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:

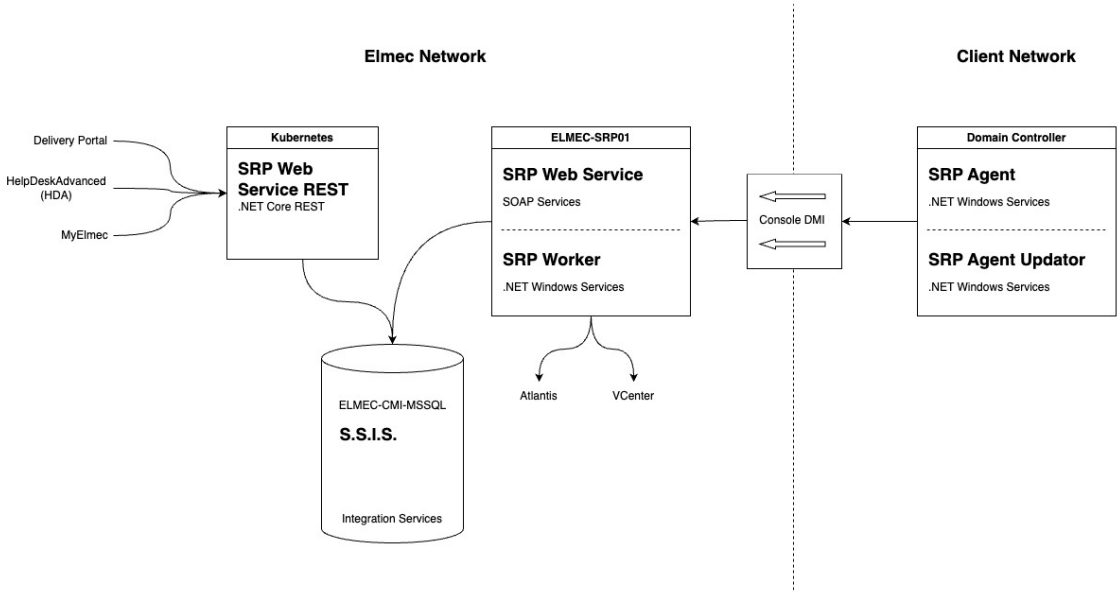

Figure 1.1: AS-IS System Architecture

  - **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**.

La console DMI funge da componente infrastrutturale chiave, agendo principalmente come un ponte di comunicazione tra la rete del cliente e la rete interna di Elmec. Nello specifico, per il sistema SRP, la console DMI è responsabile del trasferimento delle richieste di servizio (SR) dalla rete del cliente alla rete interna di Elmec, sebbene non gestisca il traffico nella direzione inversa.

Funzionamento della Console DMI

La connettività delle appliance DMI verso Elmec è realizzata tramite molteplici connessioni OpenVPN, stabilite dall'appliance verso i Data Center di Brunello 4 e Mendrisio (per i clienti svizzeri). Le console DMI non sono esposte direttamente su Internet e sono accessibili solo dal personale autorizzato. I requisiti hardware per un nodo DMI includono 4 GB di RAM, 4 Core e 50 GB di disco.

A partire da recenti evoluzioni delle DMI, le nuove console utilizzano K3S per la gestione dei container. K3S una distribuzione leggera e certificata di Kubernetes, progettata per ambienti edge, IoT e CI/CD. Si distingue per essere un singolo binario che riduce le dipendenze esterne e usa SQLite come database predefinito, semplificando notevolmente l'installazione e la gestione. Include già al suo interno componenti essenziali come *containerd* [3].

Tutte le console sono gestite centralmente tramite Rancher, e il deploy dello stack applicativo è delegato ad ArgoCD, che assicura che le versioni dei container siano costantemente aggiornate. Per motivi di sicurezza, la console in rete cliente espone di base solo l'SSH, mentre servizi come SRP vengono abilitati selettivamente e solo quando necessario. Tutti i dati applicativi presenti sulle console sono salvati su un volume cifrato LUKS (Linux Unified Key Setup)[1], che può essere aperto solo se il tunnel VPN è instaurato. L'aggiornamento del sistema operativo avviene tramite un processo standard Elmec, denominato PMD.

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]Metodo di cifratura dei dischi rigidi indipendente dalla piattaforma, che può essere utilizzato per cifrare dischi in un'ampia varietà di strumenti. Questo non solo assicura piena compatibilità e interoperabilità tra software diversi, ma garantisce che la gestione delle password avvenga in una modalità sicura e documentata [1].

and potential errors.

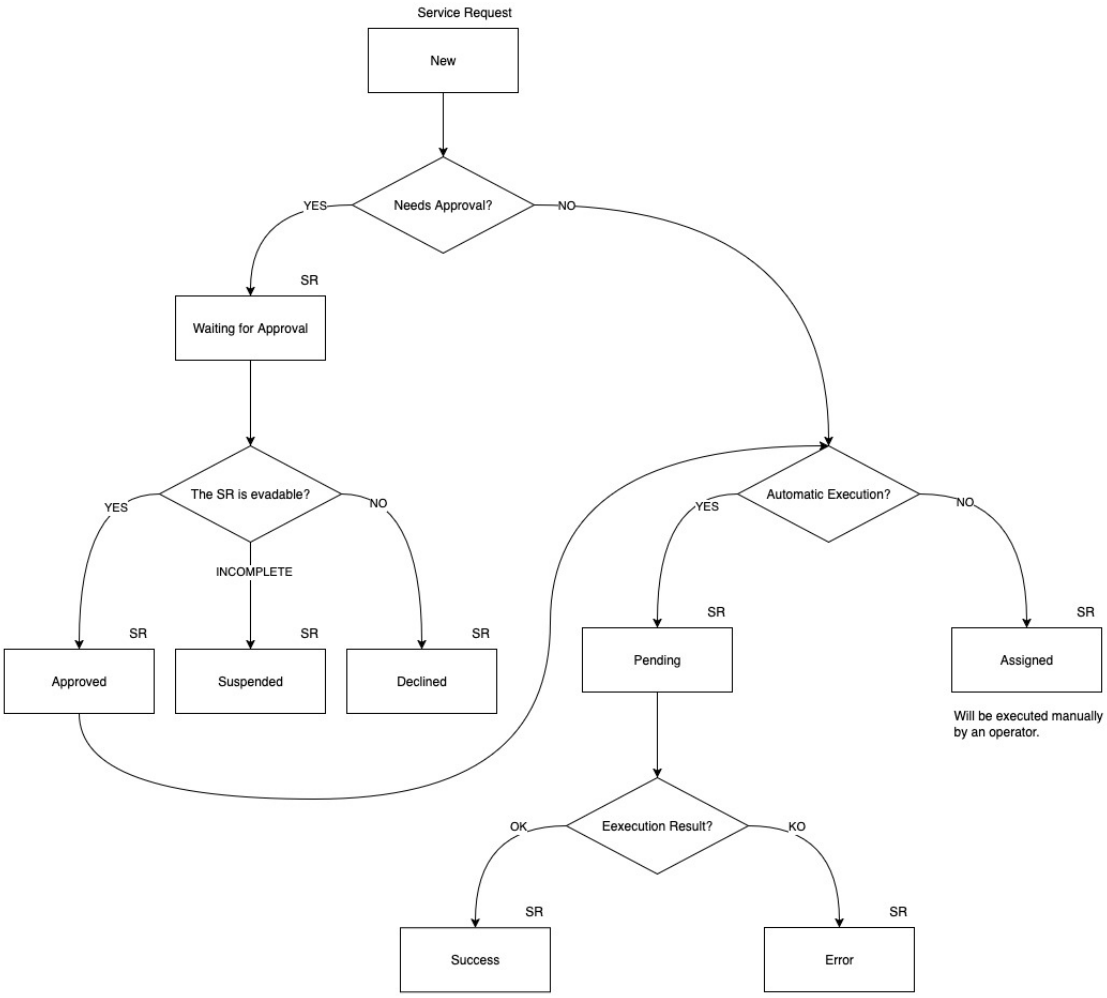

Figure 1.2: High-Level Service Request Lifecycle

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

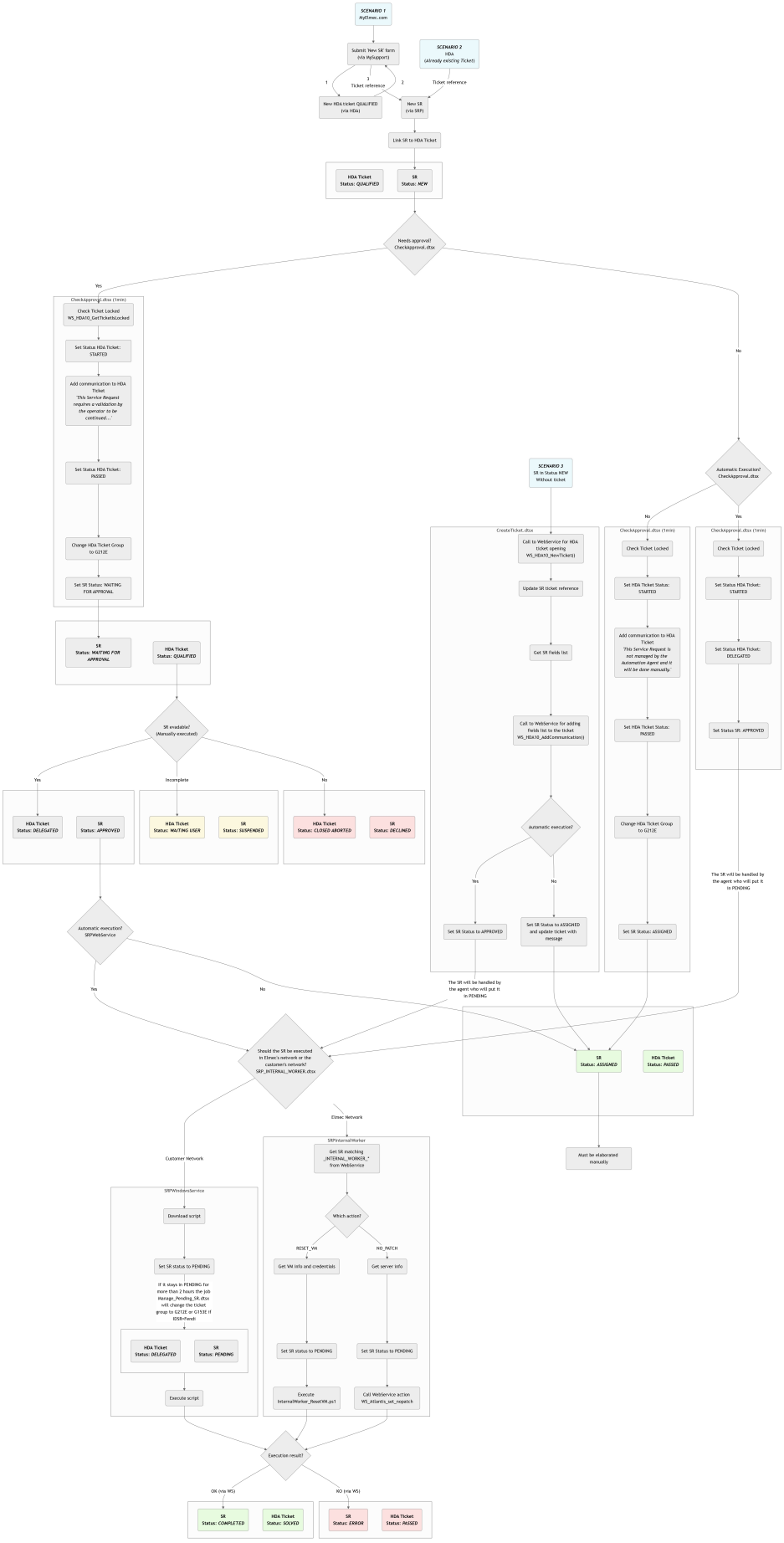

Figure 1.3: AS-IS Complete Service Request Execution Workflow

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

L'analisi dell'architettura e del flusso di lavoro del sistema SRP nella sua forma attuale ha rivelato una serie di problematiche che ne compromettono la scalabilità, la manutenibilità e l'affidabilità. Questi problemi sono profondamente radicati nelle scelte tecnologiche e di design fatte anni fa e si manifestano in diverse aree chiave del sistema, dalla gestione del catalogo all'esecuzione degli script e all'import delle anagrafiche da Active Directory.

1.3.1 Architectural and Technological Challenges

Il sistema SRP presenta una struttura architetturale frammentata e una complessa integrazione tra i suoi componenti principali, portando a diverse criticità.

Monolithic Frontend Component

L'interfaccia utente per la navigazione del catalogo delle Service Request e per la compilazione delle relative forms è un componente front-end Vue.js eccessivamente grande e complesso (quasi 11 mila righe). Questo rende le modifiche, le estensioni, e le customizzazioni per clienti specifici estremamente

difficili. La logica per la gestione di campi dinamici, le relazioni tra di essi, le regole di compilazione e la validazione dei campi sono tutte gestite a livello di frontend con una lunga serie di if con valori hard-coded, come il seguente:

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

Questo, oltre a mescolare logica di business e presentazione, crea un grosso problema a livello di configurazione del catalogo, in quanto non esiste un modo centralizzato per gestire i campi e le loro relazioni. Ogni modifica richiede quindi un intervento diretto sul codice, rendendo il sistema poco flessibile e difficile da mantenere, soprattutto in un contesto in cui l'operatore che va a mettere mano al catalogo non è un programmatore e non sa come è strutturato il codice del frontend. Ad esempio, se un operatore volesse aggiungere un nuovo campo "Domain" a una Service Request, questo campo dovrà avere esattamente descrizione "Domain" e tipo "domain", altrimenti il componente Vue.js non lo renderizzerà correttamente.

Queste dinamiche rendono il sistema fragile e suscettibile a frequenti errori che fanno sprecare tempo e risorse sia ai clienti che agli operatori.

Servizio REST Backend eccessivamente generico

Infatti, l'unica e sola informazione relativa ai campi che viene restituita dal servizio REST backend (SRPWebSeviceREST), oltre al nome del campo e alla sua mandatorietà, è il suo *type*. Diventa quindi facile immaginare come questa informazione, che dovrebbe essere solamente un'indicazione sul componente che dovrà essere renderizzato (e.g. textbox, select, radio button, etc..) può diventare un identificatore di un qualcosa di molto più ampio.

Ad esempio, un campo di tipo `UPN` è un campo composto dal valore di un altro campo 'Display Name', concatenato a una '@', con suffisso il valore di una select le cui opzioni sono i domini del cliente, forniti da un servizio REST esterno (e.g. "`n.guarini@elmec.it`", dove "`n.guarini`" è il valore del campo "Display Name", e "`elmec.it`" è il valore del campo select "Domain"). Nonostante questa complessità, il servizio REST restituisce solamente un documento JSON di questo tipo:

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

delegando quindi al frontend la responsabilità di interpretare il tipo e renderizzare il campo in modo corretto.

Dipendenza dall'agente SRP

L'importazione dei dati dagli Active Directory dei clienti e l'esecuzione effettiva delle Service Requests sono gestite da un agente installato direttamente sui Domain Controller dei clienti. Questo approccio solleva diverse problematiche:

- **Sicurezza e Permessi**: Spesso l'agente richiede credenziali con privilegi elevati (es. domain-admin), che i clienti sono sempre più restii a concedere.
- **Affidabilità e Controllo**: Le configurazioni sull'agente e sul Domain Controller sono complesse e difficili da gestire centralmente, potendo causare errori di esecuzione e di sincronizzazione dei dati.
- **Scalabilità**: La necessità di installare, configurare e manutenere un agente per ogni dominio cliente rappresenta un notevole onere operativo. Si stanno valutando alternative come sistemi di provisioning più moderni.
- **Obsolete Communication Patterns**: La comunicazione unidirezionale attraverso la *DMI Console* crea un collo di bottiglia e complica la gestione delle operazioni asincrone. La comunicazione tra i componenti nella rete del cliente e la rete aziendale è basata su polling (es. SRP Agent che interroga il server ogni 5 minuti), un approccio che introduce latenza e inefficienze.

1.3.2 Code and Data Management Issues

La gestione del codice e della persistenza dei dati è un altro aspetto critico del sistema, caratterizzato dalla mancanza di versionamento e di strumenti di debug moderni.

Unversioned and Undebuggable Stored Procedures

L'utilizzo intensivo di Stored Procedures nel database MSSQL rappresenta uno dei maggiori punti di debolezza. Queste procedure contengono logica di business fondamentale e sono:

- **Non Versionate**: Non esiste un sistema di controllo versione centralizzato per le Stored Procedures. Le modifiche vengono apportate direttamente sul database, rendendo impossibile tracciare i cambiamenti, tornare a versioni precedenti o testare le modifiche in ambienti separati.
- **Difficili da Debuggare**: Il debugging delle Stored Procedures è un processo complesso e macchinoso, richiedendo spesso l'accesso diretto e gli strumenti specifici del server SQL, con il rischio di influenzare l'ambiente di produzione.

Inaccessible and Unversioned SSIS Jobs

I pacchetti S.S.I.S. sono essenziali per l'importazione e l'esportazione dei dati (es. anagrafiche AD). Tuttavia, questi job risiedono fisicamente sul server ELMEC-CMI-MSSQL e l'unico modo per accedervi e modificarli è attraverso un desktop remoto. Questo approccio impedisce il versionamento e la collaborazione, limita l'accessibilità rendendo la manutenzione e il debugging un'attività onerosa e rischiosa, e crea un *single-point-of-failure* e un forte accoppiamento tra la logica di business di import / export e l'infrastruttura.

1.3.3 Database Schema and Data Integrity Issues

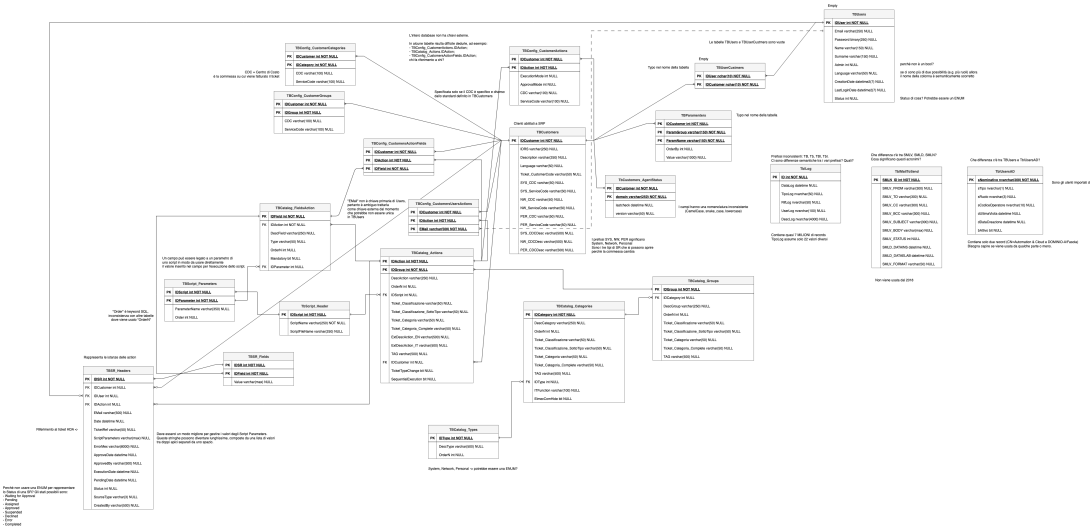

Figure 1.4: AS-IS Database ER Schema

L'analisi dello schema del database, rappresentato in figura 1.4, ha rivelato una serie di problematiche significative che influenzano la qualità e l'integrità dei dati gestiti dal sistema SRP.

Lack of Data Integrity and Consistency

Uno dei problemi più significativi riguarda la mancanza di vincoli di integrità referenziale espliciti. Nello schema in figura 1.4, sebbene siano state dedotte le relazioni tra le tabelle, queste non sono definite a livello di database mediante l'uso di chiavi esterne. Questa assenza non permette al database di garantire l'integrità dei dati: ad esempio, una riga in una tabella figlia potrebbe fare riferimento a una riga inesistente nella tabella padre. Tale scenario, noto come "dangling reference[2]", può generare anomalie e incoerenze nel sistema,

[2]Il "dangling pointer" è un problema tipicamente trattato nel contesto della programmazione a basso livello, dove un puntatore si riferisce ad un'area di memoria non più valida, perché già liberata o perché il puntatore viene utilizzato all'esterno del contesto di esistenza

rendendo i dati inaffidabili e il debugging estremamente complesso.

Data Redundancy

Un'altra problematica è l'elevata ridondanza dei dati e al grande numero di attributi duplicati in diverse tabelle. Questo design, che si discosta dai principi di normalizzazione, porta alla memorizzazione della stessa informazione in più luoghi, che non solo aumenta il volume dei dati, ma introduce anche un elevato rischio di inconsistenze.

Unclear and Inconsistent Naming Conventions

La nomenclatura degli attributi e delle tabelle risulta inconsistente e poco chiara. Lo schema presenta una mescolanza di convenzioni (es. `IDGroup` vs. `GroupID`, nomi in inglese misti a termini specifici) e typo (es. `TBParamenters`), rendendo difficile la comprensione immediata della funzione di ciascun campo. In molti casi, la nomenclatura non è sufficientemente esplicita per descrivere il ruolo di un attributo, richiedendo una conoscenza approfondita del sistema per interpretare correttamente le relazioni e la logica di business. Questa mancanza di un vocabolario univoco e standardizzato ostacola la collaborazione tra gli sviluppatori, complica la manutenzione del codice e aumenta la probabilità di errori.

1.3.4 Maintenance and Operational Challenges

La gestione quotidiana del sistema è resa complessa da diverse inefficienze e scelte di progettazione obsolete.

Il design del sistema attuale non permette riutilizzo efficace delle Service Request e delle configurazioni del catalogo. Di conseguenza, ogni modifica o aggiunta richiede la creazione di una nuova Service Request, anche per piccole variazioni, portando a un catalogo ingombrante e difficile da gestire, composto da numerosissime duplicazioni di SR e campi. Questo, per quanto funzionante, non è un approccio scalabile e porta a inefficienze significative nella gestione del catalogo stesso, ad esempio se una Service Request deve essere modificata per tutti i clienti che la utilizzano, l'operatore dovrà modificare non solo la SR in questione, ma anche tutte le SR duplicate per ogni cliente, con il rischio di dimenticare qualche modifica o di fare errori.

Questo approccio, sebbene concepito per prevenire la propagazione di errori, rende il catalogo ingombrante e difficile da navigare e manutenere.

---

della variabile cui si riferisce [2], ma la stessa dinamica si può applicare anche nel contesto dei database, dove i "puntatori" sono rappresentati dalle chiavi esterne.

La nuova architettura dovrà affrontare queste sfide introducendo un sistema più robusto, versionato e basato su principi e pattern di software design moderni.

Chapter 2

Architectural Redesign

- Collection and definition of requirements for the new SRP system (ServiceRequestManager)
- Types of fields required (free, registries, custom)
- Integration with Active Directory and management of credentials
- 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 modern technologies
- Design 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).