UNIVERSITY OF MILANO-BICOCCA

Department of Computer Science, Systems and Communication

Master's Degree Programme in **Computer Science**

Innovative Management of Service Requests: System Design, Automation, and Reccomendation through RAG and Fine-Tuned LLMs

Advisor: Prof. Leonardo Mariani

Master's Thesis by:
Nicolas Guarini
Student ID: 918670

Academic Year 2024–2025

Lorem ipsum dolor sit amet.

Contents

Chapter 0

Introduction

The technological evolution of recent decades has profoundly transformed the business landscape, making IT solutions a crucial factor for the success of enterprises. In this context, Elmec Informatica S.p.A. stands out as a significant player in the Information Technology sector in Italy. Founded in 1971 and headquartered in Varese, Elmec combines extensive industry experience with a strong focus on innovation, offering services and solutions that cover a wide range of business needs, offering advanced IT solutions for medium and large enterprises.

The company manages four datacenters (including one Tier IV certified), provides Hybrid IT services, digital workspace and Device-as-a-Service solutions, and is a key partner for cybersecurity and regulatory compliance. With over 450 certified technicians, Elmec integrates cloud technologies (in collaboration with AWS and GAIA-X), promotes environmental sustainability, and stands out for its innovation through its Technological Campus and ongoing investments in professional training.

The primary context of this internship revolves around the redesign and enhancement of the existing *Service Request Portal (SRP)*, which is essential for managing client service requests efficiently.

A Service Requests is a formal request submitted by a customer to obtain specific services or actions within their IT infrastructure. These requests can encompass a wide range of activities, from creating new user accounts in the Active Directory system to managing user permissions, resetting passwords, and deploying software updates.

Various challenges have been identified within the current architecture, particularly concerning the dynamic forms used by customers for submitting the service requests, the management of customizable fields, and the integration with external data sources.

The main objectives of this internship are:

- **Analysis of the Existing System**: Conduct a comprehensive analysis of the current state of the SRP system, identifying weaknesses and inefficiencies that hinder its performance and user experience;

- **Architectural redesign**: Propose and implement a complete architectural redesign of the system;

- **Modular and Scalable Solution**: Design a modular and scalable solution that can accomodate various client-specific configurations;
- **Migrations**: Plan and execute a full migration of databases and other company services/platforms that use SRP.
- **Fine-tuned LLMs integration**: Explore opportunities for integrating specifically fine-tuned LLM systems into the SR workflow.

Chapter 1

Overview of the existing System

The internship project was not focused on creating an entirely new system from scratch, but rather on a deep redesign and restructuring of an existing system developed many years ago using legacy technologies and affected by several issues and critical limitations.

Before discussing the architectural, technological, and implementation choices that were made, it is necessary to carry out a thorough analysis of the current system, in order to understand its behavior, operational flows, technologies used, the main issues to be addressed, how to resolve them, how to prevent them from reoccurring in the future, and how to ensure that the new system is scalable and maintainable.

1.1 Current System Architecture

The *SRP* system is fundamentally divided into two primary segments:

- Company Network;
- Client Network.

These segments interact and communicate exclusively through a *DMI Console*, which acts as a unidirectional bridge, transferring requests from the client network to the internal company network.

The overall architecture is depicted in the diagram in figure 1.1.

The system's operational flow involves several key components across both networks, orchestrating the processing of service requests.

1.1.1 Main Functionalities and Components

The current *SRP* system relies on a suite of interconnected components, each serving a specific role in the lifecycle of a service request.

Elmec Network Components:

- **SRP Web Sevice REST**: This component, deployed on a Kubernetes cluster, exposes .NET Core RESTful services. It serves as the primary interface for external systems such as:

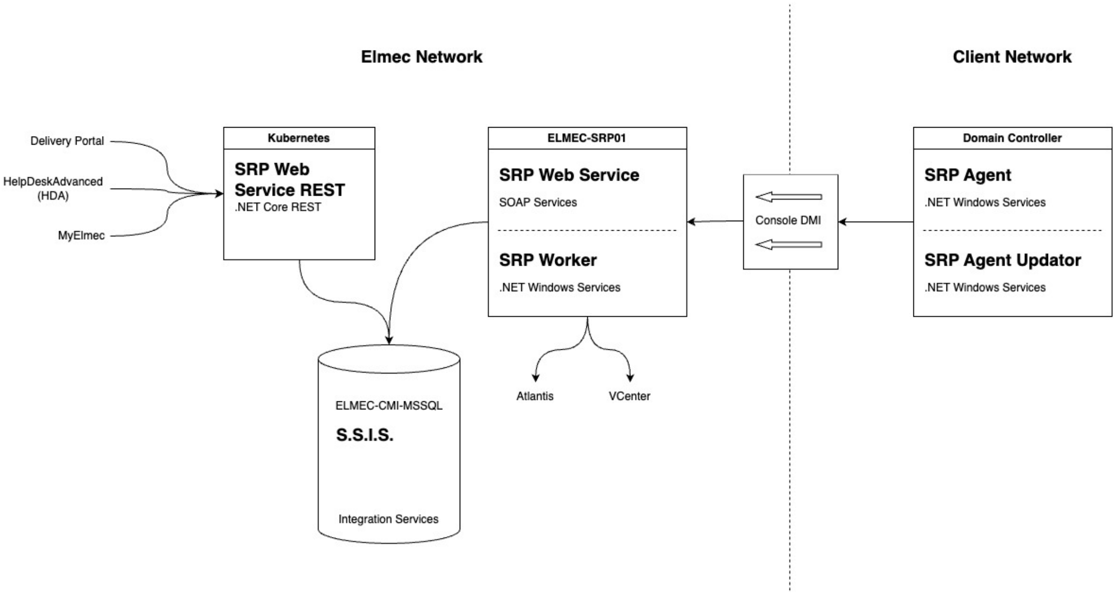

Figure 1.1: AS-IS System Architecture

- **Delivery Portal**: An asset management system used to manage Service Request catalog configuration and Elmec's internal and client servers;
- **HDA (HelpDeskAdvanced)**: A ticket management system. Each service request is intrinsically linked to an HDA ticket via an ID, and HDA is responsible for initiating and associating service requests with existing tickets.
- **MyElmec**: A client-facing portal enabling customers to create, manage, and monitor the various services provided by Elmec (e.g., servers, management software, cloud solutions).

The *SRP Web Service REST* communicates with the *ELMEC-CMI-MSSQL* server.

- **ELMEC-CMI-MSSQL**: This server hosts the central SQL Server Database for the SRP system. It is the persistent storage layer for all service request data and related information.
- **S.S.I.S. (SQL Server Integration Services)**: Running on the ELMEC-CMI-MSSQL server, SSIS packages are crucial for data transformation and import/export operations. They facilitate the transfer of data, such as master data (e.g., Active Directory users, Organizational Units, groups) sent by SRP Agents, into the database. Each SSIS package is designed to transform the received information for database insertion or vice versa.
- **ELMEC-SRP01**: This Windows Server functions as the core engine for service requests within the Elmec network. It houses all the PowerShell scripts associated with individual service requests, which are maintained by the Platform team. Key components residing on ELMEC-SRP01 in-

clude:

  - **SRP WebServiceSoap**: This component provides SOAP services primarily for communication with the SRP Agent located on the client side. It interacts with the database to retrieve necessary information before exposing it to the agents for service request execution.
  - **SRP InternalWorker**: A Windows service that operates similarly to the SRP Agent but executes entirely within the Elmec network. This is utilized for service requests that do not require interaction with the client's network or Active Directory (e.g., "no patch" service requests). The InternalWorker queries the database every 10 seconds for "internal worker" type jobs (like removing a server from a patching session, or resetting a managed Virtual Machine), connects to the SOAP service, retrieves the Service Request ID and its fields, and initiates job execution. It also monitors logs every 30 seconds to determine the job's status.

The only component in the client's network is called **Domain Controller**, which represents the client's Active Directory domain controller, where the communication tools with Elmec are installed. It hosts:

- **SRP Agent (.NET Windows Service)**: Typically, one agent is configured per client domain, specifically for Active Directory (AD) and Exchange-related service requests. This agent performs two main functions:
  - Every 8 hours, it reads master data (AD users, OUs, groups) from the domain controller and sends it to ELMEC-SRP01 for storage via SSIS.
  - Every 5 minutes, it queries ELMEC-SRP01 for pending Service Requests. Upon finding any, it requests detailed information, loads the corresponding PowerShell script, and executes it. After initiating the script, the agent informs ELMEC-SRP01 that the SR is "running" and proceeds to the next SR.
- **SRP Agent Updator (.NET Windows Service)**: This service continuously monitors the SRP Agent folder for a file with a .new extension. If detected, it stops the SRP Agent, updates it, and restarts it

1.1.2 Interactions and Communication between Customers and Company networks

The Communication between the company network and the clients' networks is a critical aspect of the SRP system's operations and is handled through the **DMI Console**.

The DMI console serves as a key infrastructural component, primarily acting as a communication bridge between the customer's network and Elmec's internal network. Specifically, for the SRP system, the DMI console is responsible for transferring service requests (SR) from the customer's network to Elmec's internal network, although it does not handle traffic in the opposite direction.

Operation of the DMI Console

The connectivity of the DMI appliances to Elmec is established through multiple OpenVPN connections, initiated by the appliance towards the Brunello 4 and Mendrisio Data Centers (for Swiss customers). The DMI consoles are not directly exposed to the Internet and are accessible only by authorized personnel. Hardware requirements for a DMI node include 4 GB of RAM, 4 cores, and 50 GB of disk space.

Following recent developments in the DMI, the new consoles use K3S for container management. K3S is a lightweight, certified Kubernetes distribution designed for edge, IoT, and CI/CD environments. It is characterized by being a single binary that reduces external dependencies and uses SQLite as the default database, greatly simplifying installation and management. It already includes essential components such as *containerd* [3].

All consoles are centrally managed via Rancher, and the deployment of the application stack is delegated to ArgoCD, which ensures that container versions are constantly updated. For security reasons, the console in the customer network by default exposes only SSH, while services such as SRP are selectively enabled and only when necessary. All application data present on the consoles is stored on a LUKS (Linux Unified Key Setup) encrypted volume[1], which can only be unlocked if the VPN tunnel is established. The operating system update is performed through a standard Elmec system, called PMD (Patching Management Dashboard).

1.2 Service Requests Workflow

The lifecycle of a Service Request within the current system is orchestrated through a series of defined states and automated processes, involving interactions between various components and human operators. The flow is designed to manage SRs from initiation to completion, handling approvals, execution,

[1] A platform-independent hard disk encryption method that can be used to encrypt disks across a wide variety of tools. This not only ensures full compatibility and interoperability between different software but also guarantees that password management occurs in a secure and documented manner [1].

and potential errors.

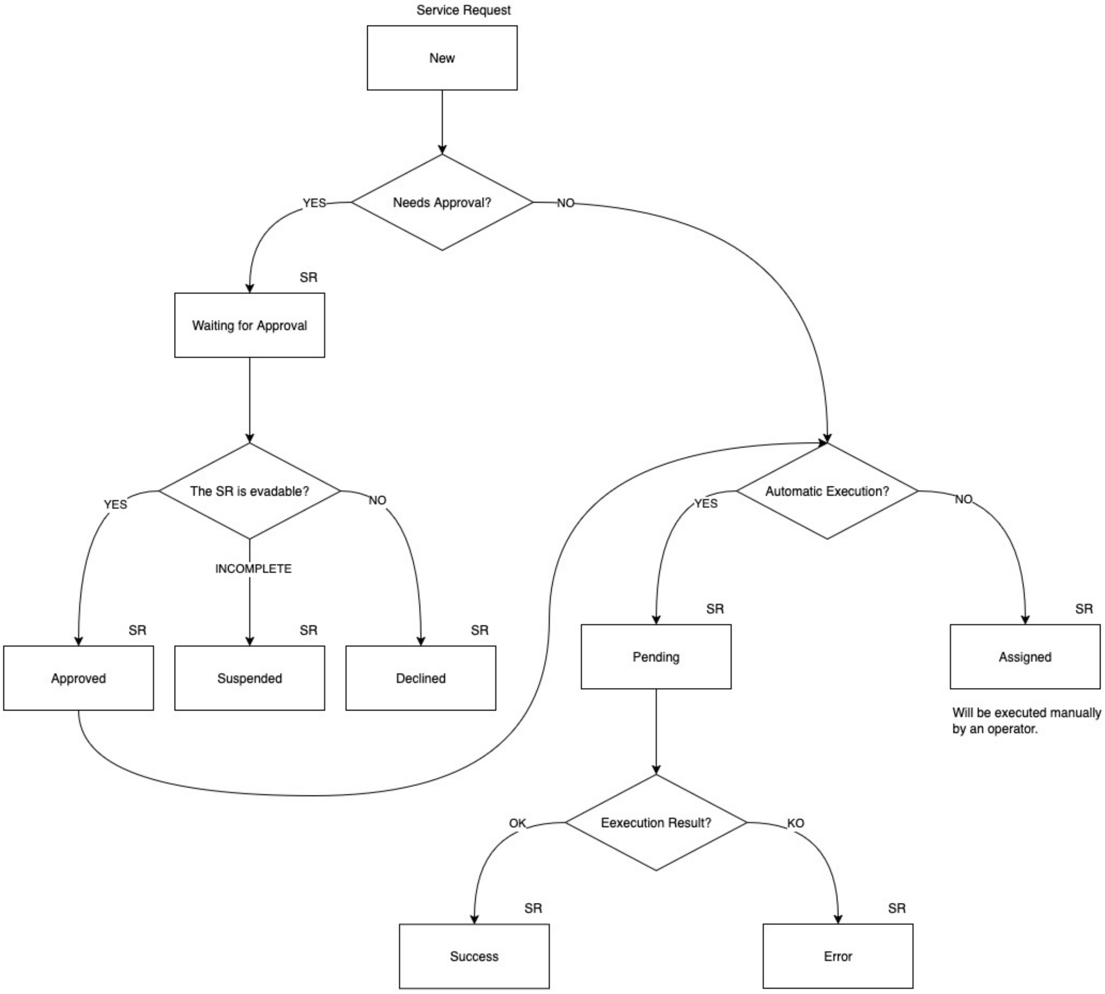

Figure 1.2: High-Level Service Request Lifecycle

In Figure 1.2 a simple and high-level flowchart is shown that illustrates the main flows and scenarios. Instead, in Figure 1.3 a complete and detailed flowchart is shown that illustrates all the details about every flow, including information on the jobs and components that manage the various sections of an SR's workflow.

1.2.1 SR Initiation and Initial State Management

A Service Request can be initiated through two primary channels:

- **Via MyElmec Portal**: Clients can submit an SR by completing a dedicated form from the MyElmec portal. Upon saving the form, a ticket is created in the HelpdeskAdvanced (HDA) system with a `QUALIFIED` status, and a corresponding Service Request is generated in SRP with a `NEW` status. These two entities are then linked via the HDA Ticket ID;
- **Via HDA Ticket by Elmec Operator**: Operators have the ability to associate a catalog SR directly with a ticket they are managing within the

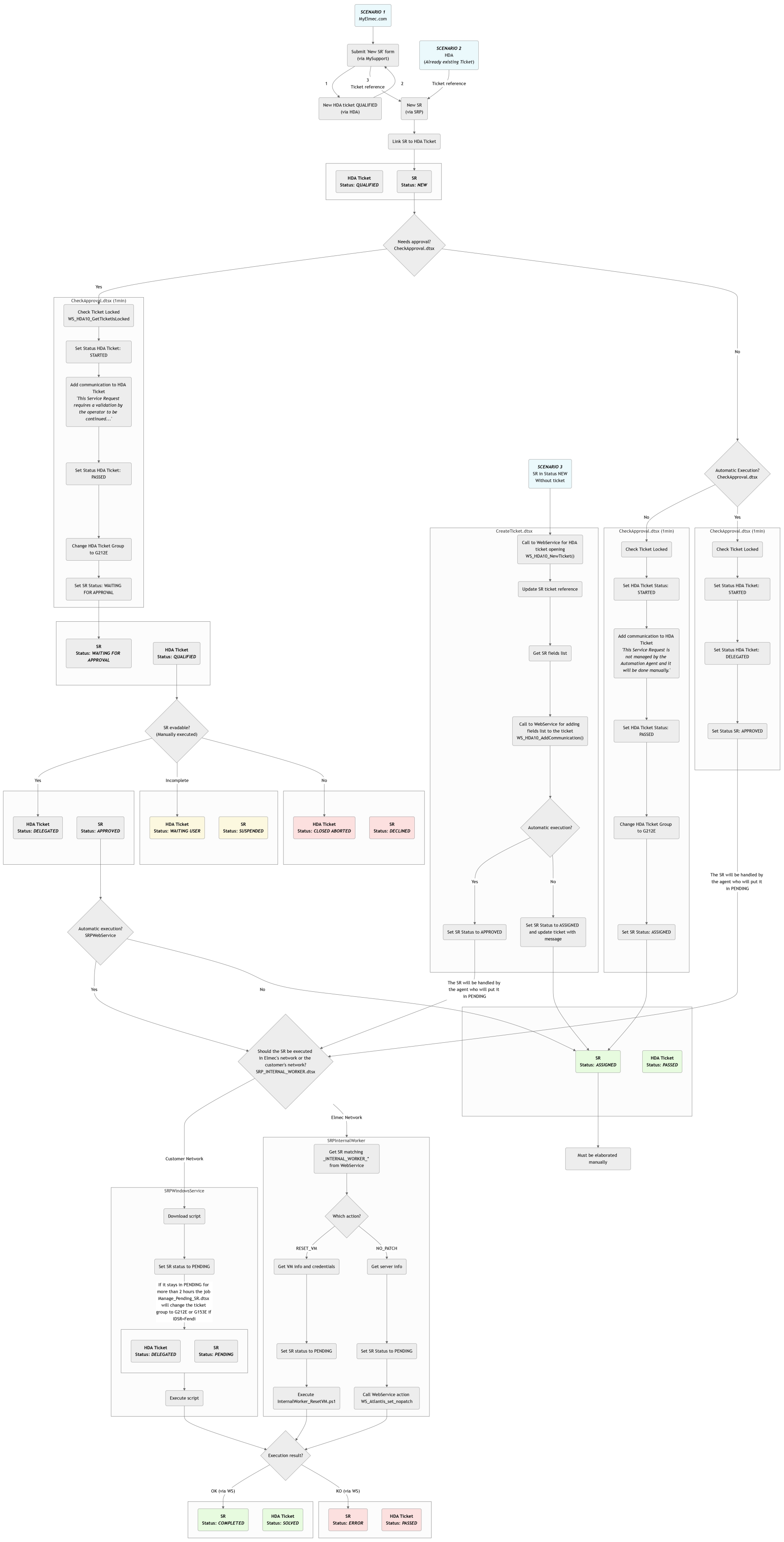

Figure 1.3: AS-IS Complete Service Request Execution Workflow

HDA interface. When the ticket is saved in a `DELEGATED` status, the SR is created in SRP with a `NEW` status, linked to the ticket, and immediately triggers the SR workflow within SRP.l

Following its creation, the system evaluates whether the Service Request requires approval by the operators. The logic for setting the initial SR and HDA ticket states is as follows:

- **If approval is required**: the Service Request is set to `WAITING FOR APPROVAL` status, and the corresponding HDA ticket is set to `QUALIFIED`. A communication is added to the ticket, indicating that the SR needs validation from an operator before execution;
- **If no approval is required and it's linked to an automatism**: the Service Request enters a `PENDING` state, and the HDA ticket is set to `DELEGATED`;
- **If no approval is required and it's not linked to an automatism**: the Service Request is `ASSIGNED`, and the HDA ticket is set to `PASSED`. A communication is added to the ticket, noting that the SR requires manual processing by an operator.

1.2.2 Approval and Execution Workflows

For SRs requiring approval (`WAITING FOR APPROVAL` state with an HDA `QUALIFIED` ticket), an operator's intervention is necessary to validate the request. The operator has three possible actions:

- **Cancel the SR**: the HDA ticket is set to `CLOSED ABORTED`, the SR status becomes `DECLINED`, and a notification email is sent to the requesting client;
- **Request client modification**: the HDA ticket is set to `WAITING USER`, the SR status becomes `SUSPENDED`, and a notification email is sent to the client. The workflow pauses until the client modifies the SR via Elmec.com;
- **Approve the SR**: the HDA ticket is set to `DELEGATED`, and the SR status becomes `APPROVED`.

Once a SR is `APPROVED`, SRP manages its subsequent flow based on whether it is linked to an automatism:

- **If the SR is linked to an automatism**: the SR transitions to a `PENDING` state, while the HDA ticket remains `DELEGATED`. This implies it's awaiting automated execution;
- **If the SR is not linked to an automatism**: the SR is immediately set to `ASSIGNED`, and the HDA ticket is moved to `PASSED`. A note is added to the ticket indicating that the SR requires manual processing.

An SR in `PENDING` state is awaiting the execution of its associated automatism. The outcome of this automated execution determines the next state:

- **Successful execution**: the SR is set to `COMPLETED`, and the corresponding HDA ticket is set to `SOLVED`;
- **Unsuccessful execution**: the SR transitions to an `ERROR` state, and the HDA ticket is set to `PASSED`. Communication regarding the error is added to the ticket for visibility within HDA.

The execution of Service Requests themselves can involve the SRP Agent on the client's Domain Controller for operations requiring client-side interaction (e.g., Active Directory or Exchange environment changes). The SRP Agent periodically checks `ELMEC-SRP01` for pending SRs. If found, it retrieves the SR details, loads the relevant PowerShell script, and executes it. The agent then informs ELMEC-SRP01 that the SR is running. Monitoring is performed by a scheduled READLOG task on the client side, which checks the PowerShell script's log file every minute. A "KEY" in the log indicates completion, prompting the SRP Agent to inform ELMEC-SRP01 to set the SR to COMPLETED, triggering an HDA status update. An "ERROR" key indicates failure, leading to the log being sent to ELMEC-SRP01 and an error signal. A timeout mechanism also signals an error if completion is not detected within a specified duration

1.3 Problems and core issues of the current system

The analysis of the architecture and workflow of the current SRP system has revealed a number of critical issues that compromise its scalability, maintainability, and reliability. These problems are deeply rooted in the technological and design choices made years ago and are evident across several key areas of the system, from catalog management to script execution and the import of master data from Active Directory.

1.3.1 Architectural and Technological Challenges

The SRP system is characterized by a fragmented architectural structure and complex interdependencies between its main components, which leads to several critical issues.

Monolithic Frontend Component

The user interface, responsible for navigating the Service Request catalog and completing the associated forms, is an excessively large and complex Vue.js frontend component (nearly 11,000 lines). This architecture makes modifications, extensions, and client-specific customizations exceptionally challenging.

The logic for managing dynamic fields, their interdependencies, and the rules for compilation and validation are all handled at the frontend level through a long series of hard-coded conditional statements, such as the following:

```
<div v-if="field.desc == 'Domain'">
    ...
</div>
```

This, in addition to mixing business logic with presentation, represents a significant challenge for catalog configuration, as there is no centralized way to manage the fields and their relationships. Every change, therefore, requires a direct intervention on the code, making the system inflexible and difficult to maintain, especially in a context where the operator modifying the catalog is not a programmer and does not know how the frontend code is structured. For instance, if an operator wanted to add a new "Domain" field to a Service Request, this field would need to have the exact description "Domain" and type "domain"; otherwise, the Vue.js component would not render it correctly.

These dynamics make the system fragile and susceptible to frequent errors that waste time and resources for both clients and operators.

Overly Generic Backend REST Service

The only information about the fields returned by the backend REST service (`SRPWebSeviceREST`), besides the field name and its mandatoriness, is its *type*. It is therefore easy to imagine how this information, which should only be an indication of the component that needs to be rendered (e.g., textbox, select, radio button, etc.), can become an identifier for something much broader.

For example, a field of type `UPN` is a composite field derived from the value of another field, '`Display Name`', concatenated with an '`@`' symbol, and suffixed with the value from a select field whose options are the client's domains, provided by an external REST service (e.g., "`n.guarini@elmec.it`", where "`n.guarini`" is the value of the "`Display Name`" field, and "`elmec.it`" is the value of the "`Domain`" select field).

Despite this complexity, the REST service returns only a JSON document of this type:

```
{
    "idField": 1603,
    "descField": "UPN",
    "type": "upn",
    "mandatory": true
}
```

thus delegating to the frontend the responsibility of interpreting the type and rendering the field correctly.

Dependency on the SRP Agent

The import of data from clients' Active Directories and the actual execution of Service Requests are handled by an agent (`SRPWindowsService`) installed directly on the clients' Domain Controllers. This approach presents several challenges:

- **Security and Permissions**: The agent often requires credentials with elevated privileges (e.g., `domain-admin`), which clients are increasingly reluctant to grant.
- **Reliability and Control**: The configurations on the agent and the Domain Controller are complex and difficult to manage centrally, which can cause execution errors and data synchronization issues.
- **Scalability**: The need to install, configure, and maintain an agent for each client domain represents a significant operational effort. Alternatives such as more modern provisioning systems will be evaluated.
- **Obsolete Communication Patterns**: The unidirectional communication through the *DMI Console* creates a bottleneck and complicates the management of asynchronous operations. Communication between components in the client's network and the company network is based on polling (e.g., the SRP Agent queries the server every 5 minutes), an approach that introduces latency and inefficiencies.

1.3.2 Code and Data Management Issues

Code and data persistence management is another critical aspect of the system, characterized by a lack of versioning and modern debugging tools.

Unversioned and Undebuggable Stored Procedures

The intensive use of Stored Procedures[2] in the MSSQL database represents one of the major weaknesses. These procedures contain fundamental business logic and are:

- **Unversioned**: There is no centralized version control system for Stored Procedures. Changes are made directly on the database, making it impossible to track changes, revert to previous versions, or test modifications in separate environments.
- **Difficult to Debug**: Debugging Stored Procedures is a complex and

[2] Stored procedures are SQL code routines, or sets of instructions, that are pre-compiled and stored within a relational database. They act as logical units of execution, encapsulating business logic directly at the database level [5].

cumbersome process, often requiring direct access and specific SQL server tools, with the risk of affecting the production environment.

Inaccessible and Unversioned SSIS Jobs

SSIS packages[3] are essential for data import and export operations (e.g., AD master data). However, these jobs reside physically on the `ELMEC-CMI-MSSQL` server, and the only way to access and modify them is via a remote desktop. This approach prevents versioning and collaboration, limits accessibility by making maintenance and debugging an onerous and risky activity, and creates a *single-point-of-failure* and a strong coupling between the import/export business logic and the infrastructure.

1.3.3 Database Schema and Data Integrity Issues

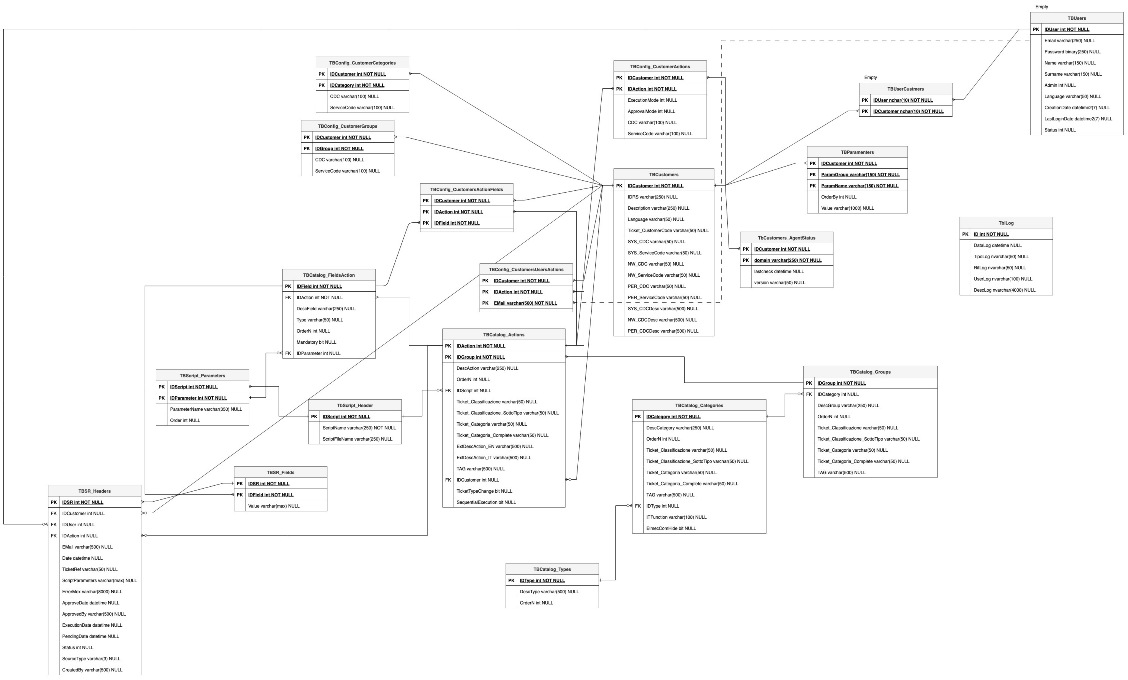
Empty
TBUsers
PK IDUser int NOT NULL
Email varchar(250) NULL
Password binary(250) NULL
Name varchar(150) NULL
Surname varchar(150) NULL
Admin int NULL
Language varchar(50) NULL
CreationDate datetime2(7) NULL
LastLoginDate datetime2(7) NULL
Status int NULL
TBConfig_CustomerCategories
PK IDCustomer int NOT NULL
PK IDCategory int NOT NULL
CDC varchar(100) NULL
ServiceCode varchar(100) NULL
TBConfig_CustomerActions
PK IDCustomer int NOT NULL
PK IDAction int NOT NULL
ExecutionMode int NULL
ApprovalMode int NULL
CDC varchar(100) NULL
ServiceCode varchar(100) NULL
Empty
TBUserCustmers
PK IDUser nchar(10) NOT NULL
PK IDCustomer nchar(10) NOT NULL
TBConfig_CustomerGroups
PK IDCustomer int NOT NULL
PK IDGroup int NOT NULL
CDC varchar(100) NULL
ServiceCode varchar(100) NULL
TBParamenters
PK IDCustomer int NOT NULL
PK ParamGroup varchar(150) NOT NULL
PK ParamName varchar(150) NOT NULL
OrderBy int NULL
Value varchar(1000) NULL
TBConfig_CustomersActionFields
PK IDCustomer int NOT NULL
PK IDAction int NOT NULL
PK IDField int NOT NULL
TBCustomers
PK IDCustomer int NOT NULL
IDRS varchar(250) NULL
Description varchar(250) NULL
Language varchar(50) NULL
Ticket_CustomerCode varchar(50) NULL
SYS_CDC varchar(50) NULL
SYS_ServiceCode varchar(50) NULL
NW_CDC varchar(50) NULL
NW_ServiceCode varchar(50) NULL
PER_CDC varchar(50) NULL
PER_ServiceCode varchar(50) NULL
SYS_CDCDesc varchar(500) NULL
NW_CDCDesc varchar(500) NULL
PER_CDCDesc varchar(500) NULL
TblLog
PK ID int NOT NULL
DataLog datetime NULL
TipoLog nvarchar(50) NULL
RifLog nvarchar(50) NULL
UserLog nvarchar(100) NULL
DescLog nvarchar(4000) NULL
TbCustomers_AgentStatus
PK IDCustomer int NOT NULL
PK domain varchar(250) NOT NULL
lastcheck datetime NULL
version varchar(50) NULL
TBCatalog_FieldsAction
PK IDField int NOT NULL
FK IDAction int NOT NULL
DescField varchar(250) NULL
Type varchar(50) NULL
OrderN int NULL
Mandatory bit NULL
FK IDParameter int NULL
TBConfig_CustomersUsersActions
PK IDCustomer int NOT NULL
PK IDAction int NOT NULL
PK EMail varchar(500) NOT NULL
TBCatalog_Actions
PK IDAction int NOT NULL
PK IDGroup int NOT NULL
DescAction varchar(250) NULL
OrderN int NULL
FK IDScript int NULL
Ticket_Classificazione varchar(50) NULL
Ticket_Classificazione_SottoTipo varchar(50) NULL
Ticket_Categoria varchar(50) NULL
Ticket_Categoria_Complete varchar(50) NULL
ExtDescAction_EN varchar(500) NULL
ExtDescAction_IT varchar(500) NULL
TAG varchar(500) NULL
FK IDCustomer int NULL
TicketTypeChange bit NULL
SequentialExecution bit NULL
TBScript_Parameters
PK IDScript int NOT NULL
PK IDParameter int NOT NULL
ParameterName varchar(350) NULL
Order int NULL
TbScript_Header
PK IDScript int NOT NULL
ScriptName varchar(250) NOT NULL
ScriptFileName varchar(250) NULL
TBCatalog_Groups
PK IDGroup int NOT NULL
FK IDCategory int NULL
DescGroup varchar(250) NULL
OrderN int NULL
Ticket_Classificazione varchar(50) NULL
Ticket_Classificazione_SottoTipo varchar(50) NULL
Ticket_Categoria varchar(50) NULL
Ticket_Categoria_Complete varchar(50) NULL
TAG varchar(500) NULL
TBCatalog_Categories
PK IDCategory int NOT NULL
DescCategory varchar(250) NULL
OrderN int NULL
Ticket_Classificazione varchar(50) NULL
Ticket_Classificazione_SottoTipo varchar(50) NULL
Ticket_Categoria varchar(50) NULL
Ticket_Categoria_Complete varchar(50) NULL
TAG varchar(500) NULL
FK IDType int NULL
ITFunction varchar(100) NULL
ElmecComHide bit NULL
TBSR_Fields
PK IDSR int NOT NULL
PK IDField int NOT NULL
Value varchar(max) NULL
TBSR_Headers
PK IDSR int NOT NULL
FK IDCustomer int NULL
FK IDUser int NULL
FK IDAction int NULL
EMail varchar(500) NULL
Date datetime NULL
TicketRef varchar(50) NULL
ScriptParameters varchar(max) NULL
ErrorMex varchar(8000) NULL
ApproveDate datetime NULL
ApprovedBy varchar(500) NULL
ExecutionDate datetime NULL
PendingDate datetime NULL
Status int NULL
SourceType varchar(3) NULL
CreatedBy varchar(500) NULL
TBCatalog_Types
PK IDType int NOT NULL
DescType varchar(500) NULL
OrderN int NULL

Figure 1.4: AS-IS Database ER Schema

The analysis of the database schema, represented in figure 1.4, has revealed a number of significant issues that affect the quality and integrity of the data managed by the SRP system.

[3] SQL Server Integration Services (SSIS) is a platform for building data integration and workflow solutions. It is a key component of Microsoft SQL Server, primarily used for Extract, Transform, Load (ETL) operations, enabling the extraction of data from various sources, its transformation, and its loading into different destinations [4].

Lack of Data Integrity and Consistency

One of the most significant problems concerns the lack of explicit referential integrity constraints. In the schema in figure 1.4, although the relationships between tables have been inferred manually, they are not defined at the database level through the use of foreign keys. This absence prevents the database from guaranteeing data integrity: for example, a row in a child table could refer to a non-existent row in the parent table. Such a scenario, known as a "dangling reference[4]", can generate anomalies and inconsistencies in the system, making the data unreliable and debugging extremely complex.

Data Redundancy

Another issue is the high data redundancy and the large number of duplicate attributes across different tables. This design, which deviates from the principles of normalization, leads to storing the same information in multiple locations, which not only increases the data volume but also introduces a high risk of inconsistencies.

Unclear and Inconsistent Naming Conventions

The naming convention for attributes and tables is inconsistent and unclear. The schema presents a mixture of conventions (e.g., `IDGroup` vs. `GroupID`, English names mixed with specific terms) and typos (e.g., `TBParamenters`), making it difficult to immediately understand the function of each field. In many cases, the naming is not explicit enough to describe the role of an attribute, requiring an in-depth knowledge of the system to correctly interpret the relationships and business logic. This lack of a unique and standardized vocabulary hinders collaboration among developers, complicates code maintenance, and increases the likelihood of errors.

1.3.4 Maintenance and Operational Challenges

The daily management of the system is made complex by several inefficiencies and obsolete design choices.

The current system design does not allow for the effective reuse of Service Requests and catalog configurations. Consequently, every modification or addition requires the creation of a new Service Request, even for small variations, leading to a innefficient and difficult-to-manage catalog, consisting of

[4] The "dangling pointer" is a problem typically addressed in the context of low-level programming, where a pointer refers to a memory area that is no longer valid because it has been freed or because the pointer is used outside the context of the variable's existence to which it refers [2], but the same dynamic can also be applied in the context of databases, where the "pointers" are represented by foreign keys.

numerous duplicate SRs and fields. This, while functional, is not a scalable approach and leads to significant inefficiencies in catalog management itself. For example, if a Service Request needs to be modified for all clients who use it, the operator will have to modify not only the SR in question but also all the duplicate SRs for each client, with the risk of forgetting a modification or making mistakes. This approach, although probably conceived to prevent error propagation, makes the catalog difficult to navigate and maintain.

The new architecture will need to address these challenges by introducing a more robust, versioned system based on modern software design principles and patterns.

Chapter 2

Architectural Redesign

- Collection and definition of requirements for the new SRP system (ServiceRequestManager)
- Propose and implement a complete architectural redesign of the SRP system

Chapter 3

Implementation of a Modular and Scalable Solution

- Overview of technologies to be used (backend, frontend, database)
- Description of tools for automation and containerization
- Reasons behind the choice of said technologies
- Implementation of a modular and scalable solution that can accomodate various client-specific configurations

Chapter 4

Migrations

- Plan and execute a full migration of databases and other company services / platforms that use SRP

Chapter 5

Reccomendation system through RAG and Fine-tuned LLMs

- Integration of Retrieval-Augmented Generation (RAG) techniques
- Fine-tuning of Large Language Models (LLMs)
- Implementation of a recommendation system for service requests

Chapter 6

Conclusions

- Reflections on the internship experience and acquired skills
- Considerations on the future of the project and potential developments

Bibliography

[1] *Cryptsetup and LUKS - open-source disk encryption.* URL: `https://gitlab.com/cryptsetup/cryptsetup` (cit. on p. 6).

[2] Paul J. Deitel Harvey M. Deitel. *C++. Fondamenti di programmazione.* Apogeo, 2005 (cit. on p. 14).

[3] *K3s - Lightweight Kubernetes.* URL: `https://docs.k3s.io/` (cit. on p. 6).

[4] *SQL Server Integration Services.* URL: `https://learn.microsoft.com/en-us/sql/integration-services/sql-server-integration-services` (cit. on p. 13).

[5] *Stored Procedures (Database Engine).* URL: `https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine` (cit. on p. 12).