusion/Exclusion list test with not enough matching rules. The rule example indicates that disliking ‘Museums’ means liking

'Beaches'. This means that the same group of users, both liking 'Museums', will not have matching rules, therefore, will not surpass the fifty-percentage defined to include to the POI inclusion/exclusion list.

```
[Fact]
public async Task
PoitoIncludeExcludeByGroup_LessThan50PercentUsersWithRules_ReturnsEmptyPOI
uggestions()
{
    // Arrange
    var associationRules = new RecommendationRule
    {
        rulesByUsers = [],
        rulesByCluster =
        [
            new AssociationRule
            {
                clusterId = "1",
                antecedent =
                [
                    new AprioriTransaction
                        { type =
AprioriTransactionType.LIKES_ATTRACTION_TYPE, transaction = "Museums" }
                ],
                consequent =
                    [new AprioriTransaction() { type =
AprioriTransactionType.DISLIKES_ATTRACTION_TYPE, transaction = "Beaches"
}]
            }
        ]
    };

    _mockMasService.Setup(service => service.aprioriAssociationRules())
        .Returns(associationRules);

    // Act
   var result = await _POIuggestionService.PoiGroupInclusionExclusionList
(new List<string> { _group[0].UserName, _group[1].UserName });

    // Assert
    Assert.NotNull(result);
    Assert.Empty(result.likedPOI);
    Assert.Empty(result.dislikedPOI);
}
```

Code Snippet 31 - Valid Group POI List Test with not enough Matching Rules

5. Solution Experimentation and Evaluation

This section unfolds the tests made on MAMS using the Apriori algorithm and providing POI inclusion/exclusion lists based on the generated rules, to help answer the questions raised in section 1.3. To collect performance data, experiments involved creating user groups and simulating interactions. Evaluation came after the experimental phase, when the collected data was examined to determine the system's performance and reliability, also based in the requirements established in section 3.4. To test the quality of the quality attributes, it was used the Goals, Questions, Metrics (GQM) approach.

5.1.Goals, Questions, Metrics

GQM is a structured process for defining objectives (Goals), developing questions, described in section 1.3, to examine those objectives (Questions), and determining metrics to quantify the answers (Basili et al., 1994). This approach ensures a methodical review of whether the goals are reached and allows for continual improvement. Table 14 illustrates those goals, questions and metrics, followed with associated research questions that are related.

Table 14 - GQM Approach

Goals	Questions	Metrics	Associated Research Question(s)
G1 - Provide POI based on group matching rules	How accurately does the system predict the preferences of groups?	Prediction Accuracy	**RQ2**
	How much time does the system take to provide the POI Inclusion/Exclusion list?	POI Inclusion/Exclusion list response time limit of 5 seconds	
G2 - Provide POI based on an individual matching rules	How accurately does the system predict the preferences of individual?	Prediction Accuracy	**RQ2**

	How much time does the system take to provide the POI Inclusion/Exclusion list?	POI Inclusion/Exclusion list response time limit of 5 seconds	
G3 - Provide Rules based on traits and attraction type preferences	How accurately does the system create association rules based on a dataset of tourists?	Prediction Accuracy	**RQ1 RQ3**
	How much time does the system take to provide the association rules?	Rules Generation response time limit of 10 seconds	
G4- Ensure scalability of the system for larger datasets with Apriori	How does the system performance scale regard Apriori with increasing dataset sizes?	Scalability Performance with average response time limit of 20 seconds	**RQ4**

The GQM table, together with the research questions related to the algorithm goals, describes goals for the accuracy, efficiency, and scalability of the Apriori implementation in the MAS system. It contains relevant questions for each goal to determine whether the goals of the algorithm itself are being met, as well as precise metrics to quantify the replies to these questions.

The metrics were obtain using indicator tools such Postman via integration tests with different datasets and in the tests described in section 4.2. The following sections specifies the hypotheses and then describes the results given the metrics defined in GQM.

5.2. Specification of research hypotheses

In this section, to evaluate the functionalities and the behavior of the algorithm within the system, research hypotheses are created for each question related to a goal described in the previous section, so that the defined metrics can meet the criteria and provide evidence to assess the plausibility of the null hypothesis.

Criteria 1 - Group Preference Match Rate - How accurately does the system predict the preferences of groups?

- H0: All implementation tests for requesting the POI inclusion/exclusion list of a group are successful.
- H1: At least one implementation test for requesting the POI inclusion/exclusion list of a group fails.

Criteria 2 – Group POI Inclusion/Exclusion List Performance - How much time does the system take to provide the POI Inclusion/Exclusion list of groups?

- H0: The system takes an average of 5 seconds or less to provide the POI Inclusion/Exclusion list of groups.
- H1: The system takes an average of more than 5 seconds to provide the POI Inclusion/Exclusion list of groups.

Criteria 3 - Individual Preference Match Rate - How accurately does the system predict the preferences of individuals?

- H0: All implementation tests for requesting the POI inclusion/exclusion list of an individual are successful.
- H1: At least one implementation tests for requesting the POI inclusion/exclusion list of an individual fails.

Criteria 4 - Individual POI Inclusion/Exclusion List Performance - How much time does the system take to provide the POI Inclusion/Exclusion list?

- H0: The system takes an average of 5 seconds or less to provide the POI Inclusion/Exclusion list of an individual.
- H1: The system takes an average of more than 5 seconds to provide the POI Inclusion/Exclusion list of an individual.

Criteria 5 - Association Rule Accuracy - How accurately does the system create association rules based on a dataset of tourists?

- H0: All comparison and implementation tests for requesting the generation of association rules are successful.
- H1: At least one implementation tests for requesting the generation of association rules fails.

Criteria 6 – Rules Generation Performance - How much time does the system take to provide the association rules?

- H0: The system takes an average of 10 seconds or less to generate the association rules.
- H1: The system takes an average of more than 10 seconds to generate the association rules.

Criteria 7 – Performance Scalability Rate - How does the system performance scale with increasing dataset sizes?

- H0: The system takes a total average of 20 seconds or less to for requesting the generation of association rules in each dataset.
- H1: The system takes an average of more than 20 seconds for requesting the generation of association rules in at least one dataset.

5.3. GQM Results

This section focuses on the results of the tests that were made to validate the acceptance criteria, described in 3.3.1, and the concrete evaluation of the metrics for each goal defined in section 6.1 based on a hypothesis test already described.

5.3.1. G1 - Provide POI based on group matching rules

All tests given in section 4.2.4 serve to answer the question " How accurately does the system predict the preferences of groups?" to determine if the system is providing accurate POI based on a specific group. To make a prediction accuracy, these tests projected the expected results based on example test case circumstances where the list is provided, and offered the following results, first, for groups. Table 15 displays the results of the IT-MAS-US-04 test case scenarios.

Table 15 - IT-MAS-US-04 Test Case Scenario Results

Test Case Scenario	Expected Result	Actual Result
POI to Include/Exclude by group of users prioritizing cluster rules	The system provides POI based on rules described with regards to the cluster specified	The system provided the same POI as expected with regards to the specific cluster rules
POI to Include/Exclude by group prioritizing global rules	The system provides POI based on global rules	The system provided the same POI as expected with regards to the global rules
POI to Include/Exclude by group of users prioritizing cluster rules without cluster identifier	The system provides POI based on the first user of the group cluster associated rules	The system provided the same POI as expected with regards to the first user of the group cluster associated rules
POI to Include/Exclude by group of users with less than 50% matching on the rules	The system provides an empty POI list	The system provided the expected empty POI list
No POI found	The system provides no POI	The system provided an empty list of POI to include or exclude
No rules found	The system provides an empty POI list	The system provided the expected empty POI list
No matching rules found	The system provides an empty POI list	The system provided the expected empty POI list

Based on the results, all the case scenarios produced the expected results. The prediction accuracy statistic has a success rate of 100%, therefore, the corresponding **H0** hypothesis in **Criteria 1** failed to be rejected since all tests were well succeeded.

Figure 21 displays the result scenario within MAMS with a group of 4 users: User 3, User 93, User 94 and User 5. The first 3 users belong to the same cluster that has as a negative preference when traveling companion is ‘Friends’ on the attraction type ‘Visit archaeological/ruin sites’, category with 4 POI. Only User 93 and User 94 has as preference that traveling companion. This

means that 50% of the group has the trait that matches the rule, thus, the correspondent POI of the stated attraction type should be on the “Exclude” list.

Another rule from the cluster is a positive preference for male users of ‘Go to a venue with live music’ attraction type, category with 7 POI. All users are male, meaning that a positive preference in POI of this attraction type should reflect in the “Inclusion” POI list.

Finally, the request of the POI Inclusion/Exclusion list was done by the group mentioned and with the cluster identifier of the first 3 users, therefore, this scenario should correspond to the first test case scenario result described in Table 15 and provide all 4 POI to include of the attraction type mentioned. All rules were bootstrapped.

Figure 21 - MAS-US-03 Grouplanner Case Scenario Result

As illustrated, the result in MAMS brought the correspondent POI from ‘Go to a venue with live music’ to include and POI of the category ‘Visit archaeological/ruin sites’ to exclude.

Regarding **Criteria 2**, the product owner defined a limit of 10 seconds for the average response time for the system to provide POI include/exclude to groups, described in section 5.1.4. To average the time responses, 20 POI inclusion/exclusion list to groups requests were made, via postman to simulate a Recommendation Engine request, presented in Table 16.

Table 16 - Time response for MAS-US-04 functionality

Number of Requests	Worst Response Time (s)	Best Response Time (s)	Average Response Time (s)
20	3.03	0.84	0.91

Based on the results, the corresponding **H0** hypothesis can't be rejected since the average time is less than 5 seconds.

5.3.2. G2 - Provide POI based on individual matching rules

To determine if the system is providing accurate POI based on an individual, all tests described in section 5.2.3 serves to answer the question "How accurately does the system predict the preferences of an individual?". Therefore, to make a prediction accuracy, these tests predicted the expected results based on example test case scenarios where the list is provided, presented the following results, firstly, for individuals. Therefore, Table 17 shows the results of the IT-MAS-US-03 test case scenarios.

Table 17 - IT-MAS-US-03 Test Case Scenario Results

Test Case Scenario	Expected Result	Actual Result
POI to Include/Exclude by an individual prioritizing the cluster rules	The system provides the expected POI based rules described with regards to the cluster of the individual	The system provided the same POI as expected with regards to the cluster rules of the individual
POI to Include/Exclude by User prioritizing global rules	The system provides the expected POI based rules described with regards to the global rules	The system provided the same POI as expected with regards to the global rules
No POI found	The system provides no POI	The system provided an empty list of POI to include or exclude
No rules found	The system provides no POI	The system provided an empty list of POI to include or exclude

No matching rules found	The system provides no POI	The system provided an empty list of POI to include or exclude
No matching rules found	The system provides no POI	The system provided an empty list of POI to include or exclude

Based on the results, all case scenarios gave the correct results as predicted. This means there is a 100% success rate in the prediction accuracy metric, therefore, all implementation tests passed, which means **H0** in **Criteria 3** can't be rejected.

Figure 22 displays the result scenario within MAMS with a tourist user (User 7) that has cluster rules regarding a positive preference to married users in the attraction type 'Visit Nature or Wildlife Reserves', a category of a set of 4 POI. There are no more positive or negative preferences in the cluster rules. Since User 7 is married, this scenario should correspond to the first test case scenario result described in Table 17 and provide all 4 POI to include of the attraction type mentioned. All rules were bootstrapped.

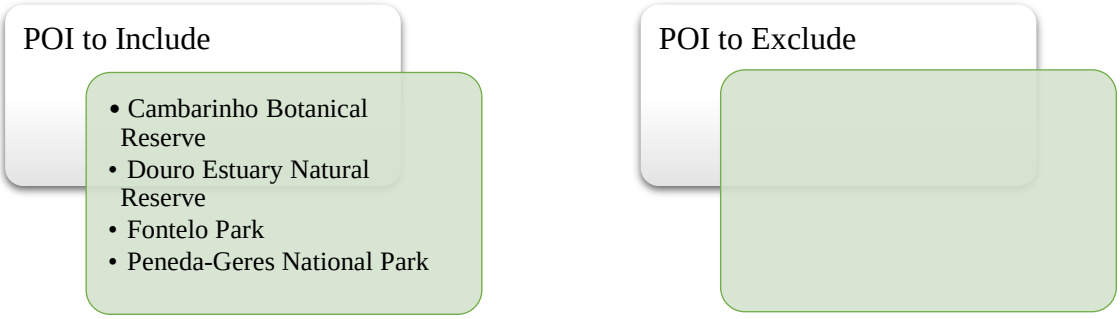

Figure 22 - MAS-US-04 Grouplanner Case Scenario Result

As illustrated, the result brought the correspondent POI of the mentioned attraction type. Since there are no rules on negative preferences, there is not any POI to exclude.

Regarding **Criteria 4**, the product owner defined a limit of 5 seconds for the average response time for the system to provide POI include/exclude to an individual, described in section 5.1.4. To

average the time responses, 20 POI inclusion/exclusion list requests were made, via postman to simulate the request, presented in Table 18.

Table 18 - Time response for MAS-US-03 functionality

Number of Requests	Worst Response Time (s)	Best Response Time (s)	Average Response Time (s)
20	2.92	0.83	0.86

Based on the results, the **H0** hypothesis can't be rejected since the average time is less than 5 seconds.

5.3.3. G3 - Provide Rules based traits and attraction type preferences

This section analyses the metrics for the goal of providing association rules with Apriori based on tourist characteristics and attraction type preferences. Firstly, to answer the question of "How accurately does the system create association rules based on a dataset of tourists?", a prediction accuracy is made to the tests of generating association rules described in section 5.2.2. Table 19 illustrates the IT-MAS-US-02 test case scenarios results.

Table 19 - IT-MAS-US-02 Test Case Scenario Results

Test Case Scenario	Expected Result	Actual Result
Generate Association Rules with users	The system provides a list of rules by each cluster, if any, and a list of rules to the remain users	The system provided the expected list of rules by cluster, and a list of rules for global users
Generate Rules with No User Data	The system provides an empty list of rules, either by cluster of global	The system provided an empty list of rules, as expected

Also, due to the algorithm's complexity, a tool was also used to test the accuracy of the algorithm. Weka is a collection of machine learning algorithms and data mining tools (Milind et al., 2011). It offers a user-friendly interface for data preprocessing, classification, regression,

clustering, and visualization, as well as the ability to use machine learning algorithms without substantial programming skills.

To evaluate the accuracy of the Apriori algorithm, various test cases were conducted using a dataset of 1,034 users, allowing for a comprehensive assessment of the algorithm's performance. Table 20 demonstrates the test case scenarios of the Apriori algorithm comparison to Weka's.

Table 20 - Apriori vs Weka Comparison

Test Case Scenario	**Description**	**Match Result**
Small Dataset	Test with a small dataset of 10 transactions of 5 users.	Matched
Large Dataset	Test with a large dataset of 1,034 users with 83 transactions each.	Matched
Varied Support Threshold	Test with varying support thresholds (85%, 90%, 95%).	Matched
Varied Confidence Threshold	Test with varying confidence thresholds (85%, 90%, 95%).	Matched
Edge Case	Dataset where each transaction contains only one different item (no two sized itemset).	Matched (no rules)

Based on the actual results, it corresponds to the expected ones, therefore, we can't reject the corresponding **H0** hypothesis from **Criteria 5** since all comparison and implementation tests regarding the MAS-US-02 tests succeeded.

Figure 23 illustrates the main rules found in MAMS with the large dataset with 90% support and confidence thresholds.

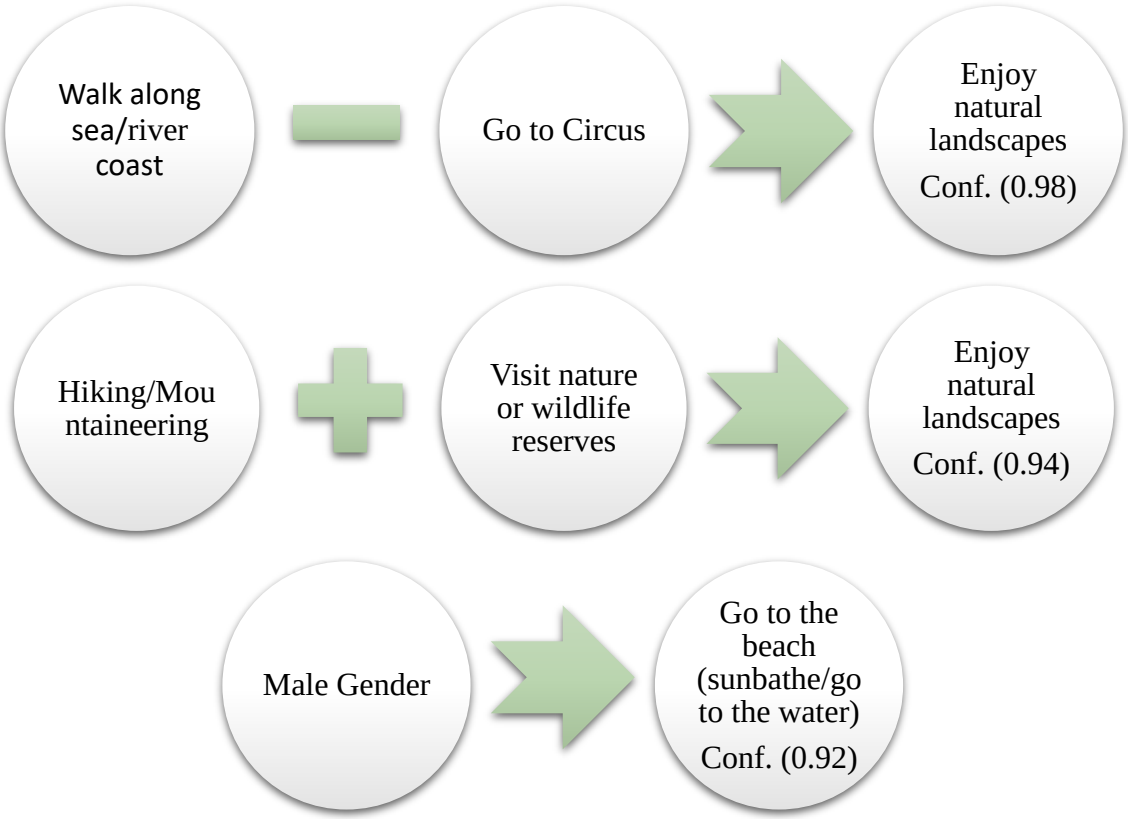

Figure 23 - MAMS Apriori Results

The association rules are based on the tourist's dataset registered in MASIn the second rule, it was found a negative preference. The first rule suggests that people that like to 'Walk along in sea/river cost' and don't like 'Going to circus', also like to 'Enjoy natural landscapes'. Second rule, if a tourist enjoys "Go hiking/mountaineering" and likes to 'Visit nature or wildlife reserves', they are also likely to enjoy natural landscapes. Finally, the last rules suggest that typically male tourists enjoy going to the beach for a sunbathe or to go to the water. It's noticeable the different rules between attraction types or user traits with different confidence levels above the threshold defined.

Regarding the question "How much time does the system take to provide the association rules?", the product owner defined an average of 10 seconds for the current dataset of MAS, regarding the implementation of MAS-US-02. Therefore, 20 requests, via Postman, of generation of Association Rules were made, with the minimum stipulated 95% of confidence and support thresholds, Table 21 plays the time response of MAS-US-02 functionality.

Table 21 - Time response for MAS-US-02 functionality

Number of Requests	Worst Response Time (s)	Best Response Time (s)	Average Response Time (s)
20	6.87	4.03	4.85

As shown, the **H0** hypothesis of **Criteria 6** can't be rejected in this evaluation, since the average response time is less than 10 seconds.

5.3.4. G4 - Ensure scalability of the system for larger datasets with Apriori

This section deep dives into the scalability aspect of the generation of the association rules with Apriori, and the results were obtained with integration tests via Postman. Based on that, different datasets were tested with 20 requests of generation of rules with 90% support and confidence thresholds, each to analyze the variation of the average time responses, in seconds, between 100, 500, 1000, 1500 and 2000 users. Each user, in any case, has the same number of 83 transactions, meaning attributes and/or preferences, that will delve into the generation of the association rules. This means that the total number of transactions being accountable in the algorithm is the number of users times the number of transactions of the respective user, with the average response times, as seen in Figure 24.

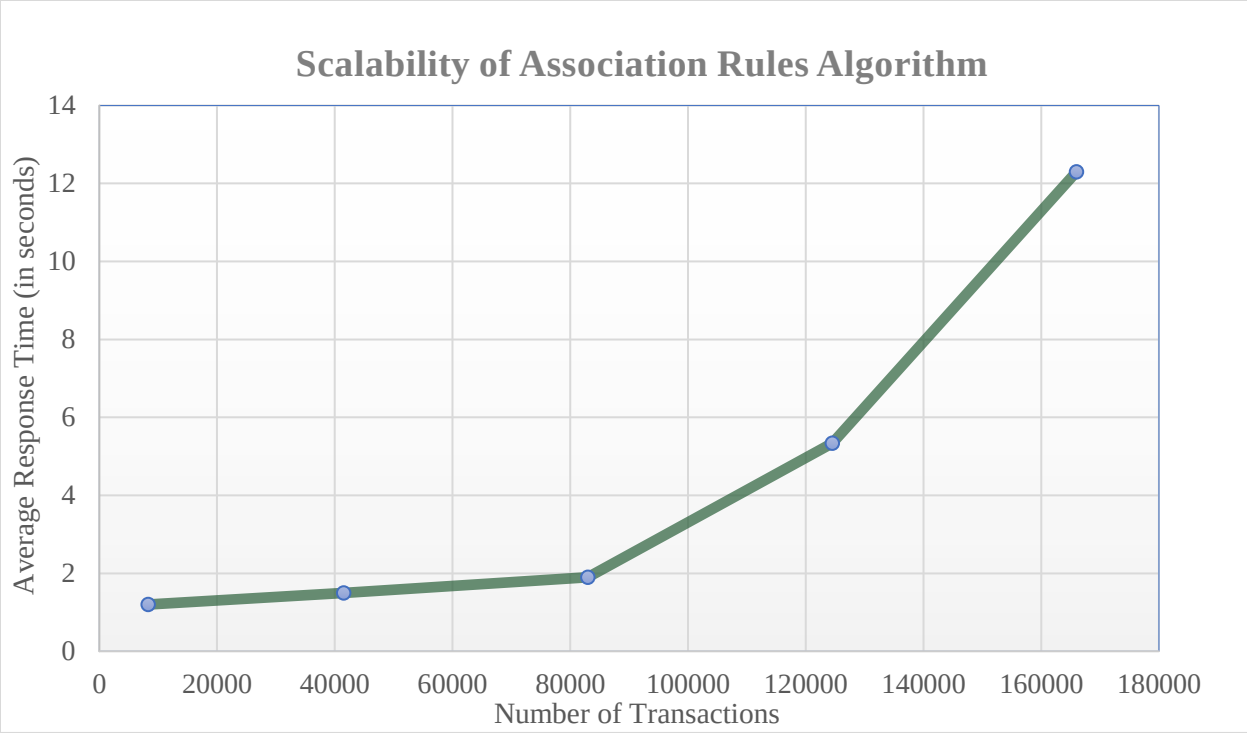

Figure 24 - Scalability of Association Rules Algorithm

The graphic indicates that the algorithm is optimal in terms of number of transactions it can support regarding the criteria defined, despite its noticeable that it's not optimal for regarding the scalability factor, however, the **H0** hypothesis in **Criteria 7** can't be rejected since the average response time does not surpass the 20 seconds' limit.

5.4. Result analyses

This section analyses whether the implemented solution met both the functional and non-functional requirements outlined in the system's design. It provides an overview of the implementation, testing results, and metrics evaluation to verify if the new features align with the objectives set in the research. The analysis focuses on how well the solution fulfills the defined criteria, including accuracy, performance, reliability, and compliance with the constraints established in earlier sections.

Firstly, the implementation and evaluation adhere to the non-functional requirements, as outlined by the FURPS+ elements. The authentication, secure communication, and role-based

permissions were already implemented in the system and, therefore, the new functionalities had no major impact in these requirements.

Regarding reliability, error handling was a key factor, and the implementation complies with the basic handling error in the Apriori rules generation, due to the fact that making numerous custom error handling would affect the performance of the algorithm (Ribarić, 2024), which is the top priority, detailed in QFD in section 3.4. However, regarding the POI lists to include or exclude, the error handling was more customized and covered all alternatives flows that would open an exception, detailed in all test case scenarios in 4.2.3, 4.2.4 and **G1** and **G2** metrics assessment. Besides that, input validation was applied using the .NET built-in Data Annotations (Microsoft, 2022) and Model Validation (Microsoft, 2022) whenever the client required. This way, input data was reliable and reduced input errors.

Regarding other elements described in 3.4.2, all non-functional requirements were respected or implemented. Performance was measured and evaluated, with an ideal success achievement. In supportability, functional features comply with the correspondent acceptance criteria, documentation were produced to be a landscape to the new functionalities. Finally, all constraints were respected in the development process.

RQ1: How accurate is the data mining algorithm when integrated into a multi-agent system at generating rules between tourists with different preferences and traits?

The system's accuracy in predicting group preferences based on tourist traits was tested in terms on accuracy, also described and defined in **G3 - Provide Rules based on traits and attraction type preferences**. The algorithm accurately generated association rules between tourists with different preferences and traits. The results indicate a perfect prediction accuracy, as measured by successful group matching, which confirms the Apriori algorithm's reliability.

Since there is no need of communication between agents to perform the algorithm itself, the major advantage of Apriori integrated in a multi-agent system is the ability to store the generated rules locally within each MAS instance, detailed in section 4.1.1, without the need for a centralized database, since the rules will be overwritten every time the algorithm is executed. This allows each MAS instance to maintain its own set of rules, enabling different agents or instances to have unique, context-specific rules.

Also, the state of the art supports these findings, as prior research highlights the Apriori algorithm's effectiveness in generating association rules in multi-agent systems.

RQ2: What impact does the use of a data mining algorithm generating rules have on the personalization of POI lists for group members with different preferences and/or traits?

This question ties to **G1 - Provide POI based on group matching rules** and **G2 - Provide POI based on individual matching rules**. MAMS successfully generated POI inclusion/exclusion lists, considering individual characteristics and group traits. Through testing and evaluation of the metrics defined in the GQM, Apriori molds the rules that then can provide the respective POI to supply to the Recommendation Engine, whether by group preferences, or individual preferences.

Also, the algorithm complies with the time response limits predefined, which confirms the efficient performance of the system in generating POI lists to include/exclude. The ability to meet the predefined response time limits- demonstrates that the system can handle real-time requests effectively, providing timely and accurate POI recommendations based on both group and individual preferences.

The Apriori algorithm has a significant impact on the system's suggestions since it determines whether POI are included or excluded based only on the association rules it creates. Therefore, by correctly recognizing frequent patterns between group members' preferences and attributes, the algorithm guarantees that the personalized POI listings are matched with users' individual or collective interests.

Finally, the state of the art agrees that Apriori has been widely utilized to identify common patterns in a range of industries, including e-commerce and tourism, where consumer preferences are flexible and diverse. The algorithm's capacity to form rules based on user input is consistent with discoveries in literature, where Apriori has effectively established meaningful relationships to improve customization, in this case, to narrow POI.

RQ3: To what extent do variable thresholds of the data mining algorithm, if any, influence the accuracy of association rules in predicting group behavior?

Firstly, highly related to **G3 - Provide Rules based on traits and attraction type preferences**, the tests to confirm its metrics show that the minimum cluster threshold influences how many user tourists must have to generate personalized association rules. When the number of users in a cluster met or exceeded the defined threshold, the system generated rules specific to that cluster. Global rules were applied for all users' which cluster was below this threshold.

The same goes for the minimum confidence and minimum support thresholds. Described in section 4.2.2, all tests suggest that the higher these thresholds are, the more likely is it for the rules to be more precise and specific among the datasets and case tests scenarios.

RQ4: What are the accuracy limitations of the data mining algorithm when applied to larger datasets in a multi-agent tourism recommendation system?

Related to **G4 - Ensure scalability of the system for larger datasets with Apriori** in GQM, results revealed that the Apriori algorithm worked well up to the client expected dataset sizes. Beyond that, while rule generation accuracy remained intact, response times are about to exceed the 20 second threshold, indicating a performance bottleneck. This implies that, while the approach is scalable, it requires additional optimization for bigger datasets to maintain acceptable response times.

The state of the art confirms that Apriori is less successful with big datasets without optimization owing to computational complexity, which is consistent with the findings of this experiment. To overcome this, future work should focus on increasing the algorithm's efficiency.

6. Conclusion

This section presents the results obtained in dissertation project based on the objectives and research questions defined, followed by limitations and threats that compromise the validity of the solution proposed. Finally, future work is suggested and a final statement about the whole project is conducted.

6.1. Results

The section 1.3 delineated the objectives to be achieved and evaluated, and the outcome allowed to make conclusions, and the success rate of the goals defined. Regarding the first objective, the study of association rules mining algorithms revealed that Apriori was a suitable choice for the MAS system due to its simplicity and effectiveness in identifying frequent patterns in different traits and attraction type preferences.

The state of the art gave an in-depth understanding of the Apriori algorithm, its use in recommendation systems, and the integration of Multi-Agent Systems (MAS) and microservices. This fundamental understanding influenced the future stages. While the other variants had other set of advantages, Apriori's ease of implementation and versatility lead to a proposal solution in the current prototype.

About the second objective, the research into RS, namely GRS, confirmed the challenges posed by conflicting preferences among groups, in this case, of tourists. The integration of Apriori into the GRS prototype allowed to generate context-specific rules based on groups and individual preferences.

Regarding the third, the study of MAS and how can the association rules data mining algorithm would be integrated, the decentralized nature proved to have no negative impact on Apriori, since the algorithm doesn't need communication between agents to function. Also, the MAS system allows for the rules to be stored in their own context-specific set of agents with the respective rules, working as a container, avoiding the need for centralized databases, for example.

Finally, the last objective, the Apriori was successful in every metric evaluated when integrated into the MAMS. During the solution analysis and design phase, the prototype was

studied, and the requirements for integrating the Apriori algorithm were established. Detailed design documentation and architectural diagrams were generated to map out component interactions and ensure a clear implementation plan.

Extensive testing and result analysis validated that the algorithm in terms of accuracy and efficiency was consistent and reliable. Nonetheless, also shown in the results, the scalability limitations when applied to larger datasets exists and was close to reaching a bottleneck in the response time limits, despite are still valid for the client purposes, so it’s suggested optimization if the porotype intends to handle larger datasets than the ones simulated in section 5.3.4.

The results also revealed that the implementation solution effectively handled the research questions by giving accurate and specific POI suggestions for both individual users and groups based on Apriori generated rules. Variable thresholds influenced rule precision or behavior, and while bigger datasets presented scaling issues, the system reached reaction time and performance targets, completing the project's goals.

Finally, the project achieved all its primary objectives by effectively integrating the Apriori algorithm into a multi-agent recommendation system. The final evaluation emphasized the system's ability to deliver context-aware POI listings, confirming the project's overall success in meeting its objectives.

6.2. Limitations and Solution Validity Threats

Prior to implementation, a significant limitation resulted from a lack of understanding and expertise with Multi-Agent Systems (MAS). Also, one of the most significant restrictions was the difficulties of incorporating improved algorithms, such as the Apriori algorithm, into our existing system.

Related to possible threats to the implemented solution, despite the success of the project solution, there are some concerns to the Apriori implementation, such as the data quality and availability, since the Apriori relies heavily on the dataset. This can change the performance of the algorithm based on the sparsity of data, which may lead to performance degradation. Related to data quality, the scalability can reach a boiling point where larger datasets and number of

transactions accountable in the Apriori algorithm needs further evaluation. Therefore, continuous improvement on metrics is suggested to ensure performance is above the acceptable thresholds.

6.3. Future Work

One of the main strategies for future development is to investigate advanced algorithmic strategies and structures within the .NET framework to complement subparts of Apriori for association rule mining creation. Improving some sections of the Apriori algorithm can be game changer for improving efficiency in case of scalability issues.

Machine learning models, like collaborative filtering or neural networks, could complement the rule-based system to refine the association rules algorithm further. Lastly, since Grouplanner is hosted on Azure, extending the system with cloud-based solutions like Azure Functions could significantly improve the processing speed of association rule mining.

6.4. Final Statement

The strategic goal of improving POI suggestions aligns with the capabilities of the Apriori algorithm, which is well-known for its ability to find patterns and correlations within datasets. The study proposes a dynamic system that adjusts to the individual characteristics of each tourist by introducing Apriori into the GRS's existing multi-Agent microservice, ultimately elevating overall increasing the quality of the tourist experience. The investigation and integration of the Apriori algorithm into the prototype GRS for Tourism has provided useful insights into resolving the issues connected with group preferences and heterogeneous user aspects.

Finally, it sets the foundation of a multi-Agent system that uses the Apriori algorithm to refine group recommendations. The implementation has the potential to change the landscape of personalized tourism suggestions by providing a dynamic and adaptive solution that responds to the different preferences seen in group travel scenarios. The findings do not only help to improve the current GRS prototype, but also open the way for more effective and tailored recommendation systems for group travel.

7. References

Aflori, C., & Craus, M. (2007). Grid implementation of the Apriori algorithm. *Advances in Engineering Software*, *38*(5), 295–300. https://doi.org/10.1016/j.advengsoft.2006.08.011

Agrawal, R., & Srikant, R. (1994). Fast Algorithms for Mining Association Rules. *Proc. 20th Int. Conf. Very Large Data Bases, VLDB.*

Ait-Mlouk, A., Gharnati, F., & Agouti, T. (2016). Multi-agent-based modeling for extracting relevant association rules using a multi-criteria analysis approach. *Vietnam Journal of Computer Science*, *3*(4), 235–245. https://doi.org/10.1007/s40595-016-0070-4

Ait-Mlouk, A., Gharnati, F., & Agouti, T. (2017). An improved approach for association rule mining using a multi-criteria decision support system: a case study in road safety. *European Transport Research Review*, *9*(3). https://doi.org/10.1007/s12544-017-0257-5

Al-Maolegi, M., & Arkok, B. (2014). An Improved Apriori Algorithm For Association Rules. *International Journal on Natural Language Computing*, *3*(1), 21–29. https://doi.org/10.5121/ijnlc.2014.3103

Al-Qutaish, R. E., & Ain, A. (2010). Quality Models in Software Engineering Literature: An Analytical and Comparative Study. In *Journal of American Science* (Vol. 6, Issue 3). http://www.americanscience.orgeditor@americanscience.org166

Alshuqayran, N., Ali, N., & Evans, R. (2016). A Systematic Mapping Study in Microservice Architecture. *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. http://12factor.net/

Alves, P., Gomes, D., Rodrigues, C., Carneiro, J., Novais, P., & Marreiros, G. (2022). Grouplanner: A Group Recommender System for Tourism with Multi-agent MicroServices. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13616 LNAI*, 454–460. https://doi.org/10.1007/978-3-031-18192-4_37

Alves, P., J Carneiro, G Marreiros, & P Novais. (2019). Modeling a Mobile Group Recommender System for Tourism with Intelligent Agents and Gamification. *International Conference on Hybrid Artificial Intelligence Systems, 2019 Springer*.

Alves, P., Martins, A., Negrão, F., Novais, P., Almeida, A., & Marreiros, G. (2024). Are heterogeinity and conflicting preferences no longer a problem? Personality-based dynamic clustering for group recommender systems. *Expert Systems with Applications*, 124812. https://doi.org/10.1016/j.eswa.2024.124812

Alves, P., Martins, A., Novais, P., & Marreiros, G. (2023). Improving Group Recommendations using Personality, Dynamic Clustering and Multi-Agent MicroServices. *Proceedings of the 17th ACM Conference on Recommender Systems, RecSys 2023*, 1165–1168. https://doi.org/10.1145/3604915.3610653

Alves, P., Martins, H., Saraiva, P., Carneiro, J., Novais, P., & Marreiros, G. (2023). Group recommender systems for tourism: how does personality predict preferences for attractions, travel motivations, preferences and concerns? *User Modeling and User-Adapted Interaction*, *33*(5), 1141–1210. https://doi.org/10.1007/s11257-023-09361-2

Alzu'Bi, S., Hawashin, B., Eibes, M., & Al-Ayyoub, M. (2018). A Novel Recommender System Based on Apriori Algorithm for Requirements Engineering. *2018 5th International Conference on Social Networks Analysis, Management and Security, SNAMS 2018*, 323–327. https://doi.org/10.1109/SNAMS.2018.8554909

ATT. (2021). *Projeto PRR Agenda Acelerar e Transformar o Turismo*. Https://Transparencia.Gov.Pt/Pt/Fundos-Europeus/Prr/Beneficiarios-Projetos/Projeto/01/C05-I10/2024.PC645192610-00000060/#project_prr_form_id.

Baresi, L., & Garriga, M. (2019). Microservices: The evolution and extinction of web services? In *Microservices: Science and Engineering* (pp. 3–28). Springer International Publishing. https://doi.org/10.1007/978-3-030-31646-4_1

Barrientos, A. H., & Luevano, A. N. (2022). A State-of-the-Art Survey on Various Domains of Multi-Agent Systems and Machine Learning. *Multi-Agent Technologies and Machine Learning. IntechOpen.* www.intechopen.com

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). THE GOAL QUESTION METRIC APPROACH. In *Encyclopedia of software engineering (1994): 528-532.*

Belabed, I., Alaoui, M. T., Miloud, J. El, & Belabed, A. (2019). Association rules algorithms for data mining process based on multi agent system. *2nd International Conference on Machine Learning for Networking (MLN), Dec 2019, Paris, France. Pp.431-443, Ff10.1007/978-3-030- 45778-5_30ff. Ffhal-03266467*, 10. https://doi.org/10.1007/978-3-030

Boratto, L., Fenu, G., & Pau, P. L. (2015). Design Criteria to Model Groups in Big Data Scenarios: Algorithms and Best Practices. *CEUR WORKSHOP PROCEEDINGS (Vol. 1489, Pp. 8-16)*.

Boudoudouh, S., & Maâroufi, M. (2018). Multi agent system solution to microgrid implementation. *Sustainable Cities and Society*, *39*, 252–261. https://doi.org/10.1016/j.scs.2018.02.020

Brown, S. (2015). *Visualise, document and explore your software architecture Software Architecture for Developers-Volume 2*. Coding the Architecture, 63. http://leanpub.com/visualising-software-architecture

Cabral, B., & Marques, P. (2007). Exception Handling: A Field Study in Java and .NET. *ECOOP 2007–Object-Oriented Programming: 21st European Conference, Berlin, Germany, July 30-August 3, 2007. Proceedings 21 (Pp. 151-175). Springer Berlin Heidelberg.*

Chajri, M., & Fakir, M. (2016). Application of Data Mining in e-Commerce. In *Web Design and Development* (pp. 302–314). IGI Global. https://doi.org/10.4018/978-1-4666-8619-9.ch015

Collier, R. W., Lillis, D., O'Neill, E., & O'Hare, G. M. P. (2019). MAMS: Multi-agent microservices. *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*, 655–662. https://doi.org/10.1145/3308560.3316509

De Lucia, A., & Qusef, A. (2010). Requirements engineering in agile software development. *Journal of Emerging Technologies in Web Intelligence*, *2*(3), 212–220. https://doi.org/10.4304/jetwi.2.3.212-220

E da Fonseca Margato. (2020). Diário da República, 2.ª série PARTE E Artigo 2.º. *PARTE E (Doctoral Dissertation, Instituto Politécnico de Lisboa)*.

Dorri, A., Kanhere, S. S., & Jurdak, R. (2018). Multi-Agent Systems: A Survey. *IEEE Access*, *6*, 28573–28593. https://doi.org/10.1109/ACCESS.2018.2831228

Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2017, February 23). Microservices: How To Make Your Application Scale. *Perspectives of System Informatics: 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11 (Pp. 95-104). Springer International Publishing.* http://arxiv.org/abs/1702.07149

Fatoni, C. S., Utami, E., & Wibowo, F. W. (2018). Online Store Product Recommendation System Uses Apriori Method. *Journal of Physics: Conference Series*, *1140*(1). https://doi.org/10.1088/1742-6596/1140/1/012034

GECAD. (2024a). *GrouPlanner*. Https://Www.Gecad.Isep.Ipp.Pt/Grouplanner/.

GECAD. (2024b). *Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development*. Https://Www.Gecad.Isep.Ipp.Pt/.

Geraldi, J., & Lechter, T. (2012). Gantt charts revisited: A critical analysis of its roots and implications to the management of projects today. *International Journal of Managing Projects in Business*, *5*(4), 578–594. https://doi.org/10.1108/17538371211268889

IEEE Computer Society. Software Engineering Standards Committee., & IEEE-SA Standards Board. (1998). *IEEE recommended practice for software requirements specifications*. Institute of Electrical and Electronics Engineers.

Jabla, R., Khemaja, M., Buendia, F., & Faiz, S. (2022). Automatic Rule Generation for Decision-Making in Context-Aware Systems Using Machine Learning. *Computational Intelligence and Neuroscience*, *2022*. https://doi.org/10.1155/2022/5202537

Jahan, M. S., Azam, F., Anwar, M. W., Amjad, A., & Ayub, K. (2019). A Novel Approach for Software Requirement Prioritization. *Proceedings - 2019 7th International Conference in Software Engineering Research and Innovation, CONISOFT 2019*, 1–7. https://doi.org/10.1109/CONISOFT.2019.00012

Jánoky, L. V., Levendovszky, J., & Ekler, P. (2018). An analysis on the revoking mechanisms for JSON Web Tokens. *International Journal of Distributed Sensor Networks*, *14*(9). https://doi.org/10.1177/1550147718801535

Kagita, V. R., Singh, A., Kumar, V., Neerudu, P. K. R., Pujari, A. K., & Bondugula, R. K. (2023). Conformal Group Recommender System. *Elsevier ArXiv:2307.12034v1 [Cs.IR] 22 Jul 2023*. http://arxiv.org/abs/2307.12034

Kavitha, M. M., & Tamil Selvi, S. T. (2016). Comparative Study on Apriori Algorithm and Fp Growth Algorithm with Pros and Cons. *International Journal of Computer Science Trends and Technology (IJCS T)*, *4*. www.ijcstjournal.org

Kotsiantis, S., & Kanellopoulos, D. (2006). Association Rules Mining: A Recent Overview. *GESTS International Transactions on Computer Science and Engineering, 2006*.

Kraus, S., Breier, M., Lim, W. M., Dabić, M., Kumar, S., Kanbach, D., Mukherjee, D., Corvello, V., Piñeiro-Chousa, J., Liguori, E., Palacios-Marqués, D., Schiavone, F., Ferraris, A., Fernandes, C., & Ferreira, J. J. (2022). Literature reviews as independent studies: guidelines for academic practice. *Review of Managerial Science*, *16*(8), 2577–2595. https://doi.org/10.1007/s11846-022-00588-8

Kravari, K. (2018). A rule-based eCommerce methodology for the IoT using trustworthy Intelligent Agents and Microservices. *International Joint Conference on Rules and Reasoning (Pp. 302-309).*

Kravari, K., Kontopoulos, E., & Bassiliades, N. (2010). EMERALD: A Multi-Agent System for Knowledge-based Reasoning Interoperability in the Semantic Web. *Artificial Intelligence: Theories, Models and Applications: 6th Hellenic Conference on AI, SETN 2010, Athens, Greece, May 4-7, 2010*.

Leon, F. (2022). ActressMAS, a .NET Multi-Agent Framework Inspired by the Actor Model. *Mathematics*, *10*(3). https://doi.org/10.3390/math10030382

Li, Z., Li, X., Tang, R., & Zhang, L. (2021). Apriori Algorithm for the Data Mining of Global Cyberspace Security Issues for Human Participatory Based on Association Rules. *Frontiers in Psychology*, *11*. https://doi.org/10.3389/fpsyg.2020.582480

Lillis, D. (2020). Delivering Multi-Agent MicroServices using CArtAgO. *Engineering Multi-Agent Systems: 8th International Workshop, EMAS 2020*. https://microservices.io/patterns/apigateway.html

Mccarthy, J. F., & Ha~ost, T. D. (1998). MUSICFX: An Arbiter of Group Preferences for Computer Supported Collaborative Workouts. *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work. 1998.*

Mccarthy, K., Salamó, M., Coyle, L., Mcginty, L., Smyth, B., & Nixon, P. (2006). CATS: A Synchronous Approach to Collaborative Group Recommendation. *FLAIRS (Pp. 86-91)*. www.aaai.org

Meline, T. (2006). T Selecting Studies for Systematic Review: Inclusion and Exclusion Criteria. In *CONTEMPORARY ISSUES IN COMMUNICATION SCIENCE AND DISORDERS* • (Vol. 33). https://pubs.asha.org

Microsoft. (2022a). *Model validation in ASP.NET Core MVC and Razor Pages*. Https://Learn.Microsoft.Com/En-Us/Aspnet/Core/Mvc/Models/Validation?View=aspnetcore-8.0.

Microsoft. (2022b). *Validation with the Data Annotation Validators (C#)*. Https://Learn.Microsoft.Com/En-Us/Aspnet/Mvc/Overview/Older-Versions-1/Models-Data/Validation-with-the-Data-Annotation-Validators-Cs?Source=recommendations.

Microsoft. (2024). *.NET 8 Overview*. Https://Learn.Microsoft.Com/En-Us/Dotnet/Core/Whats-New/Dotnet-8/Overview.

Milind, S., Aher, D., & Aher, S. B. (2011). Data Mining in Educational System using WEKA. In *International Journal of Computer Applications®*. IJCA. https://www.researchgate.net/publication/266602921

Mutalib, S., Azri, A., Subar, A., Abdul-Rahman, S., & Mohamed, A. (2014). Comparative Study of Apriori-variant Algorithms. *Knowledge Management International Conference (KMICe) 2016, 29 – 30 August 2016, Chiang Mai, Thailand*. http://www.kmice.cms.net.my/203

Ordem dos Engenheiros. (2016). *Ordem dos Engenheiros Código de Ética e Deontologia*.

Paricio García, A., Oliver, J., & Gosch, D. (2010). An Intelligent Agent-Based Distributed Architecture for Smart-Grid Integrated Network Management. *2010 IEEE 35th Conference on Local Computer Networks.*

Paul, J., & Criado, A. R. (2020). The art of writing literature review: What do we know and what do we need to know? *International Business Review*, *29*(4). https://doi.org/10.1016/j.ibusrev.2020.101717

Pnnz, W., Jarke, M., Rogers, Y., Schmidt, K., Wulf, V., O'connor, M., Cosley, D., Konstan, J. A., & Riedl, J. (2001). PolyLens: A Recommender System for Groups of Users. *ECSCW 2001: Proceedings of the Seventh European Conference on Computer Supported Cooperative Work 16–20 September 2001, Bonn, Germany. Springer Netherlands.*, 199–218.

Qasem, M. H., Hudaib, A., Obeid, N., Almaiah, M. A., Almomani, O., & Al-Khasawneh, A. (2022). Multi-agent Systems for Distributed Data Mining Techniques: An Overview. In *Studies in Computational Intelligence* (Vol. 994, pp. 57–92). Springer Science and Business Media Deutschland GmbH. https://doi.org/10.1007/978-3-030-87954-9_3

Raharjana, I. K., Siahaan, D., & Fatichah, C. (2021). User Stories and Natural Language Processing: A Systematic Literature Review. *IEEE Access*, *9*, 53811–53826. https://doi.org/10.1109/ACCESS.2021.3070606

Ramar, A., & Elamparithi, M. (2016). Extensible Multi Agent System for Heterogeneous Database: Knowledge Discovery in Database, Multi Agent Systems, Extendible Multi Agent Data mining System. *IJCSET(Www.Ijcset.Net) | June 2016 | Vol 6, Issue 6, 260-266*. www.ijcset.net

Ribarić, K. (2024). *.NET Error Handling: Balancing Exceptions and the Result Pattern*. Https://Dev.to/K_ribaric/Net-Error-Handling-Balancing-Exceptions-and-the-Result-Pattern-Ljo.

Rosenberg, D., & Stephens, M. (2019). *Use case driven object modeling with UML: theory and practice*. Scholars Portal.

Rosenberg, Doug., & Stephens, Matt. (2007). *Use case driven object modeling with UML : theory and practice*. Apress.

Samadhiya, D., & Wang, S.-H. (2010). Quality Models: Role and Value in Software Engineering. *2010 2nd International Conference on Software Technology and Engineering. Vol. 1. IEEE, 2010.*

Seela, S. (2023, June 27). *Apriori Algorithm – Frequent Pattern Algorithms*. Https://Www.Softwaretestinghelp.Com/Apriori-Algorithm/#Apriori_Algorithm_8211_Frequent_Pattern_Algorithms. https://www.softwaretestinghelp.com/apriori-algorithm/#Apriori_Algorithm_8211_Frequent_Pattern_Algorithms

Sharma, M., & Mann, S. (2013). A Survey of Recommender Systems: Approaches and Limitations. *International Journal of Innovations in Engineering and Technology Special Issue-ICAECE*.

Sidhu, S., Meena, U. K., Nawani, A., Gupta, H., & Thakur, N. (2014). FP Growth Algorithm Implementation. In *International Journal of Computer Applications* (Vol. 93, Issue 8).

Singh Talwar, K., Oraganti, A., Mahajan, N., & Narsale, P. (2015). Recommendation System using Apriori Algorithm. In *IJSRD-International Journal for Scientific Research & Development*| (Vol. 3). http://www.win.tue.nl/~laroyo/2L340/resources/rec

Soto, C. J., & Jackson, J. J. (2013). Five-Factor Model of Personality. In *Psychology*. Oxford University Press. https://doi.org/10.1093/obo/9780199828340-0120

Srinadh, V. (2022). Evaluation of Apriori, FP growth and Eclat association rule mining algorithms. *International Journal of Health Sciences*, 7475–7485. https://doi.org/10.53730/ijhs.v6ns2.6729

Sruthy. (2024). *Apriori Algorithm*. Https://Www.Softwaretestinghelp.Com/Apriori-Algorithm/.

Vázquez-Ingelmo, A., García-Holgado, A., & García-Peñalvo, F. J. (2020). C4 model in a Software Engineering subject to ease the comprehension of UML and the software development process. *2020 IEEE Global Engineering Education Conference (EDUCON) (Pp. 919-924). IEEE.* https://structurizr.com/

Warkentin, M., Sugumaran, V., & Sainsbury, R. (2012). The role of intelligent agents and data mining in electronic partnership management. *Expert Systems with Applications*, *39*(18), 13277–13288. https://doi.org/10.1016/j.eswa.2012.05.074

Williams, C. (2007). Research Methods. In *Journal of Business & Economic Research-March* (Vol. 5).

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John wiley & sons.

Wu, H., Lu, Z., Pan, L., Xu, R., & Jiang, W. (2009). An improved Apriori-based algorithm for association rules mining. *6th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2009*, *2*, 51–55. https://doi.org/10.1109/FSKD.2009.193

Yabing, J. (2016). Research of an Improved Apriori Algorithm in Data Mining Association Rules. *International Journal of Computer and Communication Engineering, 2013*.

Yu, X., & Wang, H. (2014). Improvement of Eclat Algorithm Based on Support in Frequent Itemset Mining. *Journal of Computers*, *9*(9). https://doi.org/10.4304/jcp.9.9.2116-2123

Yuan, X. (2017). An improved Apriori algorithm for mining association rules. *AIP Conference Proceedings*, *1820*. https://doi.org/10.1063/1.4977361

Zare Mehrjerdi, Y. (2010). Quality function deployment and its extensions. In *International Journal of Quality & Reliability Management* (Vol. 27, Issue 6, pp. 616–640). https://doi.org/10.1108/02656711011054524