ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Processo di Transizione e Reingegnerizzazione di un Pannello Web per DNS Filtering

Tesi di laurea in:
Paradigmi di Programmazione e Sviluppo

Relatore
Prof. Mirko Viroli

Candidato
Alberto Spadoni

Correlatori
Dott. Nicolas Farabegoli
Dott. Gianluca Aguzzi

IV Sessione di Laurea
Anno Accademico 2023-2024

Abstract

Negli ultimi anni, le minacce informatiche sono diventate sempre più diffuse, con la maggior parte degli attacchi che avvengono attraverso Internet. Per mitigare questi rischi, le aziende adottano diverse strategie di sicurezza, tra cui il filtraggio DNS, una pratica efficace per bloccare accessi a domini malevoli e applicare politiche di sicurezza a livello di rete. Questo approccio risulta particolarmente strategico in quanto opera a un livello intermedio tra gli endpoint e le applicazioni, bilanciando efficacia, semplicità di implementazione e limitato impatto sulle risorse di sistema.

In questo contesto, la presente tesi si concentra sulla reingegnerizzazione del pannello per la configurazione e gestione del filtro DNS di *FlashStart Group srl*. L'obiettivo è la progettazione e l'implementazione di una nuova infrastruttura moderna e scalabile, basata su un'architettura a microservizi. Il lavoro ha riguardato la riprogettazione del database, l'introduzione di una dashboard avanzata per MSP e la definizione di un sistema strutturato per la gestione degli errori.

L'approccio adottato ha seguito una strategia di reingegnerizzazione ibrida, ossia un mix di sviluppo incrementale e sostituzione totale, evitando così la coesistenza tra vecchio e nuovo sistema. Sono state implementate le prime funzionalità chiave, mentre ulteriori sviluppi riguarderanno l'integrazione di un sofisticato sistema di controllo degli accessi, la completa documentazione del sistema e l'ottimizzazione dell'interfaccia utente.

I risultati ottenuti finora evidenziano le potenzialità del nuovo sistema, che supera la versione legacy in termini di usabilità, sicurezza e manutenibilità. Questo progetto rappresenta un primo passo verso una piattaforma più efficiente, orientata alle esigenze di un mercato enterprise. L'architettura rinnovata costituisce una solida base per sviluppi futuri, consentendo all'azienda di rafforzare la propria posizione competitiva in un settore in rapida evoluzione.

Alla mia famiglia, a Lara ed al mio futuro

Indice

Elenco delle figure

Capitolo 1

Introduzione

La presente tesi nasce dall'esperienza di tirocinio svolta presso un'azienda specializzata nello sviluppo di soluzioni per il filtraggio DNS. Durante tale esperienza, è stato possibile contribuire alle prime fasi di reingegnerizzazione del pannello Web per la configurazione di un filtro DNS. Questo pannello, concepito per offrire funzionalità di gestione e controllo dei domini da filtrare, presentava tuttavia limiti strutturali e tecnologici che ne ostacolavano la manutenibilità e l'estensibilità. L'esigenza di modernizzare l'architettura, migliorare l'esperienza utente e garantire maggiore sicurezza è nata anche dalla volontà di posizionare l'azienda in un mercato più orientato alle medie-grandi organizzazioni. Questo obiettivo mira a favorirne la crescita e a consolidarne il ruolo come punto di riferimento a livello globale nel settore del filtraggio DNS.

Il presente lavoro si inserisce in questo contesto, con l'obiettivo di progettare e sviluppare una nuova versione del pannello di configurazione del filtro DNS aziendale, adottando un'architettura moderna basata su microservizi. Questo approccio mira a garantire maggiore efficienza, sicurezza e manutenibilità rispetto alla versione precedente, superando le rigidità del legacy e introducendo nuove funzionalità essenziali.

Scopo dell'elaborato è fornire una visione organica del processo di reingegnerizzazione svolto, illustrando le motivazioni alla base della necessità di aggiornare il pannello, i principali obiettivi progettuali e i risultati raggiunti nelle prime fasi di sviluppo.

Il contesto teorico di riferimento comprende sia il Domain Name System (DNS), con le sue logiche di funzionamento e la sua importanza nella gestione del traffico in rete, sia le metodologie di reingegnerizzazione software, che offrono linee guida per affrontare in modo sistematico la modernizzazione di soluzioni esistenti.

Dal punto di vista metodologico, il progetto ha richiesto un'attenta fase di analisi, seguita dalla definizione di un'architettura più flessibile e scalabile, fino all'implementazione di un primo set di funzionalità.

Durante lo sviluppo, sono state affrontate diverse sfide, tra cui la transizione da un'architettura monolitica a una basata su microservizi, la riprogettazione della base dati e la creazione di un sistema avanzato di gestione degli errori, in grado di operare in modo strutturato e pervasivo su tutti i livelli del sistema.

1.1 Struttura dell'elaborato

L'elaborato è suddiviso nei seguenti capitoli, ciascuno dedicato a un aspetto specifico del progetto:

- **Capitolo 2: Background** – Introduce il funzionamento del DNS e il filtraggio DNS, oltre alle metodologie di reingegnerizzazione software.
- **Capitolo 3: Analisi** – Esamina il sistema legacy, evidenziandone l'architettura, le tecnologie adottate e le limitazioni riscontrate. Inoltre, introduce i requisiti del nuovo sistema e la modellazione del dominio.
- **Capitolo 4: Design** – Presenta la nuova architettura proposta, sia lato frontend che backend, illustrando le interazioni tra i componenti e le principali scelte tecnologiche adottate.
- **Capitolo 5: Implementazione** – Descrive il processo di sviluppo del nuovo sistema, soffermandosi sull'infrastruttura, sull'organizzazione della repository e sull'implementazione del sistema di gestione degli errori.
- **Capitolo 6: Valutazione** – Analizza i requisiti soddisfatti e le funzionalità implementate, confrontandole con gli obiettivi iniziali. Inoltre, discute i miglioramenti pianificati e gli aspetti ancora da sviluppare.

- **Capitolo 7: Conclusione** – Riassume il lavoro svolto, evidenziando le principali lezioni apprese e proponendo possibili sviluppi futuri del sistema.

Grazie a questa suddivisione, l'elaborato mira a fornire una panoramica completa del percorso di reingegnerizzazione del sistema, dall'analisi dei requisiti iniziali fino alle valutazioni finali e ai suggerimenti per successivi sviluppi. L'obiettivo ultimo è offrire un contributo concreto alla modernizzazione del software, ponendo le basi per futuri interventi di mantenimento e potenziamento del sistema.

Capitolo 2

Background

2.1 Introduzione al DNS e al filtraggio

Prima di trattare l'argomento cardine del presente capitolo, si ritiene opportuno fare una breve panoramica sui concetti importanti a esso collegati. Verrà in prima battuta presentato il Domain Name System (DNS), che può essere definito come uno dei pilastri fondamentali di tutta l'architettura della rete Internet. Successivamente, ci si sposterà sull'ambito del filtraggio in Internet, che rappresenta il contesto più ampio di cui il filtraggio DNS fa parte.

2.1.1 Cos'è il DNS e il suo ruolo in Internet

Il Domain Name System è un database gerarchico e distribuito che contiene le associazioni tra nomi di dominio e altre importanti informazioni, tra cui gli indirizzi IP. Questo fondamentale sistema consente agli utenti di localizzare le risorse sulla rete andando a convertire nomi di dominio familiari e in formato leggibile dagli umani in indirizzi numerici ai quali un computer può connettersi. Un'analogia comune che si utilizza per spiegare il ruolo del sistema DNS è che esso serve da rubrica telefonica per Internet, andando a tradurre i nomi di computer comprensibili agli umani nei relativi indirizzi numerici interpretabili dalle macchine. Per fare un esempio, il nome di dominio `www.airbus.com` viene tradotto dal DNS nell'indirizzo IPv4 `107.154.76.155`.

Vulnerabilità del DNS

Il DNS rappresenta una porzione cruciale della rete Internet e per questo motivo la sua messa in sicurezza risulta molto importante. Infatti, se un individuo malintenzionato dovesse riuscire a comprometterlo, sarebbe in grado di bloccare, o comunque ridurre, le normali attività che avvengono sulla rete.

Il sistema in oggetto è stato progettato negli anni '80 per rispondere alla necessità di una risoluzione dei nomi rapida e scalabile su una rete in continua espansione. Al momento della sua concezione, come descritto nelle specifiche originarie RFC 1034 [Moc87a] e RFC 1035 [Moc87b], l'attenzione era principalmente concentrata sulla funzionalità e sull'efficienza, senza considerare i potenziali problemi di sicurezza che sarebbero emersi con la crescita esponenziale di Internet. Oltretutto, la solida fiducia che le specifiche RFC trasmettevano ai professionisti IT dell'epoca li ha portati a trascurare, o comunque a non approfondire, i potenziali rischi di sicurezza associati a tale sistema [HH14]. Questa scelta rifletteva il contesto storico in cui è nato il DNS: la rete era allora utilizzata principalmente da enti accademici e governativi, con un livello di fiducia reciproca tra i partecipanti. Tuttavia, con l'apertura di Internet a un pubblico globale, il DNS ha rivelato vulnerabilità intrinseche, tra cui la mancanza di meccanismi nativi che garantiscano l'autenticazione delle risposte e l'integrità delle informazioni da esso fornite. Tali lacune hanno reso possibile una serie di attacchi che sfruttano le debolezze intrinseche del sistema DNS. Tra questi, figurano il DNS Spoofing, il DNS Amplification Attack e il DNS Hijacking.

DNS Spoofing Il DNS spoofing, noto anche come Cache Poisoning, consiste nell'iniettare dati malevoli nella cache dei server DNS bersaglio, inducendoli a restituire informazioni errate agli utenti. Questo attacco permette ai malintenzionati di reindirizzare il traffico verso siti controllati da essi, facilitando il furto di credenziali o altre forme di attacchi avanzati. Un esempio storico di DNS spoofing è l'attacco di Eugene Kashpureff del 1997, il quale riuscì a reindirizzare tutti i visitatori del dominio `internic.net` verso il sito della compagnia Alternic, di cui era il fondatore [LMM+00].

In generale, questa tipologia di attacco sfrutta la mancanza di autenticazione

nelle risposte DNS e l'assenza di integrità nelle informazioni memorizzate nella cache. I metodi per mitigare il DNS spoofing includono sostanzialmente l'implementazione di DNSSEC, che garantisce l'autenticità delle risposte attraverso firme digitali [III99].

DNS Amplification Attacks Gli attacchi di tipo Distributed Denial of Service (DDoS) rappresentano una delle minacce più critiche per la sicurezza informatica, mirate a compromettere la disponibilità di un sistema. Questi attacchi si basano sull'invio massivo di richieste a un bersaglio per esaurirne le risorse, come CPU, memoria o larghezza di banda, rendendo il servizio inaccessibile agli utenti legittimi.

Un esempio rilevante nell'ambito del DNS è rappresentato dagli attacchi di amplificazione, una tipologia particolarmente efficace di DDoS. In questi attacchi, l'aggressore invia richieste DNS utilizzando un indirizzo IP sorgente falsificato, facendole sembrare provenienti dalla vittima. I server DNS, ingannati, rispondono con pacchetti significativamente più grandi delle richieste originali, amplificando così il volume del traffico diretto alla vittima. Sfruttando la differenza tra la dimensione delle richieste e delle risposte, gli attaccanti sono in grado di moltiplicare il traffico generato, saturando rapidamente le risorse della vittima e compromettendone il funzionamento [AKA+16].

Per difendersi dagli attacchi di amplificazione DNS, i fornitori di servizi Internet (ISP) possono limitare l'accesso al proprio server DNS solamente agli utenti autorizzati, monitorare il traffico in entrata per individuare anomalie, e impostare dei limiti alla quantità di dati che il server DNS può inviare in risposta a una query.

Un caso pratico di amplificazione DNS è quello che ha colpito Spamhaus nel 2013, considerato uno dei più grandi attacchi DDoS mai accaduti. Esso è stato caratterizzato da una richiesta di 36 byte che ha generato una risposta di 3.000 byte, amplificando il traffico di un fattore 100 e generando un volume di dati in entrata ai server della compagnia pari a 75GBps [BCL21].

DNS Hijacking Il DNS hijacking prevede la compromissione di server DNS o la manipolazione delle configurazioni DNS di un utente per reindirizzare il traffico

verso destinazioni controllate da un malintenzionato. Questo tipo di attacco può essere implementato seguendo due strategie [HH14]:

- attraverso l'uso di specifici malware che modificano le impostazioni DNS locali, oppure
- mediante la compromissione diretta dei server DNS fidati, in modo che non si comportino come da specifica.

Le conseguenze di questo tipo di attacco includono il furto di credenziali, la diffusione di malware e la censura di contenuti web. Tuttavia, alcuni provider di servizi Internet utilizzano questa tecnica per scopi commerciali, come la visualizzazione di pubblicità o la raccolta di statistiche sulla navigazione dei propri clienti. Per mitigare il DNS hijacking, si consiglia di utilizzare configurazioni DNS sicure e controllare in maniera scrupolosa le modifiche ai record DNS locali [HH14].

2.1.2 Il filtraggio di Internet

A seguito dell'analisi riguardante le vulnerabilità intrinseche del sistema DNS, emerge chiaramente l'importanza di adottare meccanismi di prevenzione per garantire una navigazione più sicura e controllata. A questo proposito, una soluzione interessante riguarda il filtraggio di Internet, che, in generale, rappresenta un insieme di tecniche e tecnologie volte a limitare o bloccare l'accesso a contenuti considerati inappropriati, illegali o dannosi. Le principali pratiche di filtraggio nell'ambito di Internet si dividono nelle tre seguenti tipologie:

- **IP Packet Filtering**: ovvero il blocco del traffico basato su indirizzi IP o numeri di porta. Questa tecnica analizza gli header dei pacchetti per decidere se consentirne il passaggio o eliminarli. Sebbene sia semplice ed efficiente, può causare sovrafiltraggio, rendendo inaccessibili tutti quei contenuti legittimi ospitati su un indirizzo IP considerato malevolo.
- **DNS Poisoning**: consiste nella manipolazione delle risposte DNS per reindirizzare richieste a indirizzi errati o per bloccarle completamente. Sebbene sia nata come una pratica malevola (Sezione 2.1.1), i suoi principi tecnici sono stati adattati e applicati a un contesto legittimo di filtraggio.

- **URL Blocking**: implementa il blocco di URL specifici tramite proxy HTTP o tecniche come la Deep Packet Inspection (DPI). Questi sistemi analizzano il contenuto delle richieste HTTP per identificare e bloccare pagine o risorse specifiche, offrendo una maggiore precisione rispetto ai metodi sopra elencati. L'unico svantaggio di quest'ultimo approccio deriva dal fatto che risulta più complesso da implementare e richiede una maggiore potenza di calcolo rispetto alle altre tecniche.

Punti di applicazione del filtraggio di Internet

Il filtraggio di Internet può essere implementato in diversi livelli dell'architettura di rete, con implicazioni specifiche in termini di efficacia, costi e complessità. Secondo V. Varadharajan [Var10], i principali punti di applicazione includono:

- **A monte dell'ISP**: i filtri possono essere posizionati nei fornitori internazionali di connettività o presso le dorsali di rete. Questo approccio consente di bloccare contenuti indesiderati prima che raggiungano l'ISP locale, risultando particolarmente utile per gestire il traffico proveniente da server internazionali. Tuttavia, questa strategia richiede una stretta collaborazione tra le entità coinvolte e una chiara mappatura dei gateway internazionali.

- **All'interno dell'ISP**: posizionare i filtri nel cuore della rete dell'ISP permette di monitorare e controllare il traffico a livello centralizzato per tutti i suoi utenti. Pur essendo efficace, questa soluzione può introdurre colli di bottiglia e richiedere investimenti significativi in hardware sofisticato, soprattutto per ISP con un alto volume di traffico.

- **Tra l'ISP e il cliente**: applicare il filtraggio nel collegamento tra l'ISP e il cliente finale offre un controllo mirato sul traffico diretto verso reti locali o singoli utenti. Questa configurazione consente di personalizzare le policy di filtraggio in maniera più granulare rispetto ai precedenti punti di applicazione, ma può comportare costi operativi elevati per ISP con grandi numeri di clienti.

- **Tra due clienti dello stesso ISP**: in reti locali o interne a un ISP, i filtri possono essere configurati per impedire lo scambio di contenuti indesiderati

tra utenti dello stesso provider. Questo approccio è utile in contesti aziendali o comunità chiuse.

- **Tra un cliente e un sito web ospitato a livello internazionale**: i filtri possono essere utilizzati per monitorare e bloccare il traffico tra un utente e server situati all'estero. Questa strategia è fondamentale per controllare contenuti provenienti da altri paesi, ma richiede un'infrastruttura adeguata e la capacità di gestire connessioni internazionali.

La possibilità di distribuire le tecniche di filtraggio su diversi livelli offre un'elevata flessibilità nell'implementazione delle strategie di controllo dei contenuti. Tuttavia, ogni livello si porta con sé alcuni compromessi: il posizionamento a monte può bloccare grandi volumi di traffico, ma con rischi di sovrafiltraggio, mentre i filtri localizzati vicino agli utenti offrono maggiore precisione ma richiedono risorse significative. Una combinazione di approcci, come quella messa in atto dalla British Telecom's CleanFeed, può ottimizzare l'efficacia del filtraggio, bilanciando precisione e scalabilità [Var10].

2.1.3 Il filtraggio DNS

Tra le varie tecniche di Internet filtering descritte in precedenza, il filtraggio a livello DNS si distingue per la sua semplicità e versatilità. Al contrario di altri approcci, esso agisce direttamente sul sistema che costituisce la base della navigazione online. Questo lo rende un punto di controllo strategico ed efficace per implementare politiche di sicurezza e controllo dell'accesso. Oltretutto, i cybercriminali stanno sempre di più prendendo di mira l'infrastruttura DNS per condurre le loro attività malevole. Per questo motivo, le tecniche di filtraggio a questo livello assumono un ruolo centrale nella missione di rendere la rete più sicura e proteggere gli utenti dai rischi di attacchi. Nei paragrafi successivi verrà fornita una definizione del metodo in questione, insieme a dettagli sulle sue applicazioni pratiche e sulle modalità che ne abilitano il funzionamento.

Il *DNS Filtering* consiste nell'applicare regole predefinite per bloccare o consentire richieste e risposte DNS, analogamente a quello che fa un firewall. Questa tecnica è in grado di prevenire l'accesso a contenuti malevoli o inappropriati, sup-

portare l'applicazione di politiche aziendali e proteggere gli utenti da minacce provenienti dalla rete [Mag24]. Volendo essere più specifici, il filtraggio DNS trova applicazione in una vasta gamma di scenari pratici [MA08, Var10]:

- **Sicurezza informatica**: viene utilizzato per bloccare siti di phishing o che distribuiscono malware e altri contenuti dannosi.
- **Applicazione di politiche aziendali**: permette alle organizzazioni di imporre restrizioni sull'accesso a siti non correlati al lavoro, come social media o piattaforme di streaming, durante l'orario lavorativo.
- **Controllo parentale**: consente la creazione di un ambiente sicuro per i minori, limitando l'accesso a contenuti non appropriati come siti per adulti o violenti.
- **Gestione della larghezza di banda**: aiuta a limitare l'accesso a siti che consumano molta banda, come i servizi di streaming video, ottimizzando così l'utilizzo delle risorse di rete.

Dopo aver introdotto il concetto di filtraggio DNS e averne illustrato le principali applicazioni pratiche, è importante approfondire le tecniche utilizzate per la sua implementazione. Sebbene il DNS Filtering sia ampiamente riconosciuto come uno strumento fondamentale per migliorare la sicurezza delle reti, la sua efficacia dipende dall'applicazione di metodi avanzati per identificare e bloccare le minacce. Come evidenziato in letteratura, la ricerca ha individuato tre tecniche principali che rappresentano lo stato dell'arte in questo campo, ovvero le *Response Policy Zones* (RPZ), i *Threat Intelligence Feeds* (TIF) e il rilevamento di domini generati algoritmicamente (*Domain Generation Algorithm*, DGA) [Mag24].

Queste tecniche non solo permettono di affrontare le minacce più comuni, come phishing e malware, ma forniscono anche strumenti per rispondere a sfide più complesse, come il rilevamento di domini dinamici utilizzati per comunicare con le botnet. Tuttavia, la letteratura sottolinea anche alcune lacune, in particolare la mancanza di un'analisi integrata di queste metodologie. Approfondire ciascuna di esse consente di comprendere meglio il loro funzionamento, le applicazioni e i limiti, fornendo una base solida per ulteriori miglioramenti nel campo del DNS Filtering.

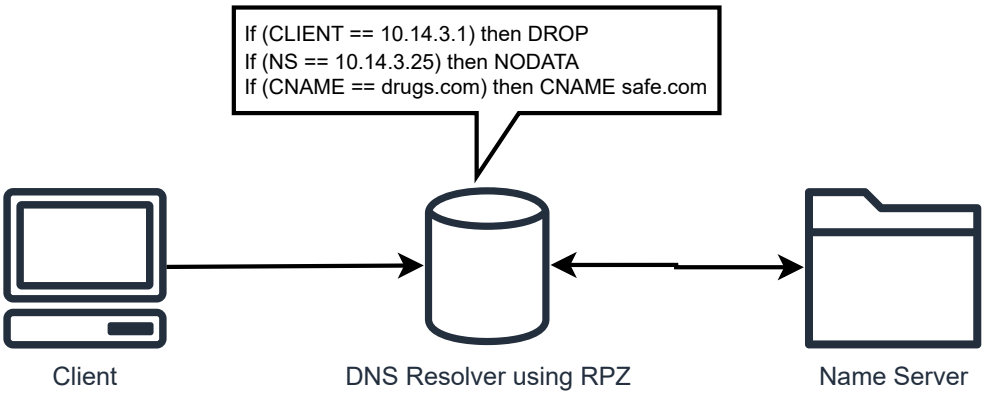

Figura 2.1: Un resolver DNS che sfrutta le Response Policy Zones.

A questo proposito, le sezioni seguenti saranno dedicate all'approfondimento delle tecniche appena citate, per poi fornire uno spunto su come potrebbero essere unite insieme per migliorare significativamente le performance dei filtri.

Response Policy Zones

L'espressione Response Policy Zones descrive una funzionalità dei resolver DNS che consente agli amministratori di rete di specificare politiche per le risposte DNS, sulla base dei nomi di dominio e del contesto. Questa tecnologia opera intercettando le richieste e le risposte DNS, per poi confrontarle con un elenco di politiche definite dall'amministratore. Quando viene rilevata una corrispondenza, il resolver restituisce una risposta in base alla politica, che poi rappresenta l'azione da compiere. Esempi di possibili azioni sono bloccare una certa query oppure reindirizzarla verso un sito sicuro (Figura 2.1, riprodotta da [Mag24]).

Le RPZ rappresentano un potente strumento per l'applicazione delle politiche di sicurezza e per proteggere la rete da attività malevole. Inoltre, il loro essere flessibili e personalizzabili, consente agli amministratori di definire policy molto specifiche, che si adattano a qualsiasi necessità. Tuttavia, per evitare falsi positivi e per mantenere l'efficacia del sistema nel tempo, le RPZ necessitano di una corretta configurazione e manutenzione. Infatti, le policy hanno costantemente bisogno di essere mantenute aggiornate, partendo da informazioni accurate e affidabili.

Infine, si ritiene opportuno puntualizzare un limite di questa tecnologia. In particolare, le RPZ sono in grado di proteggere solo da minacce conosciute. Ciò deriva dalla natura stessa di questa tecnica di filtraggio, che fa affidamento su delle policy definite sulla base di minacce necessariamente già inquadrate e studiate.

Threat Intelligence Feeds

Prima di descrivere la tecnica di filtraggio in sé, si ritiene necessario presentare brevemente i concetti a essa associati: l'espressione *cyber threat intelligence* si riferisce a un insieme di informazioni relative a minacce informatiche potenziali o reali, utili per migliorare la situazione di un'organizzazione in termini di sicurezza. Queste informazioni possono includere dettagli su tattiche, tecniche e procedure (TTP) adottate da attori malevoli noti, nonché indicatori di compromissione (IOC) come indirizzi IP, nomi di dominio e hash di malware. Nel contesto del DNS, i Threat Intelligence Feeds costituiscono dati aggiornati sui domini malevoli e relativi indirizzi IP.

I TIF possono provenire da fonti diverse, tra cui intelligence open-source, fornitori commerciali e gruppi di condivisione delle minacce specifici per settore. Questi flussi includono elenchi di IOC generati analizzando il traffico di rete, i log di sicurezza o altre fonti di intelligence. I resolver, quindi, possono utilizzare questi dati per bloccare o reindirizzare le query DNS verso domini malevoli noti.

In generale, l'integrazione dei TIF nel filtraggio DNS offre una difesa efficace contro le minacce conosciute e aiuta le organizzazioni a rilevare e prevenire gli attacchi prima che possano causare danni. Tuttavia, è importante ricordare che i flussi in questione non rappresentano una soluzione definitiva e devono essere utilizzati insieme ad altre misure di sicurezza per garantire una protezione completa.

Domain Generation Algorithms

Rappresenta una metodologia che attuano i malware per la generazione di nomi di dominio univoci, volti a consentire la comunicazione tra le vittime e i cosiddetti server *Command-and-Control* (C2). Tali domini, vengono solitamente generati partendo dalla data e ora corrente, rendendo il processo di blocco molto difficile

per i sistemi di filtraggio che si basano su liste statiche di indirizzi IP malevoli. Però, grazie a specifici algoritmi di Machine Learning, è possibile individuare questi nomi casuali, riconoscendo i pattern generati algoritmicamente. In particolare, analizzando la struttura dei domini creati dai malware, insieme al comportamento e al contesto delle query DNS, è possibile identificare e bloccare le comunicazioni con i server C2, anche se i nomi dei domini malevoli vengono utilizzati una sola volta.

Uso combinato di RPZ, TIF e rilevamento DGA

L'uso combinato di Response Policy Zones, Threat Intelligence Feeds e tecniche di rilevamento di domini generati algoritmicamente, rappresenta un approccio promettente per migliorare l'efficacia del filtraggio DNS. Le RPZ possono intercettare le query DNS, per poi confrontarle con blocklist e allowlist mantenute costantemente aggiornate grazie ai TIF. A loro volta, i TIF possono integrare tecniche avanzate di rilevamento DGA per identificare domini malevoli conosciuti e sconosciuti, analizzando query anonimizzate.

Un framework che integri queste tecniche può migliorare notevolmente la capacità di rilevare e bloccare domini malevoli, minimizzando i falsi positivi e garantendo un'intelligence tempestiva e accurata. La trasparenza nel processo di filtraggio, che si concretizza nel rendere pubblicamente disponibili le liste di domini da bloccare, può favorire la fiducia degli utenti e prevenire abusi come la censura. Inoltre, il coinvolgimento della community attraverso contributi di esperti e ricercatori nell'ambito della sicurezza informatica, può ulteriormente rafforzare l'efficacia del sistema.

Se da un lato ci sono dei vantaggi nel rendere pubbliche tali informazioni, dall'altro potrebbero emergere argomentazioni a sfavore di questa attività. Infatti, i malintenzionati sarebbero in grado di adattare le loro metodologie di attacco sulla base delle politiche di filtraggio facilmente consultabili. Ciononostante, questo approccio potrebbe generare dei vantaggi indiretti, derivanti dal fatto che i creatori di malware sarebbero costretti a modificare i propri metodi di attacco, rendendo il loro comportamento più visibile all'interno del traffico di rete. Questo cambiamento, a sua volta, può aiutare i ricercatori, e in generale i difensori, a venire

a conoscenza delle nuove tecniche di evasione messe in campo dai cybercriminali [Mag24].

2.2 Metodologie di reingegnerizzazione software

All'inizio degli anni '90 è emersa la forte necessità di reingegnerizzazione, alimentata soprattutto da tutti coloro che stavano migrando i sistemi informativi aziendali verso interfacce Web più moderne [MJS+00]. Ciò ha fatto si che il campo della reingegnerizzazione si sviluppasse per affrontare i problemi legati ai sistemi software divenuti ormai obsoleti. Tali sistemi, definiti con la parola inglese *legacy*, risultavano tuttavia fondamentali per le operazioni e i processi aziendali. Di conseguenza, da allora fino a oggi, i ricercatori stanno lavorando per progettare dei modelli e dei piani d'azione che siano flessibili e riutilizzabili per poter essere applicati al processo di reingegnerizzazione dei software legacy [MQO18].

In questa sezione verranno esaminati i concetti chiave e gli obiettivi della reingegnerizzazione, seguiti dalla descrizione di un modello generale che ne rappresenta l'architettura. Successivamente, saranno esplorati gli approcci più comuni e le fasi operative che caratterizzano questo processo. Infine, si fornirà una breve panoramica sui principali rischi associati alla reingegnerizzazione.

2.2.1 Definizione e obiettivi

Sono stati Chikofsky e Cross tra i primi ricercatori a inquadrare la reingegnerizzazione in maniera chiara e strutturata. La loro interessante definizione descrive questo concetto come, testualmente: "*the examination and alteration of software systems to reconstitute it in new form and the subsequent implementation of the new form*" [CI90]. In termini più attuali, la reingegnerizzazione software si definisce come il processo di riprogettazione e reimplementazione di sistemi legacy con l'obiettivo di migliorarne la manutenibilità. Ciò richiede un'analisi della documentazione esistente per produrne una nuova ed equivalente, una revisione dell'architettura del sistema per riorganizzarne la struttura, e infine una reimplementazione con tecnologie o linguaggi moderni che soddisfano le esigenze attuali del mercato.

Tutto questo dovrebbe essere fatto cercando di mantenere quante più funzionalità possibili del sistema rivisitato [Som11].

A differenza dell'ingegneria del software, che parte da specifiche e conduce alla creazione di un sistema attraverso design e implementazione, la reingegnerizzazione parte da un software esistente, che viene analizzato e trasformato per derivare un sistema target. Questa distinzione non è solo teorica, ma trova applicazione concreta in una serie di situazioni che rendono la reingegnerizzazione una scelta indispensabile. Più nel dettaglio, questa pratica risulta particolarmente necessaria:

- quando il codice sorgente non ha più una struttura pulita e chiara, unito al fatto che la documentazione potrebbe non esistere;
- quando il supporto hardware e software nel sistema attuale diventa obsoleto e superato a causa di cambiamenti nelle politiche organizzative o nella competizione del mercato;
- quando i sistemi legacy, dopo anni di modifiche, diventano difficili e onerosi da modificare.

Obiettivi della reingegnerizzazione

Sebbene gli obiettivi del processo di reingegnerizzazione possano variare sulla base dello scopo delle organizzazioni che lo attuano, è possibile individuarne 4 che risultano generali e applicabili in ogni situazione [RH96]:

1. **Preparazione per l'ampliamento funzionale:** la reingegnerizzazione non dovrebbe avere lo scopo diretto di migliorare le funzionalità di un sistema esistente, ma, piuttosto, di prepararlo per futuri miglioramenti. Questo include l'analisi delle attuali caratteristiche del sistema legacy per confrontarle con le specifiche desiderate, consentendo la costruzione di un sistema target più flessibile. Ad esempio, se le migliorie desiderate seguono il paradigma di programmazione orientato agli oggetti, il sistema target può essere riprogettato utilizzando tecnologie orientate agli oggetti per semplificare il processo di incremento delle funzionalità.

2. **Miglioramento della manutenibilità:** con il passare del tempo, i sistemi tendono a diventare sempre più complessi e costosi da mantenere a causa della struttura del codice poco chiara e della documentazione assente od obsoleta. La reingegnerizzazione mira a ridisegnare il sistema suddividendolo in moduli funzionali meglio definiti e con interfacce esplicite. Questo processo include l'aggiornamento della documentazione interna ed esterna, facilitando così gli interventi futuri.

3. **Migrazione:** l'industria informatica è in continua evoluzione, con l'introduzione di nuove tecnologie che spesso superano e rendono obsolete quelle esistenti in tempi molto brevi. La migrazione può quindi diventare necessaria per mantenere la compatibilità con i nuovi sistemi, per sfruttare hardware e software moderni, e per evitare costi crescenti legati alla manutenzione di tecnologie superate. La reingegnerizzazione facilita questa transizione, progettando e applicando una migrazione controllata verso nuove piattaforme hardware, sistemi operativi più moderni o linguaggi più attuali.

4. **Miglioramento dell'affidabilità:** i sistemi legacy, nella maggior parte dei casi, non hanno mai raggiunto un livello di affidabilità particolarmente elevato. Nel tempo, le molteplici modifiche apportate a tali sistemi hanno spesso generato i cosiddetti "effetti a catena", ovvero situazioni in cui un intervento introduce involontariamente nuovi problemi. Per questo motivo, uno degli obiettivi principali della reingegnerizzazione è ottenere, nel sistema target, un livello di affidabilità significativamente superiore rispetto a quello del sistema originario.

2.2.2 Modello generale

Il processo di reingegnerizzazione del software si basa su un modello generale che definisce come un sistema legacy viene trasformato in un nuovo sistema. Questo modello si articola nei tre principi fondamentali *abstraction*, *alteration* e *refinement*, che vengono di seguito analizzati [RH96]:

- **Abstraction** si riferisce al processo di aumento graduale del livello di astrazione di un sistema. Questo avviene sostituendo le informazioni dettagliate

esistenti con informazioni più generali e astratte, creando una rappresentazione che enfatizza alcune caratteristiche del sistema, andando a eliminarne altre. L'obiettivo è comprendere meglio la struttura e il funzionamento del sistema, mettendo in evidenza solo gli aspetti più rilevanti. Questo processo è parte della reverse engineering, supportata da specifici strumenti e tecniche.

- **Alteration** riguarda l'apportare una o più modifiche alla rappresentazione di un sistema senza alterarne il livello di astrazione. Ad esempio, è possibile modificare il design di un componente software senza alterare i requisiti funzionali del sistema.

- **Refinement** è il processo inverso dell'abstraction, ovvero la discesa graduale verso un livello di astrazione più basso. Partendo da una rappresentazione astratta del sistema target, si procede verso la sua implementazione concreta, definendo dettagli sempre più specifici. Questo processo, noto come forward engineering, ricalca in parte lo sviluppo tradizionale del software, ma con alcune differenze dovute al fatto che si parte da un sistema già in essere.

Il modello generale illustra come, a seconda dell'obiettivo della reingegnerizzazione, si possa intervenire su diversi livelli di astrazione. Ad esempio, se si vuole solo tradurre il codice da un linguaggio a un altro, non è necessaria la reverse engineering, poiché il lavoro si svolge già al livello di astrazione più basso, ovvero l'implementazione. Invece, se si vuole riprogettare l'architettura del sistema, il processo di reverse engineering diventa fondamentale e dovrebbe arrivare fino al livello dei requisiti, in modo da ricavare le caratteristiche funzionali dal design e dall'implementazione.

Nel corso di questa sezione si è fatto riferimento più volte ai concetti di reverse engineering e forward engineering, senza tuttavia approfondirne i dettagli. Di seguito, si propone una breve descrizione di queste tecniche, al fine di chiarirne il ruolo e le caratteristiche principali nel contesto attuale:

- Il **Reverse Engineering** è un processo fondamentale nell'ambito in questione. Il suo scopo è quello di analizzare il sistema legacy per identificarne i componenti, le loro interrelazioni e creare rappresentazioni a un livello di

astrazione superiore. In pratica, si tratta di recuperare informazioni dal codice sorgente, dalla documentazione esistente e dagli esperti del sistema per comprendere appieno la struttura del software, i requisiti e il funzionamento.

- Il **Forward Engineering**, invece, è il processo di creazione di un nuovo sistema passando dai livelli di astrazione più alti a quelli più bassi, andando a sostituire gradualmente le informazioni astratte con dettagli più specifici. Questo movimento verso il basso corrisponde al normale ciclo di sviluppo software, che passa dai requisiti al design, fino ad arrivare all'implementazione fisica del sistema.

2.2.3 Approcci e fasi del processo

Esistono tre doversi approcci alla reingegnerizzazione del software, i quali si differenziano per la quantità e la velocità con cui il sistema legacy viene sostituito con il sistema target. Ogni approccio è caratterizzato da vantaggi e rischi specifici, che vengono ora discussi nel dettaglio [MQO18, RH96]:

1. **Big Bang**: questo approccio prevede la sostituzione dell'intero sistema in un unico momento, e di solito viene messo in campo per affrontare problemi immediati come una migrazione di architettura. Il vantaggio principale è che il sistema viene trasferito in un nuovo ambiente senza dover sviluppare interfacce tra i componenti vecchi e nuovi. Lo svantaggio, però, è che il risultato finale tende a essere un progetto monolitico, caratteristica non sempre adeguata, soprattutto per i sistemi di grandi dimensioni. Inoltre, le modifiche apportate al sistema legacy durante la transizione devono essere integrate nel nuovo, rendendo difficile mantenere l'allineamento e preservare tutte le funzionalità esistenti. In più, il rischio di questo approccio risulta elevato in quanto il nuovo sistema deve essere funzionalmente intatto e operare in parallelo con quello vecchio fino alla completa sostituzione.

2. **Incrementale**: l'approccio in questione prevede la reingegnerizzazione graduale di sezioni del sistema, che vengono integrate in nuove versioni quando sopraggiunge la necessità di soddisfare nuovi obiettivi. Questo metodo suddivide il progetto in porzioni più contenute basate sulla struttura esistente del

sistema, permettendo così uno sviluppo più rapido delle componenti e una maggiore facilità nel tracciare gli errori. Inoltre, gli stakeholders possono monitorare i progressi attraverso le versioni intermedie e segnalare tempestivamente eventuali funzionalità mancanti o problemi individuati. Tuttavia, questo approccio richiede tempi più lunghi per il completamento del sistema, dato che le versioni intermedie necessitano di un controllo accurato della configurazione. Inoltre, non permette una completa ristrutturazione dell'architettura, limitandosi alle modifiche interne delle sezioni reingegnerizzate. Nonostante ciò, il rischio complessivo è inferiore rispetto all'approccio "Big Bang", poiché i rischi legati a ciascuna componente possono essere gestiti e monitorati separatamente.

3. **Evolutivo**: l'approccio "Evolutivo" prevede la sostituzione graduale delle sezioni del sistema originale con parti reingegnerizzate, selezionate in base alla loro funzionalità piuttosto che alla struttura del sistema esistente (come invece avviene nell'approccio "Incrementale"). Il sistema target viene costruito utilizzando moduli coesi dal punto di vista funzionale, riorganizzando le componenti attuali per funzioni specifiche. Tra i vantaggi principali vi sono un design modulare che facilita la manutenzione e un focus su unità funzionali più semplici, che rende particolarmente agevole l'adozione di tecnologie orientate agli oggetti. Tuttavia, questo approccio richiede un'attenta identificazione delle funzioni simili, spesso distribuite nel sistema originale, e può introdurre difficoltà di integrazione tra le componenti, con possibili peggioramenti delle prestazioni dovuti alla reingegnerizzazione isolata delle sezioni funzionali.

Dopo aver analizzato i principali approcci adottati nella reingegnerizzazione, è importante approfondire le fasi fondamentali che guidano la trasformazione del sistema legacy in un nuovo sistema. Queste fasi, articolate in cinque passaggi chiave, hanno compiti specifici volti a garantire una transizione efficace verso il sistema target [RH96]:

1. **Formazione del team di reingegnerizzazione**: il team guiderà l'intero processo, richiedendo una formazione approfondita sulla gestione del cambiamento tecnologico, sui principi della reingegnerizzazione e sui processi di

sviluppo del sistema target. Le loro attività comprenderanno la definizione di obiettivi, strategie e piani d'azione, oltre all'identificazione e all'acquisto di strumenti adeguati. Sarà loro compito promuovere il progetto all'interno dell'organizzazione, offrire supporto al personale e garantire l'applicazione corretta del processo. Infine, saranno essenziali buone capacità relazionali per affrontare eventuali resistenze al cambiamento.

2. **Analisi di fattibilità del progetto**: il primo compito del team di reingegnerizzazione è valutare i bisogni organizzativi e gli obiettivi soddisfatti dal sistema esistente, garantendo che la strategia sia allineata alle esigenze aziendali. È fondamentale analizzare il software attuale per identificare problemi, scopi e vincoli, stimando il ritorno sull'investimento. Gli obiettivi, come il miglioramento delle prestazioni o la riduzione dei costi, devono essere misurabili e confrontati con i costi della reingegnerizzazione e il valore aggiunto previsto.

3. **Analisi e pianificazione**: questa fase si articola in tre passaggi principali: l'analisi del sistema legacy, la definizione delle caratteristiche del sistema target e la creazione di un ambiente di test. L'analisi inizia con la raccolta e lo studio del codice e della documentazione disponibili, al fine di identificare peculiarità e problemi di qualità del software esistente. Successivamente, vengono definite le modifiche richieste, come la piattaforma hardware, il sistema operativo, la struttura del design e il linguaggio. Infine, si sviluppa un ambiente di test standard, necessario per garantire l'equivalenza funzionale tra il nuovo sistema e quello originale, assicurando la tracciabilità delle funzioni esistenti.

4. **Implementazione della reingegnerizzazione**: dopo che sono stati definiti gli obiettivi e l'approccio, e dopo aver analizzato il sistema legacy, iniziano le fasi di reverse e forward engineering. La prima scompone le funzioni del sistema legacy fino a raggiungere il livello di astrazione desiderato, utilizzando strumenti che siano compatibili con gli obiettivi. In seguito, la seconda riprende dal livello raggiunto per sviluppare il nuovo design, seguendo il normale processo di sviluppo software. Durante questa fase è

fondamentale evitare modifiche o aumenti di funzionalità per semplificare la validazione, applicando tecniche per il controllo della qualità, e monitorando continuamente i miglioramenti e i potenziali rischi.

5. **Test e transizione**: man mano che le funzionalità del nuovo sistema aumentano, è necessario effettuare test per individuare eventuali errori introdotti durante la reingegnerizzazione. Le tecniche di testing sono le stesse utilizzate per lo sviluppo di un sistema da zero. Se i requisiti del nuovo sistema coincidono con quelli del legacy, è possibile riutilizzare i test e l'ambiente sviluppati durante la fase di pianificazione, confrontando i risultati per verificare la funzionalità del sistema target. Inoltre, la documentazione del sistema legacy deve essere aggiornata o sostituita per riflettere il nuovo sistema e fornire le informazioni necessarie per la sua gestione e manutenzione.

Seguire queste fasi in modo strutturato è cruciale per garantire il successo di un progetto di reingegnerizzazione del software, riducendo i rischi e ottimizzando l'utilizzo di tempo e risorse.

2.2.4 Panoramica sui rischi

Per concludere la sezione corrente e fornire un quadro ancora più completo sulla reingegnerizzazione, si ritiene opportuno fare una breve panoramica sui potenziali rischi che questo importante processo si porta dietro, soprattutto se non viene gestito in maniera appropriata.

La reingegnerizzazione del software, pur offrendo vantaggi significativi, comporta rischi che devono essere identificati e gestiti con attenzione per garantire il successo del progetto. Questi rischi si manifestano in diverse aree chiave, tra cui il processo, il reverse engineering, il forward engineering, il personale, gli strumenti e la strategia [RH96].

- Un rischio rilevante riguarda l'elevato costo del **processo** di reingegnerizzazione manuale, che potrebbe non garantire benefici proporzionati nei tempi previsti. Inoltre, la mancanza di supporto da parte del management o una selezione inadeguata dei sottosistemi da reingegnerizzare possono compromettere il progetto. Altri rischi includono una gestione insufficiente della

configurazione, l'assenza del controllo qualità o di un programma di metriche adeguato, e la carenza di esperti con conoscenze sugli applicativi locali.

- Nel **reverse engineering**, una delle principali difficoltà è il recupero di requisiti e informazioni di progettazione dal codice sorgente, specialmente se il linguaggio utilizzato non consente una rappresentazione astratta efficace. La perdita di informazioni aziendali incorporate nel codice costituisce un ulteriore rischio critico.

- Anche il **forward engineering** presenta sfide significative. L'introduzione di nuovi requisiti o funzionalità può complicare il processo di validazione, mentre la migrazione dei dati esistenti e una preparazione insufficiente durante il reverse engineering possono influire negativamente sull'esito del progetto.

- I principali rischi legati al **personale** riguardano la mancanza di competenze adeguate nella reingegnerizzazione e la scarsa esperienza, che possono influire negativamente sull'efficacia del progetto. Inoltre, i programmatori potrebbero lavorare in modo volutamente meno efficiente per ostacolare un'iniziativa percepita come impopolare.

- Le problematiche connesse agli **strumenti** includono la dipendenza da soluzioni che non funzionano come previsto o che risultano nuove e immature. Anche la stabilità dei fornitori e la qualità complessiva degli strumenti possono rappresentare un rischio significativo.

- I rischi **strategici** includono l'impegno prematuro in una soluzione di reingegnerizzazione per l'intero sistema, la mancanza di una visione a lungo termine e l'adozione di obiettivi irrealistici o poco solidi. Altri problemi possono derivare dalla scelta di approcci che non rispettano gli scopi aziendali, i limiti di budget o le scadenze, oltre che dall'assenza di un piano per l'utilizzo degli strumenti di reingegnerizzazione.

Capitolo 3

Analisi

Il presente capitolo fornisce un'analisi introduttiva sul sistema legacy, descrivendone le funzionalità principali, l'architettura e le tecnologie impiegate. Verranno evidenziati i limiti strutturali e operativi che hanno portato l'azienda a intraprendere un processo di reingegnerizzazione, con l'obiettivo di sviluppare un sistema più moderno, scalabile e manutenibile.

Un aspetto cruciale di questa analisi riguarda la strategia di transizione dal sistema legacy al nuovo pannello di configurazione. Inizialmente era stata prevista una coesistenza graduale tra i due sistemi, ma la direzione aziendale ha successivamente deciso di adottare un approccio differente. Il nuovo sistema verrà sviluppato fino a includere tutte le macrofunzionalità fondamentali, che gli garantiranno una quasi completa operatività. Inoltre, sarà introdotta una nuova funzionalità che, oltre a rappresentare un'evoluzione rispetto al legacy, contribuirà a differenziare il nuovo sistema e a valorizzarne le potenzialità. Solo una volta completato, il nuovo pannello sostituirà integralmente il legacy con una transizione netta, eliminando quindi la necessità di retrocompatibilità e riducendo la complessità della fase transitoria.

Proseguendo, il capitolo illustrerà gli obiettivi e i requisiti principali che hanno guidato la progettazione del nuovo sistema, preparando la discussione del capitolo successivo, in cui verranno descritte nel dettaglio le scelte architetturali adottate per superare le limitazioni del sistema legacy e rispondere alle nuove esigenze aziendali e di mercato.

Infine, la parte finale del capitolo sarà dedicata all'analisi del dominio applicativo, con l'obiettivo di identificare i concetti e le entità fondamentali del sistema legacy che verranno utilizzati per la progettazione del nuovo sistema. Questa analisi è essenziale per definire una base strutturata su cui modellare le funzionalità del nuovo pannello di configurazione, garantendo una transizione coerente e ben organizzata dalle logiche del sistema precedente a quelle della nuova implementazione.

3.1 Introduzione al sistema esistente

Il sistema legacy qui esaminato rappresenta un pannello Web utilizzato dall'azienda *FlashStart Group srl*[1] per la gestione e la configurazione del suo filtro DNS. L'analisi in questione, effettuata nell'ambito di un tirocinio in preparazione della presente tesi, si basa sulle osservazioni dirette fatte durante lo sviluppo del nuovo sistema, dato che la documentazione del legacy risulta pressoché assente.

Come illustrato in Figura 3.1, il pannello legacy offre una dashboard che riporta statistiche sulle richieste DNS processate, suddivise tra richieste bloccate, consentite e non risolte, con grafici relativi alle categorie più filtrate e ai servizi più bloccati. L'interfaccia presenta un'organizzazione a schede, con una barra superiore per la selezione del cliente e dei profili associati. Nonostante le funzionalità offerte, il design risulta datato e poco curato dal punto di vista estetico, con un layout che non trasmette immediatamente la professionalità del prodotto e non valorizza l'esperienza utente.

Terminologia e definizioni

Da qui in avanti verranno utilizzate alcune terminologie specifiche che descrivono concetti e funzionalità del sistema legacy. Pertanto, prima di proseguire con la sua analisi, è opportuno fornire una breve spiegazione di questi termini, utilizzati dall'azienda per rappresentare concetti chiave del dominio applicativo.

[1] `https://flashstart.com`

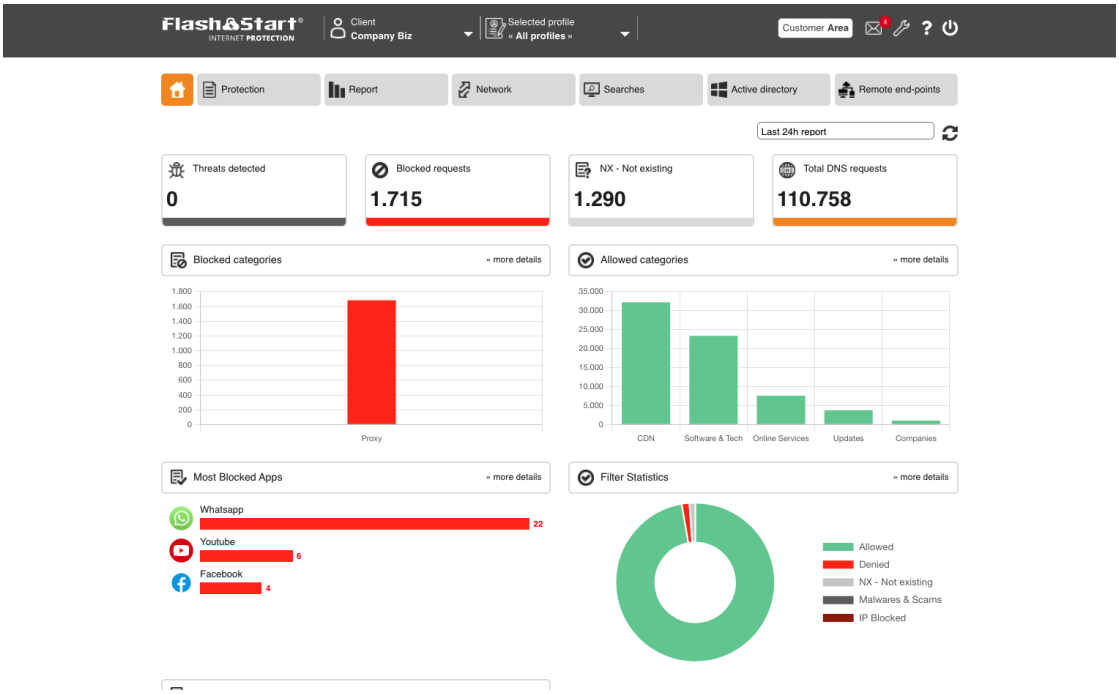

Figura 3.1: Schermata principale del pannello Web legacy di FlashStart.

- **Rete**: identifica un indirizzo IP registrato nel sistema e associato alla sede di un cliente, come ad esempio una scuola. Le reti possono essere statiche (IP pubblico fisso) o dinamiche (IP pubblico variabile).

- **Endpoint**: rappresenta un dispositivo fisico specifico registrato nel sistema, come un computer o uno smartphone. A differenza della rete, un endpoint è associato direttamente a un dispositivo, piuttosto che a un indirizzo IP generico.

- **Dealer**: un rivenditore che vende licenze del filtro DNS ai clienti finali, i quali devono, in autonomia, configurare le proprie policy di sicurezza tramite il pannello Web.

- **Managed Service Provider (MSP)**: è un particolare tipo di Dealer che, oltre a vendere licenze, si occupa anche della configurazione e della gestione delle policy di sicurezza per i propri clienti. A differenza dei precedenti for-

nitori di servizi, gli MSP impediscono ai clienti finali di accedere al pannello, occupandosi interamente della gestione.

- **Whitelabel**: una funzionalità che consente ai fornitori di servizi, come i Dealer, di personalizzare il pannello Web con il proprio logo, colori e branding, nascondendo ogni riferimento al produttore originale del sistema.
- **ClientShield**: è un'applicazione per computer e dispositivi mobili sviluppata dall'azienda, che consente di registrare un endpoint nel sistema e sfruttare il filtro DNS anche al di fuori della rete usuale.

3.1.1 Panoramica sulle funzionalità

Il pannello Web legacy rappresenta lo strumento principale per la gestione e configurazione del filtro DNS offerto dall'azienda. Nonostante le limitazioni architetturali e tecnologiche, il sistema fornisce un insieme di funzionalità che consentono agli utenti di configurare, monitorare e amministrare le policy di filtraggio DNS. Di seguito viene fornita una panoramica delle principali funzionalità offerte dal suddetto pannello.

Gestione delle policy di protezione

Il pannello consente di creare e configurare diversi profili di protezione, ciascuno dei quali può includere una combinazione personalizzata di filtri per:

- Bloccare minacce informatiche come malware e phishing;
- Limitare l'accesso a contenuti specifici, come siti per adulti o contenuti inappropriati;
- Bloccare l'accesso ad applicazioni o servizi specifici, divisi per categoria di appartenenza;
- Bloccare l'accesso a pagine Web e servizi provenienti da determinate aree geografiche.

Per aumentare il grado di flessibilità del filtro, gli utenti possono creare delle liste di accesso personalizzate, tra cui:

- **Allow list**: per consentire l'accesso a domini specifici;
- **Block list**: per bloccare domini o indirizzi IP specifici.

Queste liste possiedono una priorità più elevata rispetto alle funzioni di protezione citate in precedenza. Per questo motivo, esse consentono di specificare delle *eccezioni* rispetto alle normali policy di sicurezza. Ad esempio, è possibile concedere l'accesso a un contenuto di norma non consentito, oppure bloccare un dominio che risulta legittimo.

Gestione delle reti di protezione

Un'altra funzionalità fondamentale del pannello è la possibilità di specificare quali reti devono essere sottoposte al filtraggio, garantendo un controllo preciso e mirato sulle attività DNS. Questo permette di configurare reti aziendali o domestiche in modo che tutte le richieste DNS generate da tali indirizzi IP passino attraverso le policy di protezione impostate. La configurazione delle reti di protezione può essere adattata a diverse esigenze, supportando due principali modalità:

- **IP pubblico statico**: questa configurazione è utilizzata quando la rete dispone di un IP pubblico statico, ovvero un indirizzo IP assegnato in modo permanente dal proprio ISP. In questo caso, il pannello consente di associare le policy di filtraggio a una rete identificata da uno specifico IP, garantendo che tutte le richieste DNS provenienti da tale rete siano sottoposte ai controlli e ai filtri impostati;
- **IP pubblico dinamico**: per le reti che non dispongono di un IP pubblico statico, il sistema supporta la configurazione tramite DynamicDNS[2] (DDNS). Questo approccio consente di monitorare e filtrare le richieste DNS anche quando l'indirizzo IP della rete varia nel tempo, utilizzando un sistema di aggiornamento dinamico che associa un nome di dominio all'IP corrente della rete. Questo garantisce continuità nella protezione senza la necessità di aggiornamenti manuali.

[2] `https://www.rfc-editor.org/rfc/rfc2136.html`

Visualizzazione e analisi dei report

Il pannello offre una sezione dedicata alla generazione e analisi dei report relativi al traffico DNS della rete, permettendo di monitorare l'efficacia delle policy di filtraggio e di ottenere informazioni dettagliate sull'attività di rete in un determinato intervallo di tempo. Questi report forniscono una visione chiara e organizzata del comportamento della rete, aiutando gli utenti a identificare potenziali minacce e a ottimizzare le configurazioni esistenti.

Tipologie di report disponibili Tramite un menu a cascata, è possibile selezionare diversi tipi di report, tra cui:

- **Bloccati per categoria**: mostrano le richieste DNS bloccate verso siti indesiderati, raggruppandole per categoria o macrocategoria;
- **Consentiti per paese o categoria**: forniscono il numero di richieste consentite, organizzate per paese o per categoria di contenuti;
- **Malware e minacce bloccate**: presentano un'analisi delle richieste che hanno attivato il filtro, indicando malware o altre minacce bloccate;
- **Traffico per fasce orarie o giorni**: permettono di analizzare le richieste DNS effettuate in specifiche fasce orarie o giorni della settimana;
- **Report geografici**: forniscono una mappa del mondo che evidenzia il traffico DNS suddiviso per paesi e continenti.

Dopo aver configurato i parametri di analisi, gli utenti possono generare i report secondo diverse modalità. Essi possono essere esportati in formato PDF, oppure inviati direttamente via e-mail a destinatari predefiniti. Inoltre, il pannello offre una funzione di pianificazione che consente di programmare l'invio automatico dei report a intervalli regolari, ad esempio su base settimanale, rendendo più efficiente il monitoraggio continuo.

Gestione dei dispositivi protetti (Endpoint)

Il pannello include anche una funzionalità per gestire i dispositivi su cui è installato il ClientShield. In particolare, questa configurazione permette di tracciare con

precisione quale dispositivo ha originato una determinata richiesta DNS, fornendo un maggiore controllo e un livello di dettaglio maggiore sulle attività della rete.

Funzionalità aggiuntive

Oltre alle funzionalità principali già descritte, il pannello Web offre una serie di strumenti utili per migliorare la gestione e il controllo delle configurazioni. Tra queste, vi è la possibilità di personalizzare la pagina di blocco che viene visualizzata dagli utenti ogni volta che tentano di accedere a un dominio non consentito. Il pannello consente anche di verificare facilmente a quale categoria appartiene un determinato sito Web, aiutando gli utenti a valutare come configurare al meglio le politiche di filtraggio. Un'altra funzionalità interessante è la visualizzazione del traffico DNS in tempo reale, che fornisce un monitoraggio immediato dell'attività di rete. Inoltre, il sistema supporta l'importazione di domini in formato batch, permettendo di aggiungere rapidamente liste di siti Web consentiti o bloccati attraverso file di testo.

Per concludere la panoramica sulle funzionalità, il pannello possiede un'importante integrazione con la tecnologia Active Directory di Microsoft, che consente di ottenere informazioni dettagliate non solo sul dispositivo che ha originato una determinata richiesta DNS, ma anche sull'utente utilizzato per accedere a tale macchina.

3.1.2 Architettura e tecnologie utilizzate

Il pannello Web in esame, fin'ora descritto solo dal punto di vista delle funzionalità, presenta un'architettura monolitica, tipica dei vecchi sistemi Web-based. Nonostante sia possibile identificare due macrosezioni, denominate *Customer area* e *Pannello cloud*, non vi è una reale modularizzazione del frontend e del backend. Tutto il codice risulta scritto in modo procedurale, senza l'adozione di un paradigma orientato agli oggetti, e con una scarsa separazione delle responsabilità.

Tecnologie utilizzate

Il backend è interamente sviluppato in PHP, utilizzando un approccio "plain", ossia privo di framework moderni come Laravel[3]. Per il frontend sono stati utilizzati HTML, CSS e una versione obsoleta di jQuery[4] (`1.x`), che limita le possibilità di modernizzazione dell'interfaccia. In alcuni casi, il codice PHP si occupa anche di generare dinamicamente script JavaScript, i quali eseguono ulteriori chiamate a codice PHP lato server.

Integrazione con API esterne

L'azienda ha sviluppato un set di API pubbliche che consentono di gestire il filtro DNS senza dover necessariamente utilizzare il pannello Web in questione. Queste API offrono agli utenti la possibilità di integrare il filtro DNS in applicazioni personalizzate o di sviluppare un proprio client per la gestione delle configurazioni.

Nel tempo, alcune operazioni che il sistema legacy eseguiva direttamente sono state trasferite alle suddette API, le quali centralizzano la logica di business e gestiscono l'interazione con il database. Il pannello Web, in questi casi, funge da semplice interfaccia per chiamare le API. Tuttavia, molte operazioni continuano a risiedere direttamente sul sistema, implementando la logica applicativa e accedendo alla base dati.

Gestione del database

Il sistema legacy utilizza due database distinti per gestire le sue funzionalità e garantire la persistenza delle configurazioni. Entrambi adottano un motore di database *relazionale*, che organizza i dati in tabelle collegate tra loro tramite un sistema di chiavi. La differenza principale risiede nella tecnologia sottostante: una base dati utilizza MySQL[5], mentre l'altra è basata su PostgreSQL[6]. Ciascuna di esse è destinata a scopi specifici e presenta caratteristiche diverse in termini di configurazione e prestazioni.

[3] `https://laravel.com`
[4] `https://jquery.com`
[5] `https://www.mysql.com`
[6] `https://www.postgresql.org`

Il database MySQL è dedicato esclusivamente alla gestione delle licenze e dei dati anagrafici dei clienti. Esso viene ospitato su un server interno all'azienda e non è replicato in altre regioni. Questa configurazione rappresenta un collo di bottiglia significativo per gli utenti che sono distanti dalla sede aziendale, poiché tutte le richieste relative ai dati dei clienti o delle licenze (usate soprattutto nella fase di accesso al pannello) devono necessariamente essere inviate al server centrale per essere elaborate, causando latenze elevate.

Il database PostgreSQL, invece, è utilizzato per tutti gli altri dati, inclusi i report, le regole di protezione e le liste dei domini da bloccare. Questo database è configurato in replica globale, garantendo così prestazioni più elevate e tempi di risposta migliori per i clienti situati in diverse aree geografiche. Grazie a questa configurazione, i dati necessari al funzionamento del filtro DNS possono essere accessibili rapidamente da qualunque parte del mondo.

Un aspetto di fondamentale importanza per l'azienda è rappresentato dal contenuto del suddetto database, che costituisce un valore strategico significativo. Al suo interno, infatti, è presente una lista dei domini associati alla relativa categoria di appartenenza. Questa categorizzazione è utilizzata direttamente dal filtro DNS per bloccare l'accesso a determinati siti in base alle regole configurate dagli utenti. L'operazione di categorizzazione viene gestita internamente all'azienda e si basa su un approccio di intelligenza artificiale. Tale sistema analizza i testi delle pagine Web e determina automaticamente la categoria di appartenenza di ciascun dominio, migliorando l'efficacia del filtro DNS e arricchendo continuamente il patrimonio informativo dell'azienda.

3.1.3 Limitazioni riscontrate

Il sistema legacy presenta numerose limitazioni che hanno reso necessaria una completa reingegnerizzazione. Queste riguardano sia l'assenza di funzionalità fondamentali, come il supporto alla multiutenza, sia problematiche strutturali e di usabilità che compromettono la flessibilità e l'efficienza operativa dello stesso.

Mancanza del supporto alla multiutenza

Una delle principali limitazioni del sistema legacy, e tra quelle più sentite dai clienti dell'azienda, è l'assenza di un supporto per la multiutenza. Attualmente, infatti, il sistema permette a ciascun cliente di disporre di un unico account per accedere al pannello di configurazione, che possiede i privilegi di amministratore.

Questa mancanza rappresenta un ostacolo significativo, soprattutto per i Dealer o gli MSP che integrano il filtro DNS in altri prodotti o lo rivendono ad aziende terze. Questi ultimi si trovano spesso a dover gestire configurazioni e politiche di protezione per conto dei loro clienti, ma l'assenza di un sistema multiutente impedisce di delegare determinate operazioni o di offrire accesso limitato a figure specifiche all'interno delle organizzazioni servite. Allo stesso modo, chi utilizza direttamente il pannello non può creare profili con permessi ridotti, ad esempio per utenti che necessitano soltanto di monitorare le configurazioni o consultare i report senza possibilità di modificarli.

Implementare la funzionalità in questione nel sistema attuale richiederebbe modifiche strutturali profonde, che non sono realisticamente attuabili senza un suo completo stravolgimento.

Profili di protezione non condivisibili

Un'altra significativa limitazione del sistema legacy riguarda l'impossibilità di condividere i profili di protezione tra diversi utenti. Attualmente, ogni profilo è strettamente associato a un singolo cliente, senza alcuna possibilità di essere condiviso o ereditato da altri. Questa carenza rappresenta un ostacolo rilevante, specialmente per gli MSP che gestiscono clienti con esigenze simili, come un gruppo di scuole o aziende dello stesso settore. In tali casi, sarebbe estremamente utile disporre di profili condivisi che consentano di applicare la stessa configurazione a più clienti contemporaneamente. Inoltre, la mancanza di questa funzionalità aumenta il carico di lavoro in caso di modifiche alle regole di protezione. La situazione attuale, infatti, è tale per cui ogni variazione deve essere riportata manualmente su ciascun cliente, imponendo un processo lungo e soggetto a errori.

Limiti e debolezze architetturali

Il sistema legacy presenta numerosi limiti dovuti a scelte architetturali e tecnologiche datate, che influiscono negativamente sulla manutenibilità e sull'evoluzione del software. La mancanza di una chiara separazione tra frontend e backend complica la gestione del codice, rendendo difficoltosa l'adozione di nuove tecnologie. Oltretutto, pratiche quali l'utilizzo di PHP per generare dinamicamente codice JavaScript introducono ulteriore complessità e opacità dei sorgenti, limitando la modularità e aumentando il rischio di commettere errori.

L'architettura monolitica rappresenta una delle principali debolezze del sistema. Qualsiasi modifica, anche minima, comporta interventi che possono avere ripercussioni su altre parti del codice, a causa dell'assenza di un design accurato e di una netta separazione delle responsabilità. Questo approccio non solo rallenta il ciclo di sviluppo, ma aumenta significativamente il rischio di regressioni e rende difficoltoso il debugging. La mancanza di modularità aggrava ulteriormente il problema: ogni nuova funzionalità o aggiornamento richiede un lavoro complesso e rischioso, che spesso si traduce in un incremento della fragilità del sistema.

Problemi di usabilità

L'interfaccia utente del sistema, sviluppata con tecnologie ormai obsolete, presenta diverse limitazioni che compromettono l'esperienza degli utenti finali. L'utilizzo di una versione datata di jQuery, unito all'assenza di un layout moderno, rende l'interfaccia poco intuitiva e difficile da navigare. Questi problemi non solo riducono l'efficienza operativa degli utenti, ma influenzano negativamente anche la percezione complessiva del sistema.

Un design grafico superato e poco efficiente limita, inoltre, la capacità del sistema di competere con soluzioni contemporanee, riducendone l'attrattiva sia per gli utenti attuali che per potenziali nuovi clienti.

Conclusioni

Le limitazioni evidenziate, che includono l'assenza di funzionalità essenziali come la multiutenza e i profili condivisibili, le debolezze architetturali e i problemi legati all'usabilità e all'interfaccia utente, mettono in luce la rigidità e l'obsolescenza del

sistema legacy. Tali carenze ostacolano non solo l'efficienza operativa e la scalabilità del sistema, ma anche la capacità dell'azienda di rispondere alle richieste di mercato e di competere con soluzioni moderne. Questi fattori rendono indispensabile una completa reingegnerizzazione per soddisfare le esigenze attuali e future, garantendo al contempo un sistema moderno, scalabile ed efficiente.

3.2 Esigenza di transizione tra i due sistemi

La transizione dal sistema legacy al nuovo pannello di configurazione seguirà una strategia differente rispetto a quanto inizialmente previsto. Non sarà più adottato un approccio di coesistenza graduale tra i due sistemi, bensì una transizione netta non appena il nuovo sistema sarà pronto per il rilascio. Durante la fase di sviluppo, il vecchio pannello rimarrà l'unico utilizzato dagli utenti, mentre il nuovo verrà completato fino a includere tutte le macrofunzionalità fondamentali necessarie per garantirne l'operatività, oltre a una nuova funzionalità dedicata alla gestione dei profili condivisi. Solo a quel punto avverrà il passaggio definitivo al nuovo sistema.

Poiché gli utenti si ritroveranno i propri dati direttamente nel nuovo pannello, la migrazione delle informazioni dal legacy al nuovo sistema diventa un requisito fondamentale. Questo processo avverrà internamente all'azienda e dovrà garantire che tutti i dati necessari alle funzionalità trasferite siano disponibili nel nuovo sistema prima del rilascio. Le strategie di migrazione adottate dovranno assicurare la completa integrità e coerenza delle informazioni, evitando perdite o incongruenze tra i due ambienti.

Dopo il rilascio, la fase di transizione sarà limitata esclusivamente al periodo necessario per completare il trasferimento delle funzionalità minori ancora rimaste nel legacy. Tuttavia, rispetto all'approccio iniziale, non sarà più richiesta alcuna retrocompatibilità tra i due sistemi. Infatti, le funzionalità che continueranno a essere utilizzate nel legacy fino alla loro migrazione definitiva, non dovranno interagire con il nuovo sistema, evitando così complessità aggiuntive nella gestione dei dati.

Questa strategia di transizione consente di semplificare il passaggio al nuovo sistema, evitando la necessità di mantenere attivi entrambi i pannelli per un lungo periodo. Inoltre, riduce i rischi legati alla gestione della retrocompatibilità e

permette di concentrare gli sforzi sullo sviluppo del nuovo sistema, senza vincoli imposti dall'integrazione con il legacy.

3.3 Analisi del nuovo sistema

3.3.1 Obiettivi

Il nuovo sistema nasce con l'obiettivo di posizionarsi come una soluzione moderna e professionale, capace di soddisfare le esigenze di un mercato in espansione e di attirare anche clienti di grandi dimensioni. Per raggiungere questo scopo è fondamentale che il pannello abbia un'interfaccia più curata e in linea con gli standard richiesti da organizzazioni complesse. Questo aspetto si inserisce in un più ampio percorso di rinnovamento della *brand identity* aziendale, che include un nuovo logo, una palette di colori aggiornata e uno stile comunicativo uniforme.

Dal punto di vista tecnico, il sistema dovrà superare le limitazioni strutturali del legacy, modernizzando l'architettura per renderla più robusta e scalabile, in modo da poter gestire un numero crescente di utenti. Una delle priorità è quella di ridurre i tempi necessari per la manutenzione e l'aggiornamento, facilitando al contempo l'integrazione di nuove funzionalità.

Un obiettivo chiave della prima release del pannello è l'introduzione di funzionalità avanzate pensate per i fornitori di servizi, con particolare attenzione ai Dealer e agli MSP. Tra queste, un ruolo centrale è ricoperto dai profili condivisi, noti anche come *template*, che consentono di definire e applicare regole di protezione comuni a più clienti. Grazie ai template, gli MSP potranno gestire in modo più efficiente le configurazioni, evitando di doverle replicare manualmente per ciascun cliente e permettendo la propagazione automatica delle modifiche. Un'altra caratteristica fondamentale che differenzia questa prima versione da quella legacy è la presenza di una dashboard dedicata ai fornitori di servizi gestiti. Questo strumento consentirà loro di gestire in maniera completa ed efficace i clienti sotto la loro supervisione, offrendo una panoramica dettagliata sul traffico, sulle configurazioni applicate e sugli eventi di sicurezza rilevati. La dashboard sarà progettata per agevolare l'amministrazione su larga scala, semplificando operazioni come la

modifica delle policy di protezione, l'assegnazione dei template e il monitoraggio centralizzato delle reti gestite.

Infine, una funzionalità fondamentale che differenzia questa prima versione da quella legacy è la gestione avanzata della multiutenza, una caratteristica assente nel sistema precedente. Nel nuovo pannello, sarà possibile associare più utenti a un singolo cliente, ciascuno con ruoli e permessi differenziati. Questo rappresenta un miglioramento significativo rispetto al passato, in cui l'accesso al pannello di configurazione era ristretto a un unico account per cliente. Grazie a questa funzione, più operatori all'interno di un'organizzazione potranno collaborare alla gestione delle policy di filtraggio DNS, con livelli di accesso che vanno dall'amministrazione completa alla sola consultazione dei report.

3.3.2 Requisiti

I requisiti riportati di seguito forniscono una panoramica delle funzionalità previste per il nuovo sistema. Tuttavia, solo una parte di essi è stata sviluppata o analizzata direttamente nel contesto del tirocinio in preparazione della presente tesi. Le funzionalità che verranno introdotte successivamente, pur essendo parte integrante del sistema finale, sono incluse in questa sezione per offrire una visione completa, ma non saranno trattate nel dettaglio in questo elaborato.

Requisiti funzionali

- **Operatività del nuovo sistema al rilascio:**
  - Il nuovo sistema deve includere fin dalla prima release tutte le macrofunzionalità necessarie per garantire la piena operatività del pannello di configurazione, in modo da consentire la transizione completa dal sistema legacy.
  - Le funzionalità essenziali devono comprendere la gestione delle reti, la gestione utenti e permessi (compresa la multiutenza), la configurazione delle policy di filtraggio e la reportistica avanzata.
  - Oltre a queste caratteristiche ereditate dal legacy, il nuovo sistema dovrà introdurre almeno una caratteristica distintiva rispetto alla versio-

ne precedente. In particolare, sarà implementata la gestione avanzata dei template, che consentirà di definire profili condivisi per applicare configurazioni comuni a più clienti.

- Prima della dismissione del sistema legacy, tutte le funzionalità previste dovranno essere completamente operative e testate su quello nuovo.

- **Migrazione interna dei dati:**
  - Tutti i dati del sistema legacy devono essere migrati nel nuovo sistema prima del rilascio, garantendo continuità operativa per gli utenti.
  - Il processo di migrazione deve avvenire internamente all'azienda, senza richiedere interventi manuali da parte degli utenti finali.
  - L'integrità e la coerenza dei dati devono essere assicurate, evitando perdite o incongruenze tra il vecchio e il nuovo sistema.

- **Multiutenza e gestione utenti:**
  - Il sistema deve supportare la multiutenza con ruoli differenziati, quali SuperAdmin, Admin e altri con permessi limitati.
  - La creazione e la gestione degli utenti deve essere riservata agli amministratori.

- **Dashboard per MSP:**
  - Deve essere presente una schermata dedicata agli MSP, che fornisce loro una panoramica dei clienti gestiti.
  - Tale panoramica deve includere report e statistiche aggregate che consentano di monitorare il traffico sulla rete e avere una visione completa sull'utilizzo del filtro.
  - Deve inoltre essere presente la possibilità di impersonare uno dei clienti in gestione al fornitore di servizi, visualizzando il pannello come se tale cliente avesse effettuato l'accesso con il proprio account utente. L'MSP deve essere in grado non solo di visionare i report e le statistiche relative al cliente, ma anche di monitorare ed eventualmente modificare la sua configurazione del filtro.

  - La schermata deve adattarsi dinamicamente in base al tipo di utente (MSP o cliente finale), mostrando solo i componenti grafici e le funzionalità a cui l'utente ha accesso.

- **Gestione dei template di protezione:**
  - Il sistema deve consentire la creazione e la gestione di template, che possono essere assegnati a diversi clienti.
  - I template devono supportare opzioni di configurazione come categorie e applicazioni da bloccare, paesi da bloccare, attivazione SafeSearch e blocco completo tranne alcuni domini specifici.
  - Le modifiche apportate ai template devono essere propagate automaticamente a tutti i clienti a cui sono stati assegnati.
  - Devono essere disponibili template di eccezioni personalizzati, configurabili per soddisfare esigenze specifiche.
  - Le liste di eccezioni devono poter essere importate nel sistema anche tramite file di testo in formato `.txt` o `.csv`.

- **Reportistica:**
  - Il sistema deve permettere di creare, visualizzare ed esportare report personalizzati.
  - I report devono poter coprire intervalli temporali più estesi rispetto al legacy, passando da un mese a 3, 6 o anche 12 mesi per singolo report.
  - Deve essere possibile pianificare l'invio automatico dei report via email, con frequenza settimanale o mensile.

- **API pubbliche:**
  - Il sistema deve offrire un set di API pubbliche per la gestione delle configurazioni, consentendo agli utenti di integrare il filtro DNS in applicazioni personalizzate o di sviluppare un proprio client per la gestione delle configurazioni.
  - Le API devono supportare tutte le funzionalità disponibili tramite il pannello Web, inclusa la nuova gestione dei template.
  - Le API devono essere provviste di una documentazione chiara e completa, che descriva i metodi disponibili e i relativi parametri.
- **Notifiche:**
  - Il sistema deve fornire notifiche direttamente nel pannello Web e via email.
  - Le notifiche devono riguardare nuove funzionalità rilasciate, errori di rete e malware rilevati.
  - Le notifiche via email devono includere report pianificati dagli utenti.
- **Personalizzazione Whitelabel:**
  - Il sistema deve supportare la personalizzazione whitelabel riservata ai Dealer, consentendo di modificare elementi come: nome, logo, palette di colori, menu, messaggi personalizzati al login, email per il supporto, notifiche e redirect.
- **Audit log:**
  - Il sistema deve registrare in un log dettagliato tutte le azioni eseguite dagli utenti, inclusi gli accessi al pannello e le modifiche alla configurazione del filtro.
- **Supporto per il multilingua:**
  - Il sistema deve supportare la localizzazione in diverse lingue, e in particolare italiano, inglese, spagnolo, francese e portoghese.

  - Esso deve inoltre essere progettato per consentire l'aggiunta di nuove lingue in maniera rapida e agevole.
  - La lingua di default deve essere impostata in base alla preferenza del browser dell'utente.
  - Le traduzioni devono coinvolgere tutti gli elementi dell'interfaccia grafica, inclusi i messaggi di errore e le notifiche.
  - Per quanto riguarda le API, l'unica lingua supportata sarà l'inglese.

Requisiti non funzionali

- **Modellazione del dominio e gestione del database:**
  - Il sistema deve prevedere una nuova modellazione del dominio, al fine di migliorare la struttura dei dati e rendere il database più robusto ed efficace.
  - Il database deve essere progettato in modo da garantire una gestione più coerente delle relazioni tra entità, evitando le problematiche riscontrate nel sistema legacy.
- **Sicurezza:**
  - Il sistema deve implementare l'autenticazione a più fattori (*Multi-Factor Authentication*, MFA) per tutti gli utenti.
  - La sicurezza deve essere garantita tramite un sistema di autorizzazione centralizzato, che gestisca permessi e operazioni in base ai ruoli degli utenti e alle licenze possedute. Questo servizio dedicato deve applicare le autorizzazioni a tutti i livelli, includendo le API, l'interfaccia del pannello e la gestione delle policy.
- **Usabilità e interfaccia utente:**
  - L'interfaccia deve essere moderna, accattivante e progettata per garantire una user experience migliorata.
  - Il design deve avere un aspetto estetico professionale e orientato a clienti di grandi organizzazioni (look enterprise).

  - L'interfaccia deve essere responsive e ottimizzata per l'utilizzo su qualsiasi dispositivo, sia esso desktop o mobile.

- **Manutenibilità:**

  - Il sistema deve essere sviluppato in maniera modulare, per semplificare la manutenzione e consentire l'estensione con nuove funzionalità.
  - Ci deve essere una netta distinzione tra frontend e backend, così come la suddivisione del backend in molteplici microservizi.

Le scelte progettuali adottate per l'implementazione delle funzionalità sviluppate nel contesto di questa tesi verranno discusse nel capitolo successivo.

3.3.3 Modellazione del dominio

Per progettare correttamente il nuovo sistema è stato necessario analizzare il dominio applicativo, partendo dalla struttura concettuale del sistema legacy, per definire in modo più chiaro le entità fondamentali e le loro relazioni. Questa analisi fornisce una base strutturata per la modellazione dei dati e per l'implementazione delle funzionalità previste nel nuovo pannello di configurazione.

Attualmente, la modellazione è ancora in fase di sviluppo e la seguente descrizione rappresenta una versione parziale del sistema. In particolare, le entità **Protection**, **Network**, **License** e **Report** non sono state ancora completamente definite insieme al team aziendale. Tuttavia, si è scelto di includerle ugualmente per fornire un quadro più completo del dominio e delineare le aree che necessiteranno di ulteriori approfondimenti. Al contrario, le entità **Organization** e **User** sono state completamente modellate e costituiscono il nucleo del sistema multi-tenant.

In Figura 3.2 è riportato il diagramma delle classi in formato UML che rappresenta il dominio applicativo del nuovo sistema. Tale rappresentazione non ha la pretesa di essere esaustiva, soprattutto dal punto di vista degli attributi e dei metodi delle classi, ma mira a fornire una visione d'insieme delle entità coinvolte e delle relazioni tra di esse.

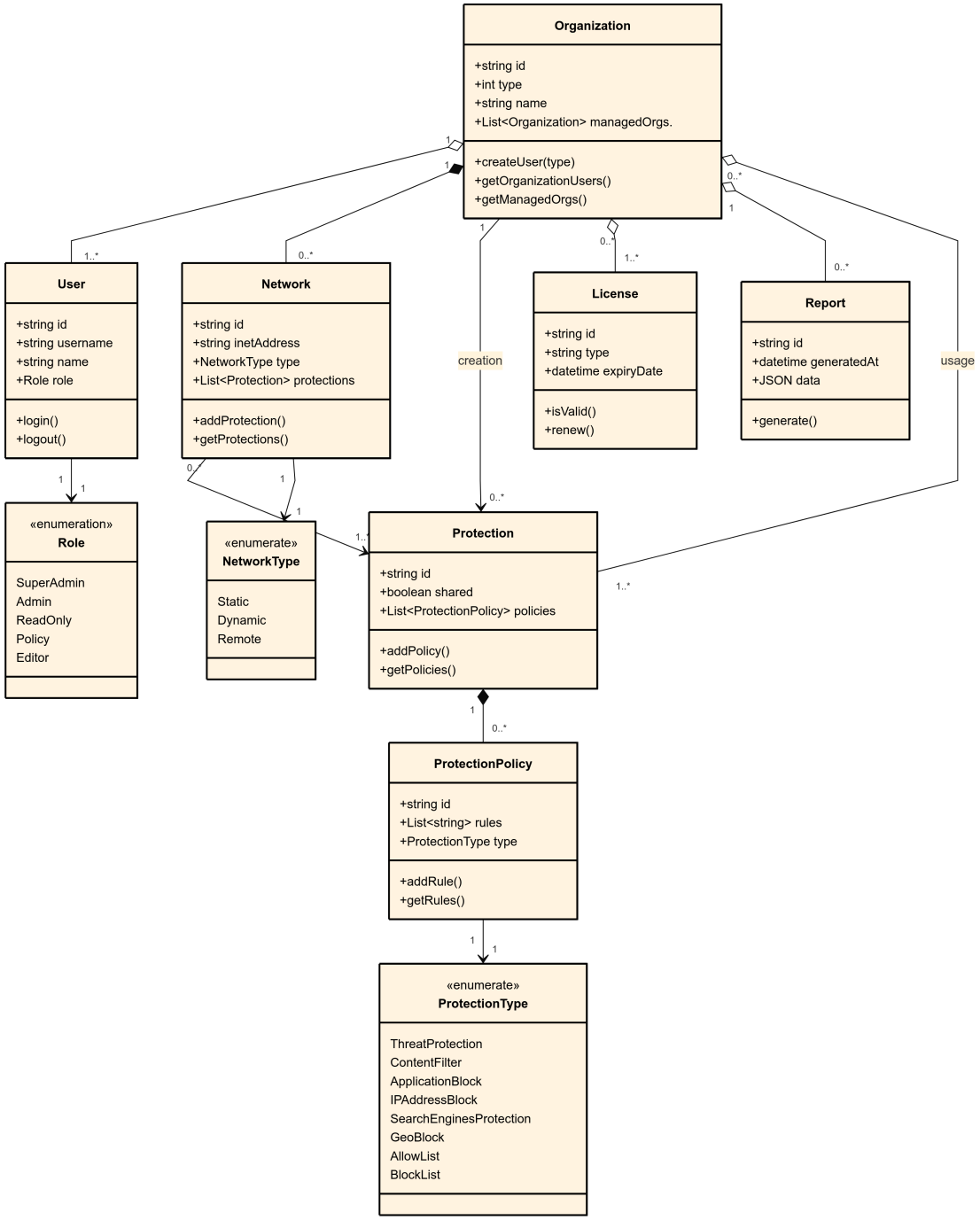

Figura 3.2: Diagramma delle classi UML relativo al dominio applicativo del nuovo sistema.

Organization

L'entità `Organization` rappresenta il fulcro del sistema multi-tenant ed è il punto di riferimento per tutte le altre componenti. Ogni utente, configurazione di filtraggio e reportistica è direttamente associata a un'organizzazione, che costituisce il contesto primario in cui avvengono le operazioni all'interno del sistema.

L'organizzazione è l'elemento che definisce l'esistenza stessa di un utente all'interno del sistema. Questi ultimi, infatti, non esistono al di fuori del contesto organizzativo: ogni utente deve appartenere a una specifica organizzazione e non può operare al di fuori di essa. Al momento della creazione di una nuova organizzazione, viene automaticamente generato un primo utente con permessi di amministratore. Tale amministratore ha la facoltà di creare nuovi utenti all'interno della propria organizzazione, o in quelle da essa gestite.

L'entità in questione non si limita a definire il perimetro degli utenti, ma è il nodo centrale attorno al quale ruotano tutte le altre configurazioni del sistema. Ogni organizzazione è associata a una o più reti (Network), che rappresentano gli indirizzi IP da sottoporre al filtraggio DNS. Le configurazioni di protezione (Protection), che definiscono cosa filtrare, sono legate all'organizzazione e non ai singoli utenti. Questo significa che tutte le policy di filtraggio sono decise a livello organizzativo. I dati statistici relativi all'utilizzo del sistema di filtraggio DNS sono raccolti e visualizzati a livello di organizzazione, consentendo agli utenti di monitorare il traffico filtrato e le attività correlate.

Ogni organizzazione è identificata da un `id` univoco, un nome (`name`) e una tipologia (`type`) che ne determina il ruolo all'interno del sistema. Esistono due macrotipologie di organizzazione:

- **Organizzazioni che gestiscono altre organizzazioni**, ovvero Managed Service Provider e Dealer.

  - **MSP**: oltre a rivendere il servizio di filtraggio DNS, si occupano direttamente della gestione della protezione per i loro clienti. In questo modello, il cliente finale non ha autonomia sulle proprie policy di protezione, in quanto è l'MSP a definirle e amministrarle.

  - **Dealer**: rivendono il servizio di filtraggio DNS, ma senza occuparsi della configurazione delle policy. I clienti finali hanno piena libertà di gestione delle proprie protezioni.

- **Clienti finali**, che possono essere:

  - **Altre organizzazioni**, come scuole, hotel, aziende, attività commerciali.
  - **Singoli individui**, che desiderano proteggere la propria rete domestica e i propri dispositivi personali.

User

L'entità `User` rappresenta qualsiasi soggetto in grado di accedere al pannello di configurazione di un'organizzazione ed eseguire azioni sulla base dei permessi definiti dal proprio ruolo. Si ribadisce il fatto che gli utenti operano sempre all'interno del contesto di un'organizzazione, non potendo esistere in modo indipendente. Il sistema adotta un modello a ruoli per differenziare le capacità operative di ciascun utente, limitandone o ampliandone i privilegi a seconda delle necessità organizzative.

Esistono cinque tipologie di utenti, ciascuna con permessi specifici nell'ambito della propria organizzazione:

- **SuperAdmin**: rappresenta l'utente con i più alti privilegi possibili. Questo ruolo è riservato al team di supporto dell'azienda produttrice del filtro DNS e gode di accesso illimitato a tutte le organizzazioni registrate nel sistema. Un SuperAdmin può creare ed eliminare utenti e organizzazioni, modificare qualsiasi policy e visionare tutti i report disponibili, indipendentemente dal cliente a cui sono associati.

- **Admin**: è l'utente amministratore della propria organizzazione e di tutte quelle al di sotto di essa nella gerarchia. Ha pieno controllo sulle entità amministrate ed è l'unico utente con il permesso di creare nuovi account all'interno del proprio contesto organizzativo.

- **Policy**: ha la facoltà di gestire le reti (Network) e i profili di protezione (Protection) all'interno della propria organizzazione. Non dispone di permessi amministrativi e non può gestire gli utenti.

- **Editor**: ha privilegi simili all'utente di tipo Policy, ma con un ambito più ristretto. In particolare, può gestire solo le configurazioni di sicurezza, senza poter modificare le impostazioni delle reti.

- **ReadOnly**: è l'utente con il livello di accesso più limitato. Ha esclusivamente il permesso di visionare report e statistiche, senza poter modificare alcuna configurazione. Il suo ruolo è strettamente legato al monitoraggio passivo delle attività del filtro DNS per una data organizzazione.

Ogni utente è identificato da un `id`, un nome (`name`), un ruolo (`role`) e da uno `username`. Quest'ultimo campo contiene l'indirizzo email dell'utente ed è utilizzato per l'autenticazione all'interno del sistema.

License

L'entità `License` rappresenta una delle componenti ancora in fase di modellazione, ma il suo ruolo nel sistema è chiaro: essa determina il livello di servizio del prodotto associato a un'organizzazione e influisce sui permessi e sulle funzionalità disponibili per quest'ultima. Ogni organizzazione deve disporre di almeno una licenza attiva per poter usufruire del servizio di filtraggio DNS.

Esistono diverse tipologie di licenze, tra cui quelle di prova, che vengono fornite gratuitamente con lo scopo di illustrare e dimostrare le capacità del prodotto prima di un eventuale acquisto. Oltre alle licenze di prova, sono previste licenze a pagamento con diversi livelli di servizio, i cui dettagli non sono ancora stati completamente definiti.

L'assegnazione di una licenza può essere effettuata esclusivamente da un utente con il ruolo di SuperAdmin, oppure dall'Admin di un'organizzazione di tipo MSP o Dealer. Ciò significa che solo il produttore del software e gli intermediari autorizzati possono concedere o modificare una licenza per un'organizzazione cliente.

Report

L'entità `Report` rappresenta la capacità del sistema di generare e visualizzare dati analitici relativi all'attività di filtraggio DNS per una o più organizzazioni. Lo scopo principale di un report è quello di fornire un monitoraggio dettagliato sull'utilizzo del filtro, offrendo statistiche utili per comprendere il traffico di rete e l'efficacia delle policy di protezione adottate. Tra le informazioni contenute in un report vi sono, ad esempio, il numero di richieste DNS bloccate, le cinque categorie di siti più frequentemente filtrate, il numero di domini non risolti e altri indicatori rilevanti per la sicurezza della rete.

Una caratteristica distintiva del nuovo sistema è la possibilità di generare report aggregati, che includono dati provenienti da più organizzazioni. Questa funzionalità è particolarmente utile per gli MSP, in quanto consente loro di ottenere una panoramica completa e centralizzata sui clienti che gestiscono. I report aggregati possono includere informazioni come l'elenco dei malware rilevati e i dispositivi su cui sono stati identificati, il numero di sessioni attive e altre metriche relative alla sicurezza e alla gestione del traffico DNS.

L'entità Report non è ancora stata completamente modellata, ma il suo ruolo nel sistema è fondamentale per garantire una visibilità chiara e approfondita sull'efficacia delle configurazioni di filtraggio applicate alle reti delle organizzazioni.

Protection

L'entità `Protection` rappresenta il concetto di protezione associato a una rete (Network), determinando in che modo il sistema di filtraggio DNS deve bloccare o consentire l'accesso ai contenuti. Ogni istanza di Protection è composta da una lista di policy di sicurezza, ognuna delle quali specifica una particolare configurazione del filtro. Generalmente, una Protection viene creata da un'organizzazione e assegnata a una o più delle proprie reti, definendo le regole di filtraggio applicate al traffico proveniente da tali indirizzi IP.

Con lo sviluppo del nuovo sistema, l'entità in esame introduce un'importante evoluzione: la possibilità di essere condivisa tra più organizzazioni. Se un'Organization crea una Protection e la contrassegna come condivisibile, tale configurazione potrà essere visualizzata e assegnata dai clienti gestiti dall'organizzazione creatri-

ce. Questa funzionalità risulta particolarmente utile per gli MSP e i Dealer, che possono fornire ai propri clienti configurazioni predefinite e standardizzate senza richiedere loro di crearne di nuove.

Un ulteriore vantaggio del nuovo modello è la propagazione automatica delle modifiche per le Protection condivise. Se l'organizzazione creatrice aggiorna una configurazione di protezione condivisa, tutte le organizzazioni che la utilizzano vedranno applicate automaticamente le modifiche, senza necessità di intervento manuale. Per supportare questa logica, la modellazione prevede due relazioni distinte tra Protection e Organization:

1. Una relazione di *associazione semplice* che esprime la creazione, collegando un'Organization alla Protection che ha generato.

2. Una relazione di *aggregazione* che esprime l'utilizzo, collegando una Protection a una o più Organization che la utilizzano. Questa relazione è valida sia per le Protection non condivise, che per quelle condivise.

Nonostante la sua importanza, Protection da sola non è sufficiente a garantire il livello di granularità richiesto da un moderno sistema di filtraggio DNS. Per questo motivo, essa è composta da una lista di configurazioni più specifiche, rappresentate dall'entità ProtectionPolicy, che definisce nel dettaglio il comportamento della protezione.

ProtectionPolicy

L'entità `ProtectionPolicy` rappresenta un insieme di regole appartenenti alla stessa tipologia di protezione, definita dall'attributo `ProtectionType`. Poiché il sistema di filtraggio DNS prevede diverse categorie di protezione, la modellazione adotta una struttura a due livelli: l'entità Protection mantiene la lista delle policy applicate, mentre ogni elemento di questa lista contiene le regole specifiche relative a una determinata categoria di protezione. Questa suddivisione consente di organizzare in modo chiaro e modulare le configurazioni di sicurezza. Un'istanza di Protection può essere composta da più ProtectionPolicy, ciascuna responsabile di un particolare aspetto del filtraggio. Questo approccio permette alle organizzazioni di combinare diverse policy all'interno di un'unica configurazione di protezione,

garantendo così una gestione flessibile e scalabile delle regole di filtraggio DNS. I possibili tipi di protezione (`ProtectionType`) includono:

- **ThreatProtection**: blocca siti identificati come minacce alla sicurezza, come malware e phishing.
- **ContentFilter**: filtra i contenuti sulla base della loro categoria tematica (es. pornografia, giochi d'azzardo, social network).
- **ApplicationBlock**: impedisce l'accesso a specifiche applicazioni o servizi online.
- **IPAddressBlock**: blocca l'accesso a determinati indirizzi IP.
- **SearchEnginesProtection**: applica restrizioni alle ricerche sui motori di ricerca, come l'attivazione forzata della modalità SafeSearch.
- **GeoBlock**: limita l'accesso a domini o indirizzi IP in base alla loro geolocalizzazione.
- **AllowList** e **BlockList**: definiscono eccezioni personalizzate, rispettivamente per consentire o bloccare domini specifici.

Network

L'entità `Network` riveste un ruolo fondamentale nel presente dominio applicativo, in quanto definisce l'ambito su cui devono essere applicate le regole di protezione del filtro DNS. Ogni rete è associata a un'Organization e rappresenta il punto di ingresso del traffico che verrà sottoposto a filtraggio. La configurazione della protezione di una rete è determinata dalla Protection assegnata ad essa, specificando così quali policy devono essere applicate. Questa relazione tra Network e Organization è modellata come una *composizione*, indicando che una rete non può esistere senza l'Organization a cui appartiene. Se un'organizzazione viene eliminata, anche tutte le reti ad essa associate vengono rimosse.

Una rete può appartenere a una delle seguenti tipologie, identificate dall'enumerazione `NetworkType`:

- **Static**: identifica reti con un indirizzo IP statico, il cui riferimento rimane invariato nel tempo.
- **Dynamic**: rappresenta reti con indirizzo IP dinamico, che necessitano dell'ausilio di un servizio di DynamicDNS per garantire un'associazione stabile nel tempo.
- **Remote**: utilizzata per proteggere dispositivi mobili ed endpoint aziendali che richiedono la protezione del filtro DNS anche quando non sono connessi alla rete principale dell'organizzazione.

Ogni Network è identificata da un `id`, da un `type` e da un `inetAddress`. Quest'ultimo supporta sia indirizzi IPv4 che IPv6 e consente l'inserimento di intervalli IP secondo lo standard Classless Inter-Domain Routing[7] (CIDR), specificandoli con una barra finale (es. `192.168.1.0/24`). L'associazione tra Network e Protection permette di applicare un insieme di policy a una specifica rete, garantendo così un controllo granulare sul filtraggio del traffico DNS.

[7] `https://datatracker.ietf.org/doc/html/rfc4632`

Capitolo 4

Design

Il presente capitolo descrive le scelte architetturali, tecnologiche e progettuali adottate per soddisfare gli obiettivi e i requisiti definiti nel capitolo precedente. L'attenzione si concentra sulle funzionalità sviluppate o analizzate durante il tirocinio, pur accennando alle linee guida per alcune delle funzionalità future, al fine di offrire una visione più completa del sistema.

La progettazione del nuovo sistema punta a superare i limiti identificati nel legacy, adottando un'architettura moderna, modulare e scalabile, che semplifichi la gestione e consenta l'integrazione di nuove funzionalità. Particolare attenzione è stata dedicata alla separazione delle responsabilità tra i vari componenti, alla progettazione di un backend flessibile e robusto agli errori, e alla realizzazione di un frontend intuitivo e con un aspetto enterprise.

Il capitolo è strutturato come segue: dopo una descrizione dell'architettura generale del sistema, si approfondiranno le principali componenti e funzionalità chiave, come la multiutenza e la gestione degli errori. Infine, saranno discussi gli aspetti relativi al miglioramento del database e alle strategie messe in campo per garantire una corretta migrazione dei dati dal sistema legacy.

4.1 Architettura generale

Per la progettazione del nuovo sistema è stata adottata un'architettura a microservizi, caratterizzata da una separazione netta tra frontend e backend. Questa

scelta è motivata dalla necessità di superare i limiti architetturali del sistema legacy, introducendo modularità e una chiara separazione delle responsabilità tra i vari componenti.

Nel nuovo sistema, il frontend è rappresentato da un singolo microservizio dedicato alla logica di presentazione dei dati e all'interazione con l'utente. Il backend, invece, è stato suddiviso in molteplici microservizi indipendenti, ciascuno dedicato a un modulo o a una funzionalità specifica dell'applicativo. Questa scomposizione consente di ottenere una maggiore flessibilità, facilitando sia l'aggiunta di nuove funzionalità che la manutenzione del sistema.

I microservizi comunicano tra loro utilizzando il protocollo HTTP e, più specificamente, attraverso le API RESTful che ciascuno di essi espone. Queste ultime vengono inoltre utilizzate sia dal frontend sia per la creazione delle API pubbliche aziendali. Tali API, già presenti insieme al sistema legacy, sono state rinnovate per adattarsi alle nuove logiche di funzionamento introdotte nel nuovo sistema.

Una rappresentazione schematica dell'architettura del sistema è riportata in Figura 4.1, dove sono evidenziate le principali componenti del frontend e del backend. Il diagramma mostra come il primo gestisca l'interazione con l'utente attraverso moduli specifici per il routing e per la gestione dello stato e della sessione, mentre il secondo sia organizzato in servizi dedicati a funzionalità come gestione utenti e autenticazione.

4.1.1 Struttura del Frontend

Il frontend è stato progettato come una Single-Page Application (SPA) [FF14], scelta che consente di offrire agli utenti un'esperienza fluida e un accesso rapido da qualsiasi browser, senza bisogno di installare alcun applicativo localmente. Questo microservizio si occupa esclusivamente della logica di presentazione, implementando l'interfaccia grafica e gestendo l'interazione con gli utenti finali. Infatti, tutte le operazioni di elaborazione dei dati, come le computazioni o la gestione delle regole di business, sono demandate al backend.

L'architettura del frontend segue un approccio modulare, in cui ogni pagina e componente è organizzato secondo una struttura gerarchica. Ciò consente di

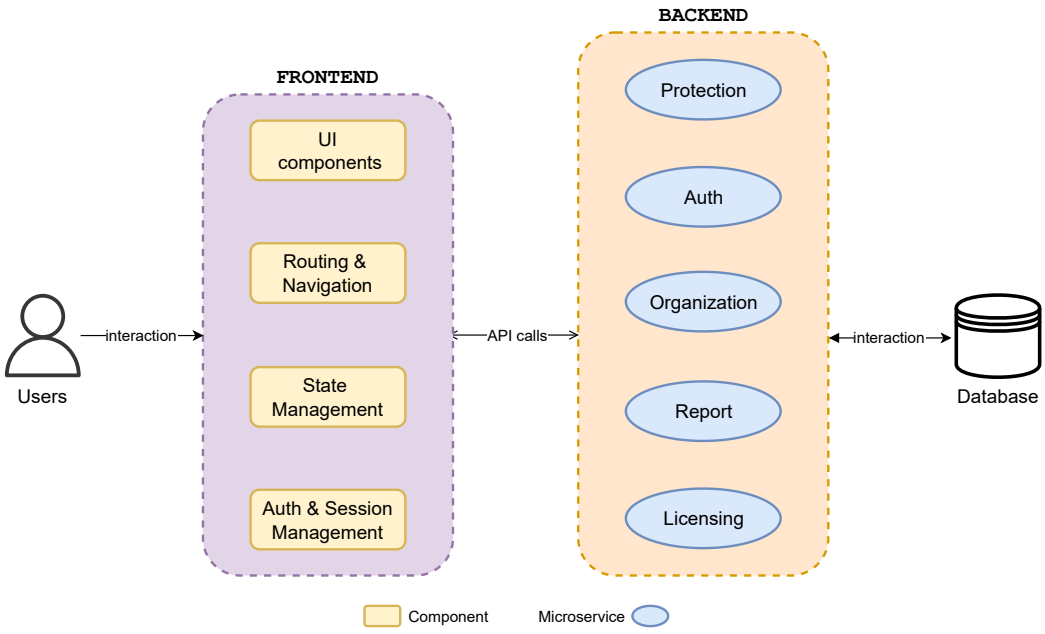

Figura 4.1: Architettura generale del sistema.

mantenere l'architettura chiara e scalabile, facilitando l'estensione del sistema e la gestione delle singole sezioni dell'applicazione.

Per garantire un design coerente e accelerare lo sviluppo dell'interfaccia, è stato adottato un template grafico altamente personalizzabile. Questo approccio ha consentito di ridurre i tempi di progettazione dell'interfaccia grafica, mantenendo al contempo la flessibilità necessaria per adattare l'aspetto grafico alle esigenze specifiche del progetto.

4.1.2 Struttura del Backend

Come già accennato, il backend non è stato concepito come un unico servizio monolitico, ma si è optato per una suddivisione in più microservizi, ciascuno responsabile di una specifica funzionalità o modulo del sistema. La definizione di questi microservizi è stata guidata dall'analisi del dominio (Sezione 3.3.3), individuando le principali entità e le loro responsabilità all'interno dell'architettura complessiva. Questo approccio ha permesso di modellare il backend in modo coerente con

le esigenze del sistema, garantendo una chiara separazione delle responsabilità e facilitando l'evoluzione futura della piattaforma. Di seguito, vengono descritti i principali microservizi di cui il backend è composto.

Auth Il microservizio `auth` è responsabile dell'autenticazione e dell'autorizzazione degli utenti. L'autenticazione è implementata attraverso un meccanismo basato su token, che consente di gestire sessioni sicure senza la necessità di mantenere uno stato lato server. L'autorizzazione, invece, fornisce un sistema per determinare se un utente ha i permessi necessari per eseguire una determinata azione. Questa valutazione si basa sul ruolo dell'utente e sulla licenza associata all'organizzazione cui appartiene, garantendo così un controllo granulare sugli accessi alle funzionalità della piattaforma.

Report Il microservizio `report` gestisce la generazione e la visualizzazione dei report relativi alle attività di filtraggio DNS. Nella prima versione del sistema, questa funzionalità è limitata alla visualizzazione di report specifici per gli MSP. Questo microservizio si interfaccia con il database per raccogliere e aggregare i dati necessari, fornendo agli utenti una panoramica dettagliata sull'attività di filtraggio e sull'efficacia delle policy applicate.

Organization Il microservizio `organization` è un componente chiave per l'implementazione della multiutenza. Esso si occupa di gestire tutte le operazioni CRUD sugli utenti, consentendo di creare nuovi account, modificarne i dati, nonché rimuovere utenti dalle organizzazioni. Grazie a questo servizio più utenti possono essere associati a un'unica organizzazione con livelli di accesso differenziati, migliorando la flessibilità e la gestione delle autorizzazioni.

Protection Il microservizio `protection` è responsabile della gestione delle policy di protezione applicate agli utenti e alle organizzazioni. Questo servizio permette di creare, modificare ed eliminare i profili di protezione, ossia insiemi di regole che determinano quali contenuti possono essere filtrati o consentiti. Inoltre, gestisce la creazione e l'amministrazione dei profili condivisi, che consentono di applicare

una configurazione comune a più clienti senza dover definire manualmente le stesse regole per ciascuno di essi.

Licensing Il microservizio `licensing` si occupa della gestione delle licenze associate agli utenti e alle organizzazioni. Attraverso questo servizio, è possibile visualizzare lo stato delle licenze attive, gestire le assegnazioni e monitorare la loro scadenza. Questo microservizio è essenziale per garantire che ogni utente abbia accesso solo alle funzionalità previste dal proprio piano, permettendo un controllo efficace sui livelli di servizio offerti.

Architettura a livelli

I microservizi appena discussi adottano un'architettura uniforme, basata sull'esposizione di API REST, garantendo una progettazione modulare e scalabile. Tale struttura consente di mantenere indipendenti le diverse componenti del sistema, facilitando la manutenzione e l'evoluzione del software. Ogni microservizio è organizzato in più livelli, ciascuno con responsabilità ben definite, come illustrato in Figura 4.2:

- **Routes**: rappresenta il punto di ingresso delle richieste HTTP. Questo livello si occupa di applicare eventuali middleware, come quello per la gestione dell'autenticazione, e di indirizzare le richieste verso il controller appropriato, in base al metodo HTTP e al percorso richiesto.
- **Controllers**: ricevono le richieste dai Routes, validano i parametri in ingresso e delegano l'elaborazione ai servizi applicativi. Una volta ottenuto il risultato, i controllers generano la risposta HTTP da restituire al client.
- **Services**: costituiscono il livello di business logic, elaborando i dati e orchestrando le operazioni necessarie. Questo livello intermedio permette di mantenere separata la logica di dominio dall'accesso ai dati, garantendo maggiore modularità e riusabilità del codice.
- **Repositories**: forniscono un'interfaccia per l'accesso ai dati, eseguendo operazioni di lettura e scrittura sul database. Questo livello è responsabile esclusivamente della gestione dei dati.

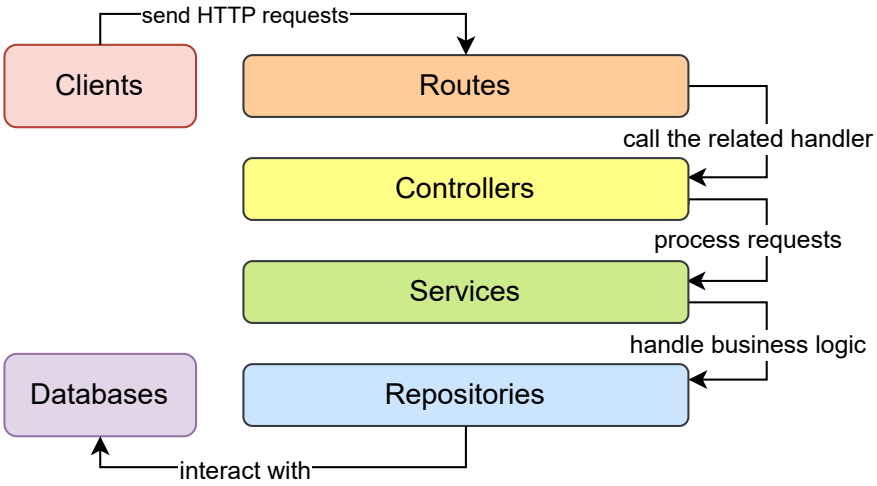

Figura 4.2: Visione schematica relativa all'architettura a livelli dei microservizi.

- *Database*: sebbene non sia parte integrante dell'architettura dei microservizi, rappresenta il livello di persistenza su cui vengono eseguite le operazioni di lettura e scrittura da parte dei repositories.

Questa suddivisione in livelli garantisce una chiara separazione delle responsabilità, migliorando la manutenibilità e la scalabilità del sistema. Inoltre, consente di evolvere il backend in modo strutturato, rendendo più semplice l'aggiunta di nuove funzionalità o modifiche alle logiche applicative senza impattare gli altri componenti.

4.1.3 Interazione tra i componenti

La comunicazione tra i componenti del sistema avviene principalmente tramite chiamate HTTP verso un'API, permettendo al frontend, e ai microservizi del backend, di interagire in maniera modulare ed efficiente. In Figura 4.3 è rappresentata la struttura generale del sistema, evidenziando i flussi di comunicazione tra frontend, backend e database, oltre alle interazioni tra i microservizi del backend.

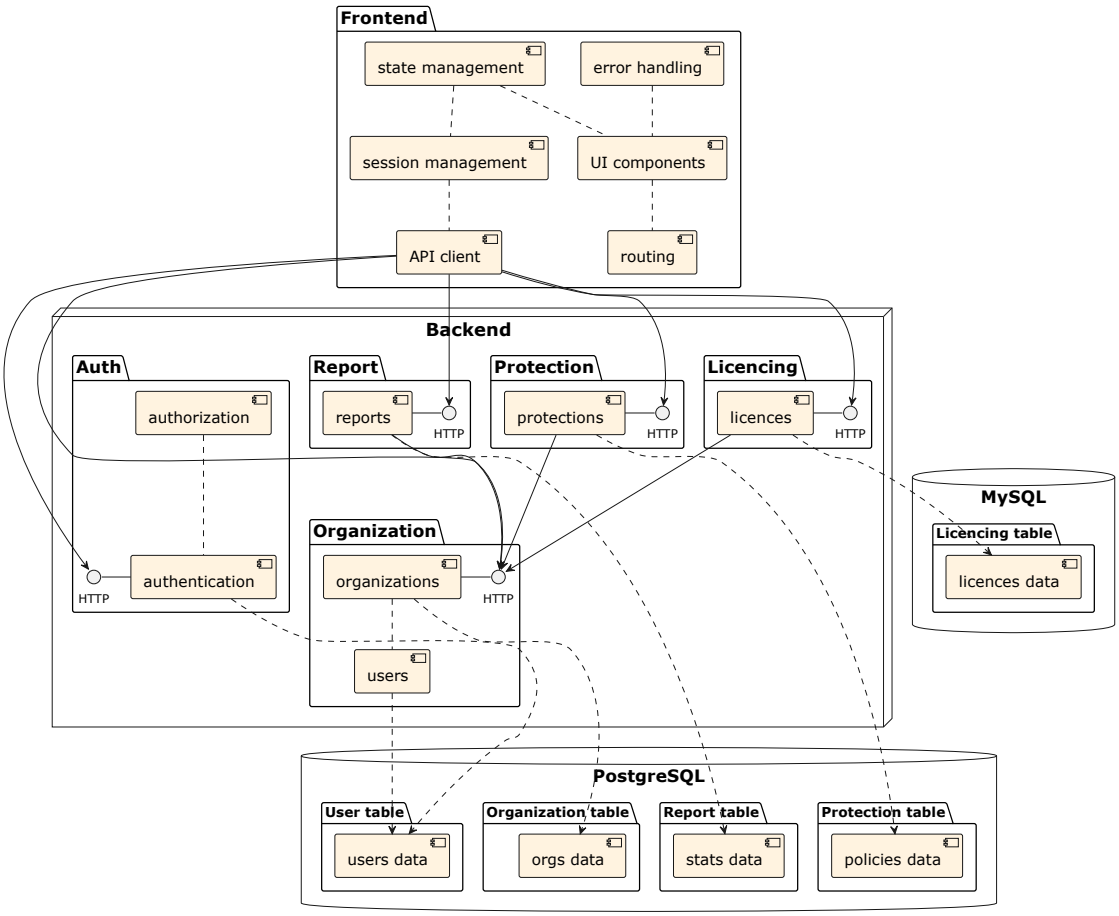

Figura 4.3: Visione globale del sistema relativamente ai suoi componenti e alle interazioni tra di essi.

Componenti e interazioni del frontend

Sebbene il frontend sia implementato come un unico microservizio, al suo interno è suddiviso in moduli funzionali che si occupano di gestire aspetti specifici dell'interfaccia utente e della comunicazione con il backend. Tra i principali:

- **Routing e navigazione**: gestisce il passaggio tra le diverse viste dell'applicazione.
- **Gestione dello stato**: mantiene e sincronizza le informazioni tra i componenti per garantire coerenza nei dati.
- **Autenticazione e sessione**: protegge le sezioni riservate e gestisce l'accesso degli utenti.
- **Client API**: funge da intermediario tra il frontend e il backend, inviando richieste e processando le risposte.
- **Gestione degli errori**: intercetta eventuali problemi nelle comunicazioni con il backend e fornisce all'utente un feedback chiaro.

Interazioni del backend

Come è già stato puntualizzato, i microservizi del backend comunicano tra loro, e con il frontend, esponendo API REST. Ogni servizio è progettato per gestire una specifica funzionalità e può interagire con gli altri quando necessario. Le principali interazioni avvengono secondo due modalità:

- **Comunicazione frontend-backend**: il frontend invia richieste ai microservizi per ottenere dati o eseguire operazioni, ricevendo le informazioni necessarie per aggiornare l'interfaccia utente.
- **Interazioni tra microservizi**: i servizi si scambiano dati tramite API REST per operazioni che coinvolgono più moduli, come la gestione degli utenti, delle organizzazioni o delle policy di protezione.

Questo modello garantisce che ogni microservizio operi in modo indipendente, riducendo le dipendenze dirette e migliorando la scalabilità e l'efficienza complessiva del sistema.

Interazione con i database

Il backend sfrutta due database distinti per la gestione della persistenza dei dati:

- **PostgreSQL**: utilizzato per la maggior parte delle operazioni, inclusa la gestione di utenti, organizzazioni, report, autenticazione e policy di protezione.
- **MySQL**: impiegato esclusivamente per la gestione delle licenze, in quanto derivante dal sistema legacy. Il pannello customer area utilizza questo database per archiviare e gestire le licenze dei clienti.

Interazione tra i livelli interni del backend

Come già menzionato, ogni microservizio del backend segue un'architettura a livelli ben definita, composta da *Routes*, *Controllers*, *Services* e *Repositories*. Tuttavia, non è stato ancora approfondito il modo in cui questi livelli interagiscono tra loro e come questa struttura influenzi il funzionamento complessivo del sistema.

L'organizzazione a livelli permette di mantenere una chiara separazione delle responsabilità, semplificando la manutenzione e l'estensibilità del software. La Figura 4.4 illustra il flusso di interazione tra questi componenti, evidenziando il percorso di una richiesta all'interno del sistema, dal momento in cui viene ricevuta fino alla gestione dei dati nel livello di persistenza.

Il processo inizia quando un client invia una richiesta HTTP al backend, includendo eventualmente un token di autenticazione. Il primo livello che intercetta la richiesta è quello delle *Routes*, che ha il compito di validare il token e determinare il controller appropriato in base al metodo HTTP e al percorso della richiesta. Se il token di autenticazione non è valido, il sistema risponde immediatamente con un errore. Al contrario, se il processo di autenticazione va a buon fine, la richiesta viene inoltrata al livello dei *Controllers*, che ha la responsabilità di verificare la correttezza dei parametri in ingresso. Se i parametri risultano non validi, viene generata una risposta di errore senza procedere oltre. In caso contrario, il controller delega l'elaborazione della richiesta al livello *Services*. I servizi rappresentano il cuore della logica di business del sistema. Qui vengono eseguite le operazioni necessarie per soddisfare la richiesta, che possono includere accessi ai dati. Se è necessario recuperare o modificare informazioni persistenti, il servizio delega l'o-

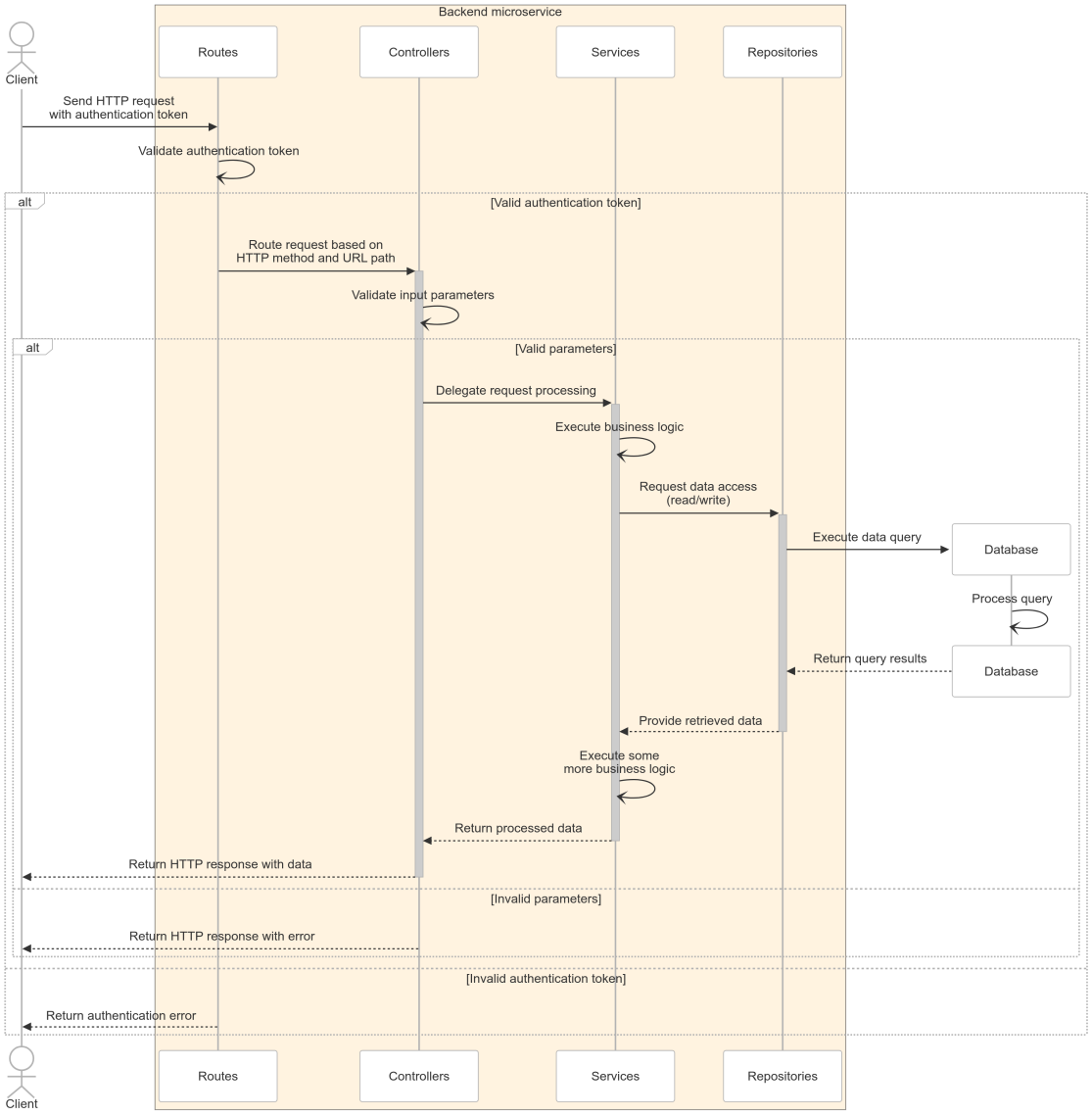

Figura 4.4: Diagramma di sequenza UML che illustra l'interazione tra i livelli dell'architettura interna del backend.

perazione al livello dei *Repositories*, il quale è responsabile della comunicazione diretta con il database. Esso esegue le query richieste e restituisce i risultati al servizio chiamante. Quest'ultimo può applicare ulteriori elaborazioni ai dati ricevuti prima di restituirli al controller. Infine, il controller invia la risposta direttamente al client, restituendo i dati elaborati o, in caso di errore, un messaggio appropriato. Questo flusso strutturato garantisce un'elevata modularità del backend, facilitando l'estensione del sistema e migliorando la gestione degli errori e della sicurezza.

4.1.4 Scelte tecnologiche

Le tecnologie utilizzate per l'implementazione del nuovo sistema erano già state definite prima dell'inizio del presente lavoro di tesi. Tuttavia, la loro adozione può essere motivata sulla base di considerazioni tecniche e di opportunità.

Per l'intero sistema, sia frontend che backend, è stato scelto il linguaggio TypeScript[1], che combina la flessibilità e la vasta disponibilità di librerie di JavaScript con i vantaggi di un linguaggio tipizzato. Grazie a TypeScript, è stato possibile sviluppare entità e funzionalità più robuste rispetto a un'implementazione in JavaScript puro, riducendo il rischio di errori a runtime. Il sistema di tipizzazione consente di individuare molteplici errori già in fase di compilazione, generando file JavaScript più affidabili e sicuri.

Come sistema di gestione delle dipendenze, si è optato per `pnpm`[2], un package manager che offre un'alternativa più efficiente rispetto ai classici `npm` o `yarn`. Questo strumento si distingue per la sua capacità di ottimizzare l'uso dello spazio su disco condividendo le dipendenze tra i progetti, evitando duplicazioni inutili e riducendo significativamente i tempi di installazione. Inoltre, garantisce una gestione più affidabile delle dipendenze grazie al meccanismo di *store globale*, che assicura un'integrità maggiore rispetto agli altri gestori. La scelta di `pnpm` è stata adottata in maniera uniforme per garantire coerenza tra frontend e backend, migliorando l'efficienza del processo di sviluppo.

[1] `https://www.typescriptlang.org`
[2] `https://pnpm.io`

Tecnologie del frontend

Il frontend è stato sviluppato utilizzando Next.js[3], un framework basato su React. La scelta di React[4] è stata dettata dalla volontà di mantenere la filosofia del sistema legacy, che forniva il pannello sotto forma di web application, e dalla sua combinazione tra ampia diffusione, efficienza e curva di apprendimento bilanciata. Tuttavia, piuttosto che utilizzare React in modo standalone, si è preferito adottare un framework che ne ottimizzasse l'utilizzo. Next.js è stato selezionato per le sue funzionalità avanzate, tra cui il pieno supporto a TypeScript e la gestione automatizzata di operazioni come il bundling e la compilazione, consentendo agli sviluppatori di concentrarsi sulla logica applicativa senza preoccuparsi della configurazione dell'ambiente di sviluppo.

Come strategia di routing delle pagine, si è scelto di utilizzare App Router[5], il sistema di routing integrato in Next.js. Questo approccio permette di definire le rotte dell'applicazione in modo strutturato, organizzando il codice in base alla gerarchia delle directory. Inoltre, offre supporto per il rendering lato server, migliorando la flessibilità nella gestione e nel caricamento dei componenti.

Dal punto di vista grafico, la scelta del template TailAdmin[6] come base di partenza è stata motivata dalla sua somiglianza con il design desiderato per il nuovo sistema. La disponibilità di schermate predefinite e componenti riutilizzabili ha accelerato lo sviluppo dell'interfaccia, mentre la sua natura open source ha garantito massima flessibilità, permettendo modifiche e personalizzazioni senza vincoli.

Tecnologie del backend

Tutti i microservizi del backend utilizzano il framework web Express[7] per esporre le loro API REST. Questa scelta è stata dettata dalla leggerezza e rapidità di tale libreria, oltre alla sua capacità di gestire le rotte in modo intuitivo e modulare. Express consente di definire una gerarchia di routing chiara, facilitando la suddivi-

[3] `https://nextjs.org`
[4] `https://react.dev`
[5] `https://nextjs.org/docs/app`
[6] `https://tailadmin.com`
[7] `https://expressjs.com`

sione delle responsabilità tra i vari livelli dell'architettura. Inoltre, supporta l'integrazione di middleware personalizzati, aumentando la flessibilità nella gestione delle logiche applicative, come l'autenticazione e la validazione delle richieste.

Per i meccanismi di autenticazione, si è optato per un sistema basato su JSON Web Token[8] (JWT). Questa soluzione è ampiamente diffusa grazie alla sua semplicità di implementazione e alla possibilità di essere utilizzata in ambienti scalabili e distribuiti senza la necessità di mantenere uno stato centralizzato per le sessioni utente. A condizione che venga implementata correttamente, JWT garantisce un elevato livello di sicurezza, permettendo un controllo efficiente sugli accessi e l'integrazione con i microservizi in maniera indipendente.

Per quanto riguarda la gestione dell'accesso ai dati, si è scelto di adottare un Object-Relational Mapping (ORM) per semplificare l'interazione tra il backend e il database, evitando la necessità di scrivere manualmente query SQL all'interno del codice applicativo. In particolare, la scelta è ricaduta su Prisma[9], che si differenzia dagli Object-Relational Mapping tradizionali per il suo approccio dichiarativo. Più in dettaglio, a differenza degli ORM convenzionali, che mappano le tabelle del database su classi del linguaggio di programmazione, Prisma utilizza uno schema dichiarativo chiamato *Prisma Schema* come unica fonte di verità per la struttura del database e dei modelli dell'applicazione. Questo consente una gestione più chiara e coerente dei dati, evitando le problematiche legate all'incompatibilità tra il modello a oggetti e quello relazionale. Dal punto di vista della programmazione, Prisma fornisce una libreria lato client che permette di eseguire operazioni di lettura e scrittura sul database in modo tipizzato e sicuro, senza la necessità di gestire manualmente istanze di modelli complessi.

Scelte tecnologiche del database

Il nuovo sistema adotta un database relazionale con tecnologia PostgreSQL, scelta dettata principalmente dalla necessità di garantire la compatibilità con i dati gestiti dal sistema legacy, che utilizza lo stesso DBMS. Oltre a questo requisito, PostgreSQL è stato selezionato per le sue caratteristiche di affidabilità, scalabilità

[8] `https://datatracker.ietf.org/doc/html/rfc7519`
[9] `https://www.prisma.io`

e performance, rendendolo una soluzione solida per applicazioni che devono gestire un elevato volume di dati e richieste concorrenti.

4.2 Aspetti funzionali del sistema

Il sistema in esame implementa diverse funzionalità fondamentali per garantire un'esperienza utente efficiente e una gestione strutturata delle operazioni. Tra queste, la multiutenza consente di amministrare utenti con permessi differenziati, assicurando un controllo granulare sull'accesso alle risorse. Parallelamente, la gestione degli errori è stata progettata per migliorare l'affidabilità del sistema, garantendo un trattamento coerente e strutturato delle anomalie, sia lato frontend che backend. Le sezioni seguenti approfondiscono queste funzionalità, descrivendo il modello adottato, le soluzioni implementate e le possibili aree di miglioramento.

4.2.1 Multitenancy e gestione degli utenti

Nel nuovo sistema, la multiutenza si riferisce alla possibilità per un'organizzazione di disporre di molteplici credenziali di accesso, consentendo a diversi utenti di accedere al pannello di configurazione del filtro. Essi, hanno quindi la possibilità di amministrarne le caratteristiche o monitorare le statistiche, in base al loro ruolo. Questo rappresenta un notevole miglioramento rispetto al sistema precedente, in cui ogni cliente disponeva di un singolo account con permessi di amministratore. Tale limitazione risultava problematica, in particolar modo per le organizzazioni più strutturate che necessitavano di un accesso distribuito tra più figure con differenti livelli di autorizzazione.

Modello attuale di gestione degli utenti

Attualmente, come già spiegato nella Sezione 3.3.3, gli utenti esistono solo nel contesto dell'organizzazione di cui fanno parte. Ogni organizzazione, infatti, al momento della creazione, dispone di un account utente con ruolo di amministratore (Admin). Quest'ultimo ha la possibilità di creare altri utenti, assegnando loro un ruolo specifico e un'organizzazione di appartenenza. Tale organizzazione può essere la stessa di chi effettua la creazione, oppure una delle organizzazioni gestite (nel

caso di utenti appartenenti a MSP o Dealer). Inoltre, l'Admin può modificare le informazioni di qualsiasi utente della propria gerarchia, nonché eliminarlo.

Gestione dell'autenticazione e della sessione

L'autenticazione nel backend segue il meccanismo JWT, garantendo sicurezza e scalabilità nella gestione degli accessi. Le password degli utenti, prima di essere memorizzate nel database, vengono sottoposte a *hashing* mediante la funzione `bcrypt`[10], utilizzando un "fattore di lavoro" pari a 10, che determina il numero di iterazioni dell'algoritmo e influisce sul tempo necessario per generare l'output. Questo parametro rende più oneroso il calcolo dell'*hash*, e il valore scelto aumenta la resistenza agli attacchi di forza bruta senza compromettere le prestazioni del sistema.

Durante il login, il backend genera e restituisce due token, in accordo con JWT: l'*access token*, incluso nel corpo della risposta, e il *refresh token*, inviato come cookie HTTP-only. Il primo incorpora le informazioni essenziali per l'identificazione dell'utente, tra cui il suo ID, l'email e il ruolo, permettendo al backend di verificare le autorizzazioni senza dover interrogare il database a ogni richiesta. Il secondo, invece, consente di ottenere un nuovo *access token* senza costringere l'utente a eseguire nuovamente l'autenticazione, migliorando così l'esperienza d'uso.

Questa strategia non solo incrementa la sicurezza, evitando l'esposizione dei *refresh token* nel codice lato client, ma semplifica anche la gestione per il frontend. Infatti, grazie all'uso dei cookie HTTP-only, il browser si occupa automaticamente dell'invio del *refresh token* nelle richieste al backend, riducendo il rischio di furti di credenziali e migliorando la protezione complessiva del sistema.

Limitazioni e aspetti da migliorare

Per garantire una corretta gestione dell'autorizzazione, si è deciso di adottare un modello basato su Role-Based Access Control[11] (RBAC). Tuttavia, tale sistema non è ancora stato implementato nella sua completezza. Attualmente, sono previste solo due tipologie di utenti, Admin e ReadOnly, ma senza una distinzione

[10] `https://www.usenix.org/legacy/events/usenix99/provos/provos_html/index.html`
[11] `https://www.sciencedirect.com/science/article/abs/pii/S0167404803006096`

effettiva nei permessi tra le due categorie. Gli utenti ReadOnly, infatti, possono eseguire le stesse operazioni degli Admin. Questa configurazione rappresenta una soluzione provvisoria, in attesa dell'introduzione di un sistema RBAC più strutturato. Una volta implementato, gli utenti ReadOnly saranno limitati esclusivamente alla visualizzazione di report e statistiche, senza la possibilità di apportare modifiche.

Oltre alla gestione dei ruoli, un ulteriore livello di complessità deriva dal fatto che i permessi non dipendono esclusivamente dal ruolo assegnato all'utente, ma anche dai privilegi associati all'organizzazione di appartenenza, i quali a loro volta sono influenzati dalla tipologia di licenza in uso. Per affrontare questa complessità è prevista l'implementazione di un sistema basato su una lista di permessi derivanti da questi fattori. Tali permessi verranno poi aggregati e consolidati dal backend durante il processo di autorizzazione, restituendo al client solo il fatto che l'utente abbia o meno il permesso di eseguire una determinata azione.

Questo sistema di permessi sarà applicato sia nel frontend, per regolare l'accesso alle pagine e ai componenti dell'interfaccia utente, sia nel backend, per proteggere le API e le operazioni disponibili per ciascun utente. In questo modo, sarà possibile garantire un accesso controllato e coerente su tutti i livelli del sistema, migliorando la sicurezza e l'affidabilità del processo di autorizzazione.

4.2.2 Gestione degli errori

Uno degli aspetti chiave della progettazione del nuovo sistema riguarda la definizione di un meccanismo di gestione degli errori strutturato e centralizzato. Questo sistema, sviluppato appositamente nell'ambito del tirocinio che ha portato alla presente tesi, consente di rappresentare e trattare in modo coerente le diverse tipologie di errore, garantendo uniformità tra backend e frontend e migliorando l'esperienza utente.

L'obiettivo principale di questa soluzione è evitare una gestione dispersiva e poco strutturata degli errori, introducendo un modello tipizzato e scalabile. Il sistema consente di differenziare gli errori in base al contesto e alla loro natura, assicurando che ogni anomalia sia rappresentata con un formato chiaro e prevedibile. Questo approccio consente inoltre di evitare l'esposizione di informazioni sensibili

e di mantenere la logica di gestione degli errori separata dalle altre componenti del sistema.

Gestione degli errori nel backend

Il sistema di gestione degli errori nel backend è stato progettato attorno a una gerarchia di classi che permette di categorizzare le diverse tipologie di errore. Alla base vi è una classe astratta che definisce una struttura comune, fornendo un codice identificativo, un codice di stato HTTP e un eventuale insieme di dettagli aggiuntivi. Le classi derivate rappresentano errori specifici dell'applicazione e sono strutturate per coprire due aspetti principali: da un lato, scenari generici come accesso non autorizzato o errori interni del server; dall'altro, errori strettamente legati alle entità del dominio applicativo. In quest'ultimo caso, ogni entità è associata a una serie di errori tipici, come `UserNotFound` per indicare l'assenza di un utente richiesto o `UserCreateError` per segnalare un problema nella creazione di un nuovo utente.

Per mantenere un formato coerente, tutti gli errori seguono una mappatura predefinita, che associa ogni codice identificativo a un insieme strutturato di dettagli. Questa soluzione garantisce che ogni errore disponga delle informazioni necessarie per essere compreso dal frontend senza ambiguità. La gestione delle eccezioni è completata da un middleware centralizzato, che intercetta gli errori a tutti i livelli del backend e restituisce una risposta strutturata, includendo solo le informazioni rilevanti per il client.

Grazie a questa architettura, la gestione degli errori nel backend risulta modulare ed estensibile, permettendo di aggiungere nuove tipologie di errore senza impattare il resto del sistema.

Gestione degli errori nel frontend

Il sistema di gestione degli errori è stato progettato in modo da garantire una perfetta integrazione con il frontend. In particolare, il codice d'errore e i dettagli aggiuntivi forniti dal backend consentono al frontend di costruire messaggi chiari e contestualizzati per l'utente, senza che le stringhe siano *hard-coded* nel backend.

Un aspetto rilevante della progettazione è la compatibilità con il sistema di multilingua del frontend. La struttura dei codici di errore, infatti, segue la stessa organizzazione delle directory utilizzate per la localizzazione delle stringhe, rendendo immediata la traduzione del messaggio in base alla lingua selezionata dall'utente. Questo permette di generare notifiche e avvisi coerenti senza necessità di duplicare la logica di gestione degli errori in più parti del sistema.

4.3 Miglioramento del database

Il database del nuovo sistema è stato quasi completamente rinnovato come parte integrante del processo di reingegnerizzazione in corso. Questo intervento è stato reso possibile dal cambio di strategia descritto nella Sezione 3.2, che ha eliminato la necessità di una coesistenza tra il vecchio e il nuovo sistema, optando invece per una transizione netta. L'assenza di vincoli legati alla retrocompatibilità ha permesso di riprogettare la base dati in accordo con l'analisi del dominio, affrontando e risolvendo le criticità presenti nella precedente modellazione. Questa libertà ha consentito di introdurre una struttura più coerente e scalabile, ottimizzando la gestione delle informazioni e predisponendo il sistema per l'integrazione di nuove funzionalità.

4.3.1 Struttura della base dati

La seguente analisi descrive l'architettura del nuovo database, mettendo in evidenza le principali differenze rispetto al sistema legacy. Le entità saranno presentate in gruppi tematici, organizzati in base alla loro funzione, fornendo una visione strutturata delle relazioni e delle caratteristiche di ciascuna componente del modello dati.

Gestione delle organizzazioni, degli utenti e dei permessi

La gestione delle organizzazioni e degli utenti è centrale nel modello del database in quanto permette di strutturare il sistema in modo scalabile e multi-tenant. Ogni organizzazione definisce un'unità amministrativa indipendente, all'interno della quale operano utenti con ruoli e permessi specifici.

Struttura gerarchica delle organizzazioni Il database prevede la possibilità di modellare relazioni gerarchiche tra organizzazioni, consentendo la gestione di scenari in cui un'entità superiore controlla più sotto-organizzazioni. Questo è realizzato attraverso una relazione auto-referenziale sull'entità Organization. Pertanto, ogni organizzazione può opzionalmente riferirsi a un'organizzazione genitore, creando così una struttura ad albero. Tale approccio è particolarmente utile nei casi in cui un provider di servizi (Dealer o MSP) debba gestire clienti separati, mantenendo al contempo un controllo centralizzato.

Associazione tra utenti e organizzazioni Gli utenti nel sistema appartengono a una specifica organizzazione, stabilendo un legame chiaro tra le entità amministrative e gli individui che vi operano. Questo vincolo è fondamentale per garantire che ogni utente possa accedere solo ai dati e alle configurazioni pertinenti alla propria organizzazione, rispettando i livelli di isolamento tra clienti diversi.

Un aspetto critico del modello è la gestione dei ruoli e dei permessi, che determinano le operazioni che un utente può eseguire all'interno della propria organizzazione o di quelle subordinate. Il ruolo di ciascun utente viene interpretato direttamente dal backend del sistema e regola le azioni disponibili nell'applicazione, garantendo un controllo granulare sull'accesso alle funzionalità.

Gestione dei permessi Nel database vengono gestite due tipologie di permessi, che operano su livelli differenti (Figura 4.5):

- **RolePermission**: definiti staticamente all'interno del backend e assegnati in base alla tipologia dell'utente. Essi determinano le operazioni consentite all'interno del sistema, come la possibilità di apportare modifiche ai parametri del filtro o visualizzare determinate informazioni.

- **OrganizationPermission**: memorizzati nel database ma gestiti dalla *customer area* (un sistema esterno a quello in esame). Questi permessi sono definiti sulla base della licenza associata all'organizzazione e regolano le funzionalità accessibili per gli utenti al suo interno. Sebbene non siano amministrati direttamente dal backend, risultano fondamentali per il servizio

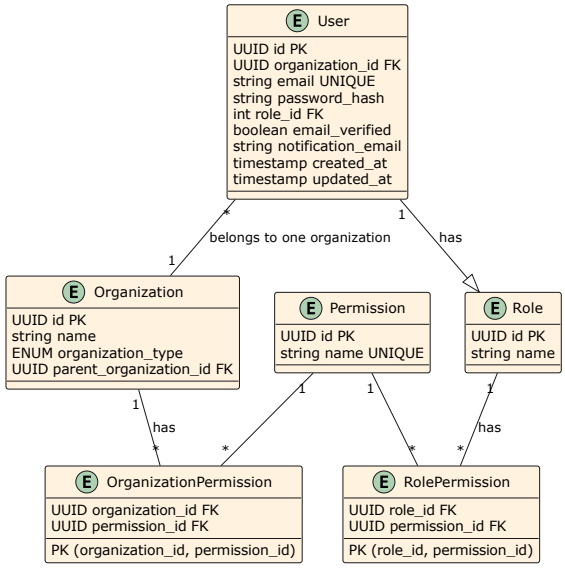

Figura 4.5: Diagramma ER relativo alla gestione delle organizzazioni, degli utenti e dei permessi.

di autorizzazione, che li utilizza per stabilire se un utente ha il diritto di eseguire una determinata operazione.

Gestione delle reti e dei profili di protezione

Il modello di database adotta un approccio basato sui profili, che rappresentano l'unico punto in cui vengono definite le policy di filtraggio. Le reti non contengono direttamente regole di protezione, ma si associano a uno o più profili per determinare quali configurazioni di sicurezza applicare.

Gestione delle reti Ogni organizzazione può registrare una o più reti, che rappresentano gli indirizzi IP dai quali provengono le richieste al sistema. Una delle principali innovazioni introdotte in questa modellazione è il supporto alla notazione CIDR, che permette di rappresentare un intero intervallo di indirizzi IP con una sola entry nel database. Questo migliora significativamente l'efficienza del sistema, riducendo il numero di righe necessarie rispetto al modello precedente, in cui ogni indirizzo IP veniva memorizzato singolarmente, anche se appartenente allo stesso range.

Per supportare scenari in cui gli indirizzi IP di una rete cambiano dinamicamente, è presente un meccanismo di aggiornamento tramite *DDNS*. Il database memorizza le informazioni necessarie per gestire queste variazioni, includendo dati di autenticazione e timestamp relativi all'ultimo aggiornamento della rete registrata.

Profili di protezione Nel nuovo database, i profili di protezione sono modellati come entità indipendenti, associate a una specifica organizzazione, ma con la possibilità di essere condivise con le organizzazioni a essa subordinate. Questo meccanismo è implementato tramite un attributo che indica se un profilo è ereditabile all'interno della gerarchia. In tal caso, esso risulta visibile e utilizzabile anche per le organizzazioni figlie, evitando la duplicazione delle configurazioni di filtraggio.

Filtri e categorie di protezione Ogni profilo può essere associato a una serie di filtri di protezione, che regolano il blocco di contenuti in base a diverse tipologie di criteri. Il sistema prevede una tabella specifica per ciascun tipo di filtro, tra cui:

- Blocco per categoria, per contenuto o singolo servizio;
- Abilitazione forzata di SafeSearch per i motori di ricerca;
- Limitazioni per indirizzi IP e domini.

La struttura del database prevede che per ogni opzione bloccata sia presente una riga separata. Ad esempio, se un profilo vieta l'accesso a cinque categorie di contenuti, la tabella relativa al blocco per categorie conterrà cinque righe distinte, ognuna riferita al profilo e alla categoria corrispondente.

Il database consente inoltre di definire i *profili ristretti* attraverso un attributo dedicato, che indica se un profilo opera in modalità restrittiva. In questa configurazione, tutte le protezioni risultano attive di default, mentre l'accesso è consentito esclusivamente ai domini autorizzati. Sebbene l'informazione sia memorizzata nel database, la logica che governa il comportamento di questi profili è demandata al livello applicativo, che interpreta tale configurazione e applica le restrizioni di conseguenza.

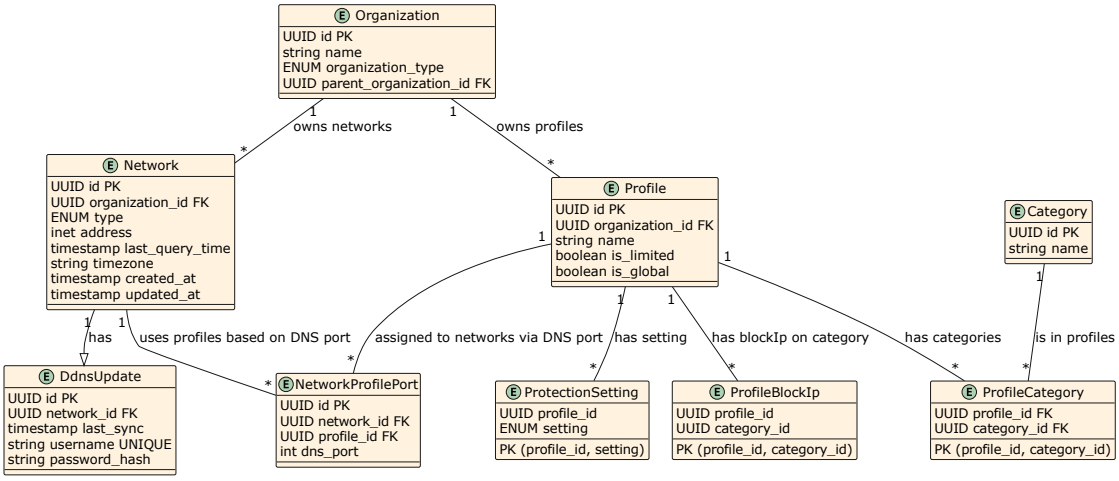

Figura 4.6: Diagramma ER relativo alla gestione delle reti e dei profili di protezione.

Associazione dei profili alle reti L'assegnazione di un profilo a una rete avviene attraverso la coppia di server DNS utilizzata per le richieste. Questo meccanismo consente di determinare quale profilo debba essere applicato in base alla configurazione DNS impostata dall'amministratore, evitando così la necessità di separare fisicamente i dispositivi su reti diverse per applicare configurazioni differenti. Nel database, questo concetto è rappresentato dal termine *porta* (Figura 4.6), una scelta terminologica che potrebbe apparire incoerente rispetto al suo significato tradizionale. Tuttavia, essa deriva da una versione precedente del sistema, in cui la selezione del profilo veniva effettivamente implementata sfruttando il numero della porta TCP.

Questa modellazione permette quindi di gestire scenari avanzati in cui più profili possono coesistere sulla stessa rete, garantendo un elevato livello di flessibilità nella definizione delle policy di filtraggio e riducendo la necessità di configurazioni ridondanti.

Gestione delle categorie e delle eccezioni

Il nuovo database introduce una modellazione più strutturata delle categorie di contenuti e delle eccezioni, con l'obiettivo di migliorare la gestione delle configurazioni di filtraggio. Rispetto alla versione precedente, questa modellazione consente

una maggiore flessibilità nella definizione delle policy, facilitando sia la gestione delle categorie predefinite sia la creazione di gruppi di eccezioni personalizzati.

Categorie e macrocategorie Le categorie rappresentano insiemi di domini classificati in base alla tipologia di contenuti, come *social media*, *streaming*, *contenuti per adulti* e così via. Per migliorare la chiarezza e la gestione delle policy, ogni categoria appartiene a una *macrocategoria*, che raggruppa più categorie affini.

L'introduzione delle macrocategorie non ha solo un valore organizzativo, ma viene sfruttata dal livello applicativo per semplificare le operazioni di configurazione. Grazie a questa struttura, un amministratore può decidere di abilitare o bloccare tutte le categorie appartenenti a una macrocategoria con un'unica azione, senza dover selezionare manualmente ogni singola voce.

Gestione delle categorie nei template Il database prevede anche la possibilità di definire insiemi preconfigurati di categorie attraverso il concetto di *Template Category*. Un template di categoria rappresenta una configurazione standardizzata che include molteplici categorie, utile per velocizzare l'applicazione di policy predefinite. L'utilizzo dei template permette di semplificare la gestione delle policy di filtraggio, evitando la necessità di selezionare manualmente ogni singola categoria. Un esempio pratico è la definizione di livelli di protezione preimpostati, con restrizioni progressive, che possono essere selezionati rapidamente senza dover configurare manualmente le singole categorie di contenuti. Questo approccio risulta particolarmente utile per fornire agli utenti meno esperti una modalità intuitiva per applicare politiche di filtraggio senza entrare nel dettaglio delle singole impostazioni.

Gestione avanzata delle eccezioni In questo contesto, una delle principali innovazioni rispetto al database precedente riguarda la gestione delle eccezioni. In passato, il sistema si basava esclusivamente su due liste distinte: una per i domini consentiti e una per i domini bloccati. Questo approccio era limitato, poiché ogni profilo doveva gestire separatamente le due liste, senza possibilità di organizzare le eccezioni in modo strutturato.

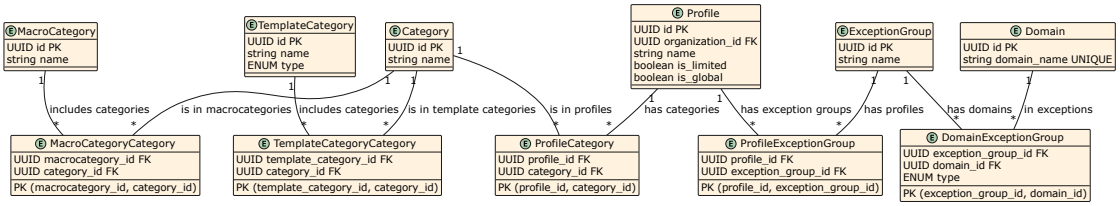

Figura 4.7: Diagramma ER relativo alla gestione delle categorie e delle eccezioni.

Il nuovo database introduce il concetto di *gruppi di eccezioni*, che rappresentano insiemi di domini personalizzati contenenti sia elementi bloccati sia consentiti. Ogni gruppo può essere associato a uno o più profili, permettendo così di riutilizzare la stessa configurazione su più contesti senza dover ridefinire manualmente le liste di riferimento. La modellazione delle eccezioni si basa su tre entità principali (Figura 4.7):

- **ExceptionGroup**: rappresenta i gruppi di eccezioni, identificati da un nome.
- **ProfileExceptionGroup**: definisce la relazione tra i profili e i gruppi di eccezioni, consentendo di applicare la stessa configurazione a più profili.
- **DomainExceptionGroup**: stabilisce l'associazione tra i domini e i gruppi di eccezioni, permettendo di definire quali appartengono a un gruppo specifico e se sono bloccati o consentiti.
- **Domain**: rappresenta la lista dei domini inseriti tra le eccezioni.

Questa modellazione consente di gestire le eccezioni in modo molto più flessibile, permettendo di definire configurazioni granulari e riutilizzabili. Un'organizzazione può, ad esempio, creare un gruppo di eccezioni che include sia domini consentiti sia domini bloccati e applicarlo a più profili senza dover replicare manualmente le stesse configurazioni. Inoltre, grazie alla presenza dei profili condivisi, anche i gruppi di eccezioni associati a questi ultimi vengono ereditati dalle organizzazioni subordinate. Questo meccanismo consente di mantenere una gestione centralizzata delle eccezioni, assicurando coerenza tra le configurazioni applicate ai diversi livelli della gerarchia organizzativa.

4.3.2 Linee guida per la migrazione dei dati

Il processo di migrazione dei dati rappresenta un'operazione critica nel passaggio dal sistema legacy alla nuova infrastruttura poiché coinvolge informazioni essenziali per il funzionamento del filtro DNS e per l'esperienza degli utenti. Affinché la transizione avvenga in modo efficace, il nuovo sistema dovrà essere già popolato con tutti i dati esistenti prima della sua messa in produzione, evitando così discontinuità operative o perdita di informazioni.

Attualmente, non è ancora stata definita nei dettagli la strategia di migrazione che si vuole adottare, poiché il nuovo sistema si trova in una fase iniziale di sviluppo. Tuttavia, è possibile delineare alcune considerazioni preliminari basate sul confronto tra la struttura del database legacy e la nuova modellazione dei dati. Queste riflessioni costituiscono un primo passo verso la progettazione di un processo di migrazione strutturato, che garantisca coerenza, integrità e compatibilità tra i due ambienti. Di seguito, verranno illustrate alcune linee guida fondamentali per la migrazione, organizzate in base alle principali entità del database. L'obiettivo è fornire un quadro chiaro delle trasformazioni necessarie e delle possibili strategie da adottare per assicurare una transizione fluida e priva di anomalie.

Prima di tutto, un aspetto fondamentale del processo in questione, che coinvolge ogni singola entità da trasferire, riguarda la gestione degli identificativi. Nel vecchio sistema, gli identificativi delle entità erano di natura numerica e seguivano il tipo di dato `Decimal(1000, 1)` di SQL. Nel nuovo sistema, si utilizzeranno invece identificativi di tipo Universally Unique IDentifier[12] (UUID), garantendo maggiore scalabilità e unicità. Di conseguenza, per ogni entità migrata, sarà necessario generare un nuovo UUID e aggiornare tutte le relazioni di conseguenza. Qualsiasi riferimento all'identificativo legacy dovrà essere tracciato temporaneamente in una tabella di mapping per facilitare il passaggio al nuovo schema.

Migrazione delle organizzazioni e degli utenti

La migrazione delle organizzazioni e degli utenti rappresenta un punto cruciale, in quanto bisogna passare da un sistema privo di multiutenza a uno in cui la funzionalità multi-tenant ha guidato lo sviluppo fin dall'inizio. Nel sistema legacy,

[12] `https://datatracker.ietf.org/doc/html/rfc9562`

non esisteva una distinzione tra organizzazione e utente: ogni riga della tabella `admin` con l'attributo `tipo = 1` rappresentava sia un cliente che il relativo utente amministratore. Questa struttura era possibile poiché vi era un singolo account per ogni cliente, eliminando la necessità di una separazione tra le due entità. Nel nuovo sistema, invece, utenti e organizzazioni sono stati modellati come entità distinte. La migrazione dovrà quindi adattare i dati esistenti a questa nuova struttura, assicurando che ogni cliente del vecchio database venga convertito in una vera e propria organizzazione, con almeno un utente amministratore associato.

In particolare, il primo passo del processo di migrazione prevede la creazione delle organizzazioni nel nuovo sistema. Ogni riga della tabella `admin` con `tipo = 1` dovrà essere trasformata in una entry nella corrispondente tabella del nuovo database. Gli attributi identificativi dell'organizzazione, come il nome, dovranno essere trasferiti e adattati alla nuova struttura.

Per ricreare la gerarchia delle organizzazioni sarà necessario analizzare il valore del campo `cloud_idfornitore` della tabella `admin`. Questo campo, se valorizzato, contiene l'indirizzo email dell'organizzazione padre, ovvero del cliente che gestisce l'entità in questione. Se il campo risulta vuoto, significa che l'organizzazione non è subordinata ad altre ed è quindi posizionata al livello più alto della gerarchia, come nel caso di Dealer o MSP. Nel nuovo sistema, l'associazione tra organizzazioni non avviene più tramite email, ma attraverso l'UUID dell'organizzazione padre. Pertanto, nel processo di migrazione, sarà necessario risolvere queste relazioni traducendo il valore di `cloud_idfornitore` nel corrispondente UUID dell'organizzazione padre. Questo passaggio dovrà essere eseguito in due fasi:

1. Creare tutte le organizzazioni migrando i dati dalla tabella `admin`, senza impostare subito il campo `parent_organization_id`.

2. Dopo che tutte le organizzazioni sono state inserite nel nuovo database, aggiornare il campo `parent_organization_id` associando a ciascuna il corrispondente UUID dell'organizzazione padre, risolvendo la relazione tramite il valore presente in `cloud_idfornitore`.

Questa strategia garantisce che la gerarchia venga ricostruita correttamente senza incongruenze nella fase di migrazione.

Successivamente, per ogni organizzazione migrata, dovrà essere creato il corrispondente utente amministratore. Quest'ultimo dovrà essere associato all'organizzazione di riferimento tramite una chiave esterna, garantendo così una chiara relazione tra le due entità. Poiché nel sistema legacy l'utente e l'organizzazione coincidevano, sarà necessario separare logicamente i dati, assicurandosi che il nuovo sistema mantenga coerenza e integrità.

Migrazione dei profili di protezione

La migrazione dei profili di protezione rappresenta una delle operazioni più complesse in quanto il nuovo sistema ha introdotto il concetto di profili condivisi, modificando radicalmente la struttura e le modalità di associazione delle configurazioni di filtraggio.

Nel sistema legacy, i profili erano identificati tramite un *codice profilo* e associati ai clienti attraverso la tabella `admin`, con un legame statico e univoco. Questo significava che ogni profilo apparteneva esclusivamente a un singolo cliente, senza possibilità di essere condiviso tra più entità. Nel nuovo sistema, invece, i profili non sono più legati a una singola istanza del vecchio database, ma vengono associati direttamente alle organizzazioni, con la possibilità di essere riutilizzati dalle organizzazioni subordinate.

Sebbene sia teoricamente possibile effettuare un'analisi per identificare i potenziali profili da trasformare in condivisi, questa operazione risulta particolarmente complessa. Nel database legacy, infatti, non esiste un'informazione chiara e diretta che indichi se un profilo sia utilizzato da più clienti. Questo rende difficile stabilire automaticamente quali profili potrebbero essere condivisi senza introdurre ambiguità o errori nella migrazione. Per garantire una transizione coerente, si può quindi procedere migrando tutti i profili esistenti e impostandoli inizialmente come non condivisibili, riproducendo fedelmente la configurazione che gli utenti avevano nel vecchio pannello. Una volta completata la migrazione, saranno gli stessi amministratori ad avere la possibilità di convertire i profili tradizionali in condivisi e riutilizzarli per più sottoclienti, sfruttando le nuove funzionalità del sistema.

Migrazione delle categorie di blocco e delle eccezioni

La migrazione delle categorie di blocco avverrà trasferendo i dati esistenti nelle nuove tabelle, assicurando che le associazioni tra profili e categorie rimangano invariate. Nel sistema legacy, l'associazione tra profilo e categoria era effettuata tramite il codice profilo collegato alla relativa entità nella tabella `admin`, mentre nel nuovo sistema viene utilizzato l'UUID del profilo. Di conseguenza, la migrazione dovrà prevedere la conversione dei riferimenti al codice profilo nel nuovo identificativo univoco, garantendo la coerenza con la nuova struttura dati.

Un aspetto più critico riguarda la gestione delle eccezioni, che nel nuovo sistema introduce gruppi riutilizzabili anziché liste statiche per ogni profilo. Per garantire una transizione coerente, la migrazione dovrà trasformare le eccezioni esistenti adattandole alla nuova struttura. Una possibile soluzione consiste nel creare un gruppo di eccezioni predefinito per ogni profilo, popolandolo con i domini bloccati e consentiti presenti nel vecchio database. Questo approccio mantiene le configurazioni originali, offrendo al contempo la maggiore flessibilità del nuovo sistema.

Migrazione delle reti

La migrazione delle reti richiede un processo di consolidamento degli indirizzi IP per adattarli alla nuova rappresentazione basata sulla notazione CIDR. Anziché memorizzare ogni singolo indirizzo come una entry separata, il nuovo sistema consente di rappresentare un intero range di IP in modo compatto, riducendo la ridondanza e ottimizzando lo spazio di archiviazione.

Per garantire una transizione coerente, la migrazione dovrà prevedere la conversione degli indirizzi IP esistenti nella loro corrispondente rappresentazione CIDR. Questo comporta l'aggregazione degli indirizzi appartenenti allo stesso range e la loro trasformazione in un'unica entry nel nuovo database. Tale processo migliorerà le operazioni di gestione e ricerca delle reti, riducendo significativamente il numero di record memorizzati.

Capitolo 5

Implementazione

Questo capitolo illustra il piano di sviluppo del nuovo sistema, evidenziando gli elementi tecnici di maggiore rilievo per il valore complessivo della presente tesi. In particolare, viene analizzata la gestione degli errori, descrivendo le soluzioni adottate sia nel backend sia nel frontend. L'approfondimento proposto amplia e integra quanto discusso nella Sezione 4.2.2, dove era stata esaminata la fase di progettazione in modo preliminare.

5.1 Piano di Sviluppo

Lo sviluppo del nuovo sistema è stato condotto seguendo la metodologia Agile *SCRUM*[1], che prevede un'organizzazione del lavoro in cicli iterativi e incrementali della durata di due settimane (*sprint*). Questo approccio ha permesso di suddividere il progetto in incrementi funzionali rilasciabili progressivamente, favorendo una maggiore flessibilità nei confronti dei requisiti emergenti e un continuo miglioramento del prodotto.

Al termine di ogni sprint, si è tenuto un meeting di revisione con il *Project Manager* (PM), durante il quale sono state analizzate le criticità emerse, verificata la conformità delle ore lavorate rispetto a quelle preventivate e pianificate le attività per lo sprint successivo. Questo processo ha garantito un flusso di sviluppo costante

[1] `https://scrumguides.org/scrum-guide.html`

e ben organizzato, consentendo al team di intervenire tempestivamente su eventuali problematiche e di mantenere un'elevata qualità del codice.

Il team di sviluppo è composto esclusivamente da *Full Stack Developer*, senza una suddivisione rigida tra frontend e backend. Le attività di progettazione grafica non sono state gestite internamente, ma affidate a un designer esterno incaricato del rinnovo della brand identity. Di conseguenza, per quanto riguarda la UI/UX, il team si è occupato esclusivamente dell'implementazione delle nuove interfacce grafiche, attenendosi alle specifiche fornite dal designer e garantendo la corretta integrazione delle soluzioni proposte all'interno del sistema.

Per la gestione e il monitoraggio delle attività, è stato utilizzato lo strumento di project management *ClickUp*[2], che ha permesso di tracciare i task, assegnare priorità e garantire un'organizzazione efficace del lavoro.

Le attività svolte durante gli sprint hanno incluso lo sviluppo di nuove funzionalità, la risoluzione di bug, il refactoring del codice e l'ottimizzazione delle performance. Particolare attenzione è stata dedicata alla qualità del software, con l'integrazione di test automatici e revisioni periodiche del codice prima del rilascio delle feature.

5.1.1 Ambiente di sviluppo e infrastruttura

L'ambiente di sviluppo è stato configurato in modo da rispecchiare il più possibile l'ambiente di produzione, garantendo una maggiore affidabilità nei test e riducendo il rischio di anomalie legate a differenze infrastrutturali. Per ottenere questa uniformità, sono state adottate tecnologie che permettono di replicare fedelmente la configurazione di produzione.

L'intero sistema di sviluppo è stato *containerizzato*[3] tramite *Docker*, con i vari servizi orchestrati da *Docker Compose*. Questo permette di avviare l'ambiente di sviluppo in modo rapido e riproducibile su diverse macchine, riducendo problemi di configurazione tra membri del team. Inoltre, l'uso di *Docker-in-Docker* nel-

[2] `https://clickup.com`

[3] Il termine "containerizzare" si riferisce al processo di incapsulamento di un'applicazione e delle sue dipendenze all'interno di un container, ovvero un'unità standardizzata di software che garantisce che l'applicazione possa essere eseguita in modo coerente in qualsiasi ambiente. (`https://www.docker.com/resources/what-container`)

la pipeline *Continuous Integration/Continuous Deployment* (CI/CD) consente di replicare l'ambiente di esecuzione anche nei *job* automatici di test e build.

Per la gestione delle configurazioni, ogni microservizio utilizza un file `.env`, che contiene le variabili d'ambiente necessarie per la configurazione dinamica dello stesso, come le credenziali di accesso al database e gli endpoint delle API. Durante l'esecuzione della pipeline CI/CD, questo file viene iniettato nei container, permettendo di mantenere un ambiente coerente tra sviluppo, test e produzione, senza la necessità di configurazioni separate per ciascun contesto.

Per migliorare l'affidabilità dei test, è stata implementata una strategia di *seeding* del database, che consente di popolare l'ambiente di test con dati coerenti a ogni esecuzione, assicurando risultati riproducibili e verifiche affidabili.

5.1.2 Struttura della repository

L'intero codice del progetto è gestito tramite una *monorepo* ospitata su *GitLab*. La scelta di adottare una monorepo è motivata dalla necessità di mantenere in un unico repository sia il frontend che tutti i microservizi del backend, semplificando la gestione delle dipendenze, la coerenza tra i moduli e l'esecuzione delle pipeline di CI/CD.

Per ottimizzare la gestione della monorepo, è stato utilizzato `Turborepo`[4], un tool specificamente progettato per lo sviluppo di applicazioni in TypeScript. Questo strumento consente di affrontare in modo efficiente il problema dei lunghi processi di compilazione, tipici delle monorepo con numerosi pacchetti e molteplici task (come compilazione, testing e linting). Il tool pianifica l'esecuzione dei task in modo ottimizzato, parallelizzandoli su tutti i cores disponibili e implementando un avanzato sistema di caching. Questo permette di ridurre drasticamente i tempi di build successivi al primo, migliorando la produttività del team di sviluppo.

L'organizzazione della monorepo sfrutta la funzionalità *workspace* di `pnpm`, suddividendo i pacchetti in due macrocategorie:

- **Apps**: Contiene tutti i servizi che vengono eseguiti in maniera indipendente e che costituiscono i diversi microservizi del sistema. Qui sono presenti il frontend e tutti i componenti del backend che operano come unità autonome.

[4] `https://turbo.build`

- **Packages**: Include pacchetti di supporto utilizzati dalle *apps*. Tra questi vi sono il package `commons`, che contiene definizioni e metodi condivisi tra i vari servizi, i pacchetti dedicati alla gestione dello schema di *Prisma* e all'esportazione del client per il database, oltre ai pacchetti per il seeding di quest'ultimo nella fase di test.

Modello di branching

Per organizzare al meglio il lavoro di sviluppo del team, è stato adottato un modello di branching strutturato, che consente di gestire in modo chiaro e controllato il ciclo di vita del codice. Esso prevede i seguenti branch principali:

- **Main**: Contiene il codice stabile e rappresenta l'unico branch dal quale si effettuano rilasci in produzione.

- **Unstable**: Include le funzionalità candidate al rilascio in produzione. Il codice qui presente è testato in un ambiente di staging prima di essere eventualmente integrato nel branch *main*.

- **Branch personali**: Ogni sviluppatore lavora principalmente su un branch personale, denominato con il proprio cognome, per eseguire i commit delle proprie modifiche prima di unire il codice nei branch condivisi.

- **Feature branches**: Per lo sviluppo di funzionalità specifiche, possono essere creati branch temporanei, indipendenti dai branch personali, in modo da favorire un'organizzazione più modulare del codice.

Questa strategia consente di mantenere un flusso di sviluppo ordinato, con una chiara separazione tra il codice in produzione, il codice in fase di test e le modifiche in sviluppo. Inoltre, grazie all'integrazione con le pipeline CI/CD di GitLab, il sistema è in grado di eseguire automaticamente test e build per ogni *push* e *merge request*, garantendo un'elevata affidabilità prima della promozione del codice verso l'ambiente di produzione.

Pipeline CI/CD

L'automazione del processo di sviluppo è stata realizzata attraverso una pipeline di *CI/CD*, integrata direttamente in GitLab. La pipeline è suddivisa in più fasi, organizzate per garantire una validazione progressiva del codice prima del rilascio in staging.

L'intero processo di build e test si basa su Docker-in-Docker[5] (DinD), una soluzione che permette alla pipeline di eseguire e gestire container Docker all'interno di ambienti di esecuzione basati sulla stessa tecnologia. Questo approccio consente di costruire immagini Docker direttamente nei job della pipeline, riducendo la dipendenza da infrastrutture esterne e garantendo isolamento e una maggiore coerenza tra gli ambienti di sviluppo, test e deploy. Il flusso di lavoro della pipeline prevede le seguenti fasi:

1. **Build dell'ambiente di sviluppo**: I servizi vengono compilati e avviati tramite il comando `docker-compose`, con supporto alla parallelizzazione dei task e caching per ottimizzare i tempi di build.

2. **Esecuzione dei test**: Ogni microservizio è testato in container isolati. Viene eseguito il processo di seeding del database per simulare scenari reali e ottenere una copertura completa del codice.

3. **Build per la produzione**: Se i test risultano superati, viene generata una build ottimizzata per il rilascio, con tagging e push delle immagini Docker nel registry di GitLab.

4. **Deploy sull'ambiente di staging**: Il codice viene distribuito aggiornando i container in esecuzione e ripristinando il database con i dati necessari per il test pre-produzione.

La pipeline utilizza strategie di caching avanzate per ridurre i tempi di esecuzione. I pacchetti gestiti da `pnpm` e la cache di `Turborepo` vengono riutilizzati tra i job, evitando ricompilazioni non necessarie. Inoltre, il sistema di *retry* proprio

[5] `https://www.docker.com/resources/docker-in-docker-containerized-ci-workflows-dockercon-2023`

della pipeline, assicura la ripetizione automatica dei job in caso di errori transitori, migliorando l'affidabilità del processo.

Grazie a questa configurazione è possibile garantire un flusso di sviluppo strutturato e sicuro, assicurando che solo codice stabile e testato venga promosso verso l'ambiente di staging e, successivamente, verso quello di produzione.

5.1.3 Test e controllo qualità

La verifica della qualità del codice e la copertura tramite test automatizzati hanno rappresentato un aspetto fondamentale del piano di sviluppo. Il processo di testing è stato strutturato seguendo queste linee guida:

- Garantire una copertura del 100% del codice sorgente dell'intero sistema, attraverso test unitari e di integrazione.

- Mantenere standard elevati di qualità del codice attraverso strumenti di analisi statica come `ESLint`[6] e `Prettier`[7], utilizzati per il rispetto delle best practices di sviluppo (il primo) e per il controllo della formattazione (il secondo).

- Valutare le performance e il comportamento del sistema in scenari critici mediante test prestazionali.

- Per il backend, è stata utilizzata la libreria `Mocha`[8], che consente di eseguire test unitari e di integrazione in un ambiente controllato, verificando il corretto funzionamento delle API e della logica applicativa.

- Per il frontend, è stato adottato `Playwright`[9], uno strumento per il testing end-to-end che permette di simulare interazioni utente su diversi browser.

[6] `https://eslint.org`
[7] `https://prettier.io`
[8] `https://mochajs.org`
[9] `https://playwright.dev`

5.1.4 Approccio alla reingegnerizzazione

Come descritto nella Sezione 2.2.3, ogni modello di reingegnerizzazione presenta vantaggi e criticità differenti. Il processo adottato per lo sviluppo del nuovo sistema si pone a metà tra l'approccio "Big Bang" e quello "Evolutivo" in quanto procede in modo incrementale, senza tuttavia una fase di coesistenza con il vecchio. Una volta completato, esso sostituirà integralmente il sistema precedente, fatta eccezione per alcune funzionalità che continueranno a essere gestite dal legacy senza necessità di retrocompatibilità.

La transizione avverrà in modo diretto, senza un rilascio graduale in produzione. Questo semplifica la migrazione, evitando la necessità di interfacce di compatibilità tra i due sistemi. Tuttavia, a differenza di un classico approccio "Big Bang", lo sviluppo non è stato affrontato come una riscrittura monolitica. Il nuovo sistema, infatti, è stato progettato fin dall'inizio con un'architettura a microservizi, organizzata sulla base delle funzionalità piuttosto che sulla replica della struttura esistente.

Questo approccio ibrido permette di bilanciare i vantaggi dei due modelli. La sostituzione completa del vecchio sistema elimina la necessità di mantenere allineate due versioni in parallelo, riducendo la complessità operativa. Allo stesso tempo, l'architettura modulare migliora la manutenibilità e facilita l'integrazione di nuove tecnologie nel tempo. Questa fusione di metodologie garantisce una transizione più controllata, riducendo il rischio di regressioni e assicurando una maggiore stabilità per il sistema finale.

5.2 Implementazione della gestione degli errori

La gestione degli errori rappresenta un aspetto cruciale dell'implementazione del nuovo sistema, garantendo uniformità e coerenza tra backend e frontend. Come descritto nella Sezione 4.2.2 del capitolo precedente, il design della gestione degli errori è stato strutturato attorno a un modello tipizzato e scalabile, che consente di categorizzare le anomalie in base alla loro natura e al contesto in cui si verificano. In questa sezione verranno discussi gli aspetti salienti relativi all'imple-

mentazione di questo modello, partendo dalla struttura del backend e analizzando successivamente il meccanismo in funzione del frontend.

5.2.1 Gestione degli errori nel backend

Nel backend, la gestione degli errori è stata implementata attraverso una struttura tipizzata e centralizzata, che garantisce una rappresentazione coerente delle anomalie e facilita la comunicazione con il frontend. Il sistema si basa su una gerarchia di classi che categorizzano le diverse tipologie di errore, sfruttando appieno il sistema di tipi di TypeScript per garantire sicurezza e scalabilità. In questa sezione si analizzeranno nel dettaglio le componenti principali dell'implementazione, tra cui il meccanismo di mappatura degli errori tramite `ErrorDetailsMapping`, la classe base `AppError` e il tipo `AppErrorPayload`.

Struttura di `ErrorDetailsMapping`

Un elemento centrale nella gestione degli errori è rappresentato dalla struttura `ErrorDetailsMapping`, che definisce una mappatura tra i codici d'errore e le informazioni associate a ciascuno di essi. Questo meccanismo consente di tipizzare in modo rigoroso le eccezioni, garantendo una gestione strutturata e uniforme tra backend e frontend.

Il tipo `ErrorDetailsMapping` è stato realizzato utilizzando il concetto di *Discriminated Union*[10], una tecnica avanzata di TypeScript che consente di associare a ogni codice d'errore una struttura dati ben definita. Questo approccio elimina il rischio di assegnare dettagli incoerenti o incompleti, assicurando che ogni errore sia rappresentato in modo chiaro e prevedibile. Di seguito è riportata una porzione di codice relativa al tipo `ErrorDetailsMapping`:

Listing 5.1: Porzione di `ErrorDetailsMapping`

```
type ErrorDetailsMapping = {
  // --- Validation errors ---
  "validation.input_error": {
    entity: ErrorEntity;
    operation: ErrorOperation;
```

[10] https://www.typescriptlang.org/docs/handbook/2/narrowing.html#discriminated-unions

```
    inputErrors: InputError[];
  };

  // --- User entity errors ---
  "user.not_found": { userId: string };
  "user.create_failed": {
    cause: CreateErrorCause;
  };

  // --- Authentication errors ---
  "authentication.unauthorized": { message?: string };
  "authentication.forbidden": { action?: string };

  // --- Generic internal server errors ---
  "internal.server_error": { message?: string };
  "internal.legacy_error": { message?: string };
  "internal.database_error": { message?: string };
};
```

La mappatura degli errori è organizzata in categorie, ciascuna identificata da un prefisso che segue una convenzione precisa. Ad esempio, gli errori di validazione degli input sono contrassegnati dal prefisso "`validation`", mentre quelli relativi alla gestione delle entità utilizzano prefissi come "`user`" o "`organization`". Questa scelta non è casuale: i prefissi sono stati definiti in modo da rispecchiare la struttura delle cartelle nel frontend, che organizza i file di traduzione per le stringhe di testo presenti nell'applicazione. In questo modo, la corrispondenza tra i codici di errore del backend e le chiavi di traduzione nel frontend è diretta e immediata.

Ogni chiave in `ErrorDetailsMapping` è associata a un oggetto che rappresenta i dettagli dell'errore corrispondente. La struttura di questi dettagli varia in base al tipo di errore: alcuni codici, come "`user.not_found`", sono associati a un semplice oggetto contenente un identificativo, ad esempio il campo `userId`, che indica l'utente non trovato. Altri errori, invece, richiedono una descrizione più articolata del problema.

Un esempio di questa maggiore complessità si trova negli errori relativi alla creazione di un'entità, come "`user.create_failed`". In questo caso, l'oggetto associato include un campo `cause`, che fornisce una rappresentazione schematica della causa del fallimento. Questo campo non è un semplice identificativo, ma può assumere valori differenti a seconda della natura dell'errore. La sua definizione è

basata sugli *Union Types*[11] di Typescript, ovvero un'unione di tipi che permettono di distinguere tra cause generali, come "`already_exists`", e problemi più specifici legati al database, come violazioni di unicità o vincoli di chiave esterna. La seguente definizione mostra come queste categorie di errore sono tipizzate:

Listing 5.2: Definizione del tipo `cause` per operazioni di creazione

```
type DatabaseErrorCause = "unique_violation" | "foreign_key_violation";
type CreateErrorCause = "already_exists" | DatabaseErrorCause;
```

Questo approccio offre un controllo statico rigoroso sugli errori generati dal backend, assicurando che ogni valore di `CreateErrorCause` sia riconducibile a una stringa predefinita che rappresenta la causa esatta dell'errore. Ciò evita l'uso di messaggi arbitrari e favorisce una comunicazione strutturata tra backend e frontend. Un ulteriore vantaggio di questa struttura è la sua leggibilità. I codici di errore e le relative cause sono stati progettati per essere facilmente convertibili in stringhe comprensibili, consentendo al frontend di generare messaggi chiari per l'utente finale. Questo meccanismo centralizza la gestione delle stringhe di errore, riducendo la necessità di definizioni ridondanti e migliorando la manutenibilità del sistema. L'adozione delle discriminated unions garantisce inoltre che:

- Ogni codice d'errore sia associato ai dati corretti, evitando ambiguità.
- Il frontend possa interpretare le informazioni senza controlli manuali sui tipi.
- L'aggiunta di nuovi errori sia semplice e non impatti il codice esistente.

Classe `AppError`

La gestione degli errori nel backend si basa anche su una classe astratta denominata `AppError`, che funge da base per la definizione di tutte le eccezioni applicative. Questa classe fornisce un modello coerente per la rappresentazione degli errori, assicurando che ogni anomalia venga espressa attraverso un formato strutturato e facilmente interpretabile dal sistema. L'obiettivo principale è quello di centralizzare la gestione degli errori, evitando soluzioni dispersive e garantendo una chiara separazione tra la logica applicativa e il trattamento delle anomalie.

[11] `https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#union-types`

Per garantire la massima flessibilità e sicurezza dei tipi, la classe `AppError` sfrutta i *Generics*[12] di TypeScript. Questo approccio consente di vincolare ogni istanza di errore a un preciso codice identificativo, associandolo a un insieme di dettagli specifici definiti nel tipo `ErrorDetailsMapping`. Grazie a questa struttura, ogni errore è caratterizzato da tre elementi principali: un codice univoco, uno stato HTTP corrispondente e un set di dettagli contestuali che forniscono informazioni aggiuntive sull'errore.

Listing 5.3: Implementazione della classe `AppError`

```
1  abstract class AppError<
2    K extends keyof ErrorDetailsMapping,
3  > extends Error {
4    public readonly code: K;
5    public readonly statusCode: number;
6    public readonly details: ErrorDetailsMapping[K];
7
8    protected constructor(code: K, details: ErrorDetailsMapping[K]) {
9      super();
10     this.code = code;
11     this.statusCode = getStatusCodeFromErrorCode(this.code, details);
12     this.details = details;
13   }
14
15   toJSON(): AppErrorPayload {
16     return {
17       code: this.code,
18       details: this.details,
19     } as AppErrorPayload;
20   }
21 }
```

Definizione del tipo generico per `AppError` Un elemento fondamentale dell'implementazione della classe `AppError` è l'uso della keyword `keyof`[13] durante la definizione del tipo generico `K`. Questo costrutto di TypeScript, visionabile a riga 2 del Listing 5.3, consente di ottenere un'unione di stringhe che rappresentano tutti i possibili codici d'errore previsti nel sistema. In altre parole, `keyof ErrorDetailsMapping` genera dinamicamente un insieme di stringhe che

[12] `https://www.typescriptlang.org/docs/handbook/2/generics.html`
[13] `https://www.typescriptlang.org/docs/handbook/2/keyof-types.html`

corrispondono esattamente ai nomi delle chiavi definite in `ErrorDetailsMapping`. Consideriamo, ad esempio, il seguente sottoinsieme di codici di errori:

Listing 5.4: Esempio di unione di stringhe generate da `keyof`

```
"validation.input_error" | "internal.database_error" | "user.create_failed"
```

Grazie a questo meccanismo, il parametro generico `K` della classe `AppError` è automaticamente limitato ai valori definiti in `ErrorDetailsMapping`. In questo modo, qualsiasi istanza della classe d'errore può essere creata solo con un codice conforme alle specifiche del sistema, evitando l'utilizzo di stringhe arbitrarie che potrebbero causare errori di inconsistenza.

Vincolo sul tipo dei dettagli dell'errore Oltre a garantire che il codice di errore appartenga a un insieme predefinito, il tipo generico `K` consente anche di vincolare il formato dei dettagli associati all'errore. Ogni codice d'errore definito in `ErrorDetailsMapping` è associato a un oggetto contenente informazioni specifiche per quel particolare tipo di anomalia. La classe `AppError` assicura che il campo `details` rispetti esattamente la struttura prevista per il codice d'errore selezionato. Ad esempio, se si crea un'istanza di errore per rappresentare un utente non trovato, TypeScript garantirà che i dettagli contengano esclusivamente le proprietà previste per quel tipo di errore. Il codice qui sotto mostra un caso d'uso concreto:

Listing 5.5: Esempio di utilizzo della classe `AppError`

```
class UserNotFoundError extends AppError<"user.not_found"> {
  constructor(userId: string) {
    super("user.not_found", { userId });
  }
}

const userError = new UserNotFoundError({ userId: "1403-2410" });
console.log(userError.code);    // "user.not_found"
console.log(userError.details); // { userId: "1403-2410" }
```

In questo caso, il codice dell'errore è limitato alla stringa "`user.not_found`", e i dettagli sono strutturati esattamente come specificato in `ErrorDetailsMapping`. Se si tentasse di fornire un oggetto dettagli con proprietà non conformi, TypeScript genererebbe un errore a tempo di compilazione, impedendo di introdurre inconsistenze nel sistema.

Definizione del tipo `AppErrorPayload` Un altro elemento centrale di `AppError` è il metodo `toJSON()`, il quale ha il compito di serializzare l'errore in un formato compatibile con la comunicazione tra backend e frontend. Questo metodo restituisce un oggetto che segue la struttura imposta dal tipo `AppErrorPayload`, di seguito definito:

Listing 5.6: Definizione del tipo `AppErrorPayload`

```
1 type AppErrorPayload = {
2   [K in keyof ErrorDetailsMapping]: {
3     code: K;
4     details: ErrorDetailsMapping[K];
5   };
6 }[keyof ErrorDetailsMapping];
```

La sintassi utilizzata in questa definizione sfrutta alcune funzionalità avanzate di TypeScript:

- **Mapped Types**: la parte di codice a linea 2 itera su ciascuna chiave `K` presente in `ErrorDetailsMapping`. Per ogni chiave, viene creato un nuovo *object* avente due proprietà:
  - `code` di tipo `K`, che rappresenta il codice d'errore;
  - `details` di tipo `ErrorDetailsMapping[K]`, che contiene i dettagli specifici associati a quel codice.
- **Indexed Access Types**: l'intera mappatura è poi indicizzata con `[keyof ErrorDetailsMapping]` a linea 6. Questa operazione estrae dall'oggetto mappato una unione di tutti i tipi generati per ogni chiave. In altre parole, il risultato finale è l'unione dei tipi object definiti per ciascuna chiave in `ErrorDetailsMapping`.

In sintesi, `AppErrorPayload` rappresenta uno union type, in cui ogni membro dell'unione è un oggetto con due proprietà:

- `code`, il quale contiene un identificativo dell'errore;
- `details`, il quale fornisce informazioni specifiche associate a quell'errore, basate sulla mappatura definita in `ErrorDetailsMapping`.

Questo design consente a `AppError` di gestire in modo robusto e tipizzato una serie di errori diversi, garantendo che ognuno di essi abbia una struttura dati coerente e ben definita.

Gestione degli errori di validazione degli input

Nel backend, la validazione degli input ricevuti dalle API è affidata alla libreria `Zod`[14], la quale fornisce un sistema di definizione di schemi e di verifica automatica della conformità dei dati. Quando un input non rispetta lo schema previsto, Zod genera un'eccezione di tipo `ZodError`, che contiene informazioni dettagliate sugli errori riscontrati. Tuttavia, questo errore non è direttamente utilizzabile per la comunicazione con il frontend, motivo per cui è necessario un processo di elaborazione che trasformi i dati forniti da Zod in un formato strutturato e coerente con il modello di gestione degli errori che si sta discutendo. A tale scopo, il backend intercetta l'errore lanciato da Zod e lo elabora per costruire un oggetto di tipo `InputError`. Questo oggetto contiene:

- Il nome del campo che ha fallito la validazione;
- Un insieme di dettagli strutturati che permettono di comprendere la natura dell'errore.

Questi dettagli includono il codice d'errore specifico di Zod (come "`too_small`", "`invalid_string`", "`unrecognized_keys`", ecc.), il valore atteso e quello effettivamente ricevuto, e altri parametri che variano a seconda del tipo di errore (come i limiti minimi e massimi nel caso di valori numerici o stringhe con vincoli di lunghezza). Ancora una volta l'obiettivo è quello di produrre un'informazione chiara e strutturata, che possa essere facilmente interpretata dal frontend per la generazione di messaggi d'errore dettagliati relativi agli input non validi.

Il risultato di questa elaborazione viene quindi incluso nei `details` dell'errore con codice "`validation.input_error`", accompagnato dall'entità coinvolta e dal tipo di operazione che ha generato la validazione fallita. In questo modo, il frontend, ricevendo un errore di questo tipo, è in grado di analizzare i dettagli

[14] `https://zod.dev`

forniti e costruire dinamicamente un messaggio specifico per ogni campo errato, mantenendo la piena compatibilità con il sistema di internazionalizzazione.

Considerazioni finali

La gestione degli errori nel backend è stata strutturata in maniera modulare e tipizzata, garantendo uniformità e scalabilità. L'uso dei *generics* di TypeScript e della classe `AppError` ha permesso di standardizzare la rappresentazione delle anomalie, facilitando l'integrazione con il frontend. Grazie alla mappatura definita in `ErrorDetailsMapping`, ogni errore include informazioni specifiche e contestualizzate, assicurando una comunicazione chiara tra i livelli del sistema. Inoltre, la gestione degli errori di validazione tramite Zod permette di trasformare le anomalie sugli input in messaggi strutturati e facilmente interpretabili.

Questa architettura garantisce che il backend restituisca risposte coerenti e prevedibili, semplificando la generazione dei messaggi d'errore lato client e migliorando la manutenibilità complessiva del sistema.

5.2.2 Gestione degli errori nel frontend

Nel frontend, la gestione delle anomalie provenienti dal backend è stata progettata per garantire una comunicazione efficace e multilingua nei messaggi destinati all'utente. Gli errori restituiti dai vari microservizi, come è già ben noto, non possono essere utilizzati direttamente per fornire un feedback comprensibile, poiché si limitano a rappresentare il problema in modo schematico. Per questo motivo, è necessario un sistema di interpretazione che analizzi le proprietà dell'oggetto errore e generi dinamicamente un messaggio esplicativo.

Definizione e utilizzo di `Translator` Per garantire un efficace supporto alla traduzione dei messaggi di errore, è stato definito il tipo `Translator`, che rappresenta una funzione generica in grado di convertire una chiave testuale e un insieme opzionale di parametri in una stringa localizzata. Questa astrazione consente di mantenere il sistema di gestione degli errori indipendente dalla libreria specifica utilizzata per la multilingua, rendendolo più flessibile ed estensibile. La definizione di `Translator` è riportata nel Listing 5.7.

Per integrare questo meccanismo nei componenti del frontend, è stato realizzato un apposito hook denominato `useTranslator`, che fornisce un'interfaccia uniforme per la traduzione dei messaggi, senza vincolare il codice a una particolare implementazione. Questo hook consente inoltre di specificare un prefisso per le chiavi di traduzione, facilitando l'organizzazione e la gestione delle stringhe localizzate.

Listing 5.7: Definizione del tipo `Translator` e dell'hook `useTranslator`

```
type Translator = (key: string, params?: TranslationValues) => string;

const useTranslator = (translationsPrefixPath = ""): Translator => {
  const t = useTranslations(translationsPrefixPath);
  return (key: string, values?: TranslationValues): string => t(key, values);
};
```

Grazie a questa astrazione, ogni componente del frontend può ottenere facilmente un'istanza della funzione di traduzione senza dipendere direttamente da una libreria specifica. Questo approccio migliora la modularità e la manutenibilità del codice, permettendo di sostituire o aggiornare il sistema di internazionalizzazione senza modificare la logica di gestione degli errori.

Costruzione dinamica dei messaggi di errore

Nel frontend, ogni volta che viene effettuata una chiamata HTTP al backend, se la risposta ricevuta ha un codice di stato HTTP diverso da `2XX`, il sistema sa che si è verificato un errore e attiva il meccanismo di generazione del messaggio corrispondente. Per fare ciò, viene analizzato l'oggetto di tipo `AppErrorPayload` che rappresenta il risultato dell'invocazione di `toJSON()` da parte del backend su un'istanza di `AppError`.

Un primo livello di gestione è implementato attraverso l'invocazione del metodo `parseBackendError()`, che, sulla base del codice di stato HTTP ricevuto, chiama metodi specifici per la costruzione del messaggio di errore appropriato. In particolare, esso opera come segue:

- Se lo stato HTTP è `400`, si tratta di un errore di validazione. In questo caso, il metodo estrae i dettagli dell'errore dalla risposta del backend e, se il codice restituito è "`validation.input_error`", accede alla proprietà `inputErrors` all'interno dei `details`. Questo array contiene l'elen-

co dei campi che non hanno superato la validazione e le relative anomalie. Successivamente, il metodo `getValidationErrorMessage()` elabora queste informazioni per generare un messaggio chiaro e contestualizzato per l'utente.

- Se lo stato HTTP è `409`, si tratta di un errore di conflitto, tipicamente legato a operazioni fallite su entità esistenti. In questo scenario, il metodo analizza il codice del problema, estraendo il nome dell'entità coinvolta e il tipo di errore. Successivamente, invoca metodi specifici come `getUserErrorMessage()`, che costruiscono il messaggio contestualizzato all'entità coinvolta.

- Se lo stato HTTP è `500` o un altro valore non gestito, il metodo restituisce un messaggio generico di errore, indicando all'utente che si è verificato un problema imprevisto. Questo è l'unico caso in cui i messaggi d'errore vengono costruiti lato backend, in quanto non è possibile prevedere tutti i possibili scenari. Questi messaggi sono in lingua inglese e non vengono tradotti.

Gestione degli errori relativi alle entità

Nel frontend, la generazione dei messaggi d'errore per le diverse entità del sistema, come utenti e organizzazioni, avviene attraverso metodi specifici per ciascuna di esse. Ogni entità ha infatti caratteristiche e dettagli di errore peculiari, che richiedono un'elaborazione dedicata per costruire un messaggio chiaro e informativo per l'utente.

Ogni metodo di gestione degli errori riceve tre parametri principali: il tipo di errore associato all'entità coinvolta, i dettagli forniti dal backend e la funzione di traduzione. I dettagli dell'errore contengono informazioni strutturate, come l'identificativo dell'entità o la causa specifica del problema. Sulla base di questi dati, il metodo elabora dinamicamente il messaggio da restituire.

A seconda del tipo di errore, il messaggio viene costruito in modi diversi. Ad esempio, per un errore di creazione, la funzione di gestione verifica se la causa è una violazione di unicità o un problema di integrità referenziale e genera un messaggio che spiega all'utente il motivo del fallimento. In caso di un'entità non trovata, il messaggio include l'identificativo cercato, offrendo un'indicazione chiara sulla natura del problema. Se invece l'errore è legato a una modifica o eliminazione

non riuscita, il messaggio può evidenziare l'operazione specifica che non è stata eseguita con successo.

Questa suddivisione in metodi specifici per ogni entità permette di mantenere il codice organizzato e facilmente estendibile. Nuovi tipi di errore possono essere aggiunti senza modificare la logica generale, assicurando che il sistema di gestione delle anomalie rimanga flessibile e scalabile.

Gestione degli errori di validazione

Come descritto nella Sezione 5.2.1, gli errori di validazione degli input vengono strutturati in modo da fornire tutte le informazioni necessarie per costruire un messaggio dettagliato. Nel frontend, il metodo `getValidationErrorMessage()` di seguito riportato, si occupa di elaborare questi dati per generare una stringa comprensibile e localizzata da mostrare all'utente.

Listing 5.8: Metodo `getValidationErrorMessage()`

```
const getValidationErrorMessage = (
  entity: ErrorEntity,
  operation: ErrorOperation,
  inputErrors: InputError[],
  translator: Translator
): string => {
  const errorDetails = inputErrors
    .map(({ field, issues }) => {
      const issuesMessages = issues
        .map((issue) =>
          translator(`issue.${issue.errorType}`, {
            field,
            expected: String(issue.expected),
            received: String(issue.received),
            min: issue.min,
            max: issue.max,
            keys: issue.keys ? issue.keys.join(", ") : undefined,
          })
        )
        .join(", ");
      return `${field}: ${issuesMessages}`;
    })
    .join(" | ");

  return translator("input_error", { entity, operation, errorDetails });
};
```

Il metodo analizza gli errori ricevuti, associando ogni campo con i problemi segnalati dal backend. Per ciascuna anomalia, viene costruito un messaggio traducendo dinamicamente il codice d'errore di Zod (ad esempio, "`too_small`" o "`invalid_string`") e integrando informazioni come il valore atteso, quello ricevuto e altri dettagli rilevanti. Infine, i vari messaggi vengono concatenati in un'unica stringa strutturata, così da fornire un'indicazione chiara sui problemi riscontrati e facilitare la correzione dei dati inseriti.

Considerazioni finali

Il sistema di gestione degli errori nel frontend è stato progettato per garantire scalabilità, modularità e supporto alla multilingua. Una delle sue caratteristiche principali è la separazione tra la logica di interpretazione degli errori e la libreria di internazionalizzazione utilizzata, resa possibile dall'uso del tipo astratto `Translator`. Questo approccio consente di adattare facilmente il sistema a diverse soluzioni di traduzione, senza influire sulla logica di elaborazione degli errori.

Oltre a questa astrazione, il frontend gestisce gli errori attraverso metodi dedicati che costruiscono dinamicamente i messaggi sulla base del tipo di errore ricevuto. Gli errori di validazione vengono elaborati per restituire un feedback dettagliato sui campi non conformi, mentre quelli relativi alle entità, come utenti e organizzazioni, vengono interpretati in base alla loro specificità, includendo informazioni rilevanti per l'utente finale.

Grazie a questa architettura, il frontend è in grado di convertire gli errori restituiti dal backend in messaggi informativi chiari e contestualizzati, migliorando l'esperienza utente e garantendo un'interazione fluida con il sistema. La modularità del sistema facilita inoltre l'estensione della gestione degli errori, permettendo di integrare nuove casistiche senza modificare la logica esistente.

Capitolo 6

Valutazione

L'obiettivo del presente capitolo è discutere i risultati finora ottenuti, in particolare sulla base dei requisiti già soddisfatti e delle migliorie introdotte dal nuovo sistema rispetto alla versione legacy. Poiché il sistema è ancora in fase di sviluppo, non è possibile fornire una valutazione complessiva, effettuare un confronto esaustivo con il vecchio sistema, né presentare risultati di test prestazionali e di usabilità. Tuttavia, è possibile analizzare le funzionalità già progettate e implementate, valutandone il contributo rispetto agli obiettivi iniziali e identificando le garanzie che il nuovo sistema è già in grado di fornire.

6.1 Requisiti soddisfatti

Sin dall'inizio era stato stabilito che questa tesi non avrebbe condotto alla realizzazione completa del sistema, ma si sarebbe concentrata principalmente sulle fondamentali fasi di analisi e progettazione, includendo anche l'implementazione di una prima versione funzionale. Nonostante ciò, è possibile tracciare un bilancio dei requisiti già soddisfatti, i quali evidenziano le potenzialità del nuovo sistema rispetto al precedente. A tal fine, i requisiti verranno suddivisi in due categorie: quelli già soddisfatti sia dal punto di vista progettuale che implementativo e quelli attualmente soddisfatti solo dal punto di vista progettuale.

6.1.1 Elementi progettati e implementati

Tra i requisiti soddisfatti, alcuni non si limitano alla fase di progettazione ma sono già stati concretamente implementati, costituendo il nucleo della prima versione del sistema. Questi elementi forniscono un primo miglioramento rispetto al legacy e rappresentano una base solida per l'evoluzione futura del progetto. Di seguito vengono analizzate le funzionalità già operative, evidenziandone l'impatto e le garanzie offerte.

Dashboard panoramica per MSP e Dealer

La prima funzionalità significativa già implementata è la dashboard panoramica, progettata specificamente per MSP e Dealer. Nel sistema legacy, questi utenti non disponevano di una visione aggregata ed efficace dei clienti sotto la loro gestione. Il nuovo sistema, invece, include una schermata con statistiche e grafici che permettono di ottenere una panoramica dettagliata delle attività del filtro DNS. In particolare, la dashboard mostra le seguenti informazioni:

- Nella parte superiore, vengono presentate statistiche di base relative alle richieste elaborate dal filtro. Tra queste: il numero totale di richieste ricevute, il numero di minacce bloccate, il numero di categorie e indirizzi IP bloccati, il numero di categorie consentite, il numero di richieste DNS non risolte (NXDOMAIN) e, infine, il numero di richieste che hanno forzato l'utilizzo della SafeSearch sui motori di ricerca e YouTube. Questi dati offrono un quadro generale sul funzionamento del sistema e sull'efficacia del filtraggio.
- Nella parte centrale, sono presenti grafici per monitorare l'andamento delle richieste DNS. In particolare, vi è:
  - Un grafico a barre che mostra le cinque categorie più bloccate;
  - Un grafico a torta che rappresenta la distribuzione percentuale delle richieste DNS in base alla loro tipologia (richieste bloccate, richieste consentite, blocchi per IP, domini non risolti).
- Nella parte inferiore, è presente un grafico che mostra l'andamento delle richieste DNS nel tempo, distinguendo tra quelle bloccate e quelle consentite.

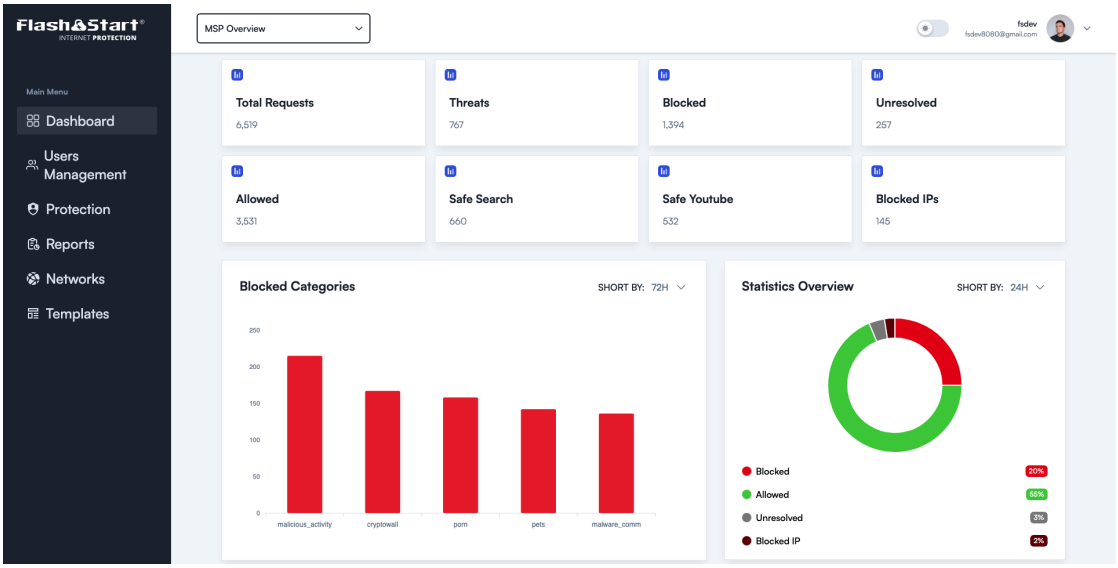

Figura 6.1: Schermata del nuovo sistema che rappresenta la dashboard MSP.

- Tutti e tre i grafici consentono di modificare l'intervallo temporale di visualizzazione, permettendo di analizzare i dati su finestre temporali di 24, 48 o 72 ore.

Oltre a offrire una panoramica complessiva, la dashboard consente di visualizzare statistiche e report specifici per ciascun cliente gestito dall'MSP o dal Dealer. È infatti possibile selezionare un cliente dall'apposito menu a tendina e ottenere un quadro dettagliato delle sue attività. Questa funzionalità risulta particolarmente utile per monitorare eventuali anomalie o problemi specifici di un singolo cliente.

Al momento, la dashboard è ancora in una fase iniziale e non è possibile esprimere garanzie sulla qualità dell'aggregazione dei dati, che verrà affinata con le iterazioni successive dello sviluppo.

Struttura e layout della dashboard Dal punto di vista dell'interfaccia utente, come si può vedere in Figura 6.1, la dashboard è stata progettata con una struttura chiara e organizzata per garantire un'esperienza di navigazione fluida. Il menu di navigazione, posizionato nella parte sinistra dello schermo, è sempre visibile indipendentemente dalla pagina in cui ci si trova. Nei dispositivi aventi un display

ridotto, come gli smartphone, il menu rimane nascosto per ottimizzare lo spazio e può essere aperto su richiesta dall'utente.

Nella parte superiore dell'interfaccia sono presenti diversi elementi chiave:

- **Il logo dell'azienda**, posizionato a sinistra.
- **Il selettore del cliente**, che consente agli MSP di filtrare i report per un cliente specifico, oppure di lasciare l'opzione predefinita per visualizzare la panoramica di tutti i clienti gestiti.
- **Il selettore del tema**, situato sulla destra, che permette di passare dalla modalità scura a quella chiara.
- **L'area utente autenticato**, da cui è possibile accedere alle impostazioni del profilo e alla funzione di logout.

In futuro, nell'header verrà integrata una sezione notifiche, accessibile tramite un'icona che indicherà la presenza di nuovi avvisi e permetterà di visualizzarli all'interno di un pannello dedicato.

Infine, il corpo centrale della dashboard ospita i grafici e le statistiche descritte in precedenza, fornendo un'analisi dettagliata del traffico DNS e della sicurezza del sistema. L'interfaccia è stata progettata per garantire flessibilità, con un layout adattivo che ottimizza la visualizzazione su schermi di diverse dimensioni.

Gestione della multiutenza

La necessità di introdurre un sistema che supportasse la multiutenza ha costituito uno dei principali requisiti per il nuovo pannello. Rispetto al precedente, in cui la gestione degli account risultava assai limitata, la nuova implementazione permette ora di associare più credenziali di accesso a una singola organizzazione e di abilitare operazioni di creazione, modifica ed eliminazione degli utenti. Questa flessibilità si traduce in un notevole progresso rispetto al passato, consentendo a più persone, con diversi ruoli, di accedere alla dashboard e monitorare le informazioni di loro competenza.

Un ulteriore aspetto riguarda la futura integrazione del controllo degli accessi: sebbene tale meccanismo non sia ancora stato sviluppato, l'architettura del

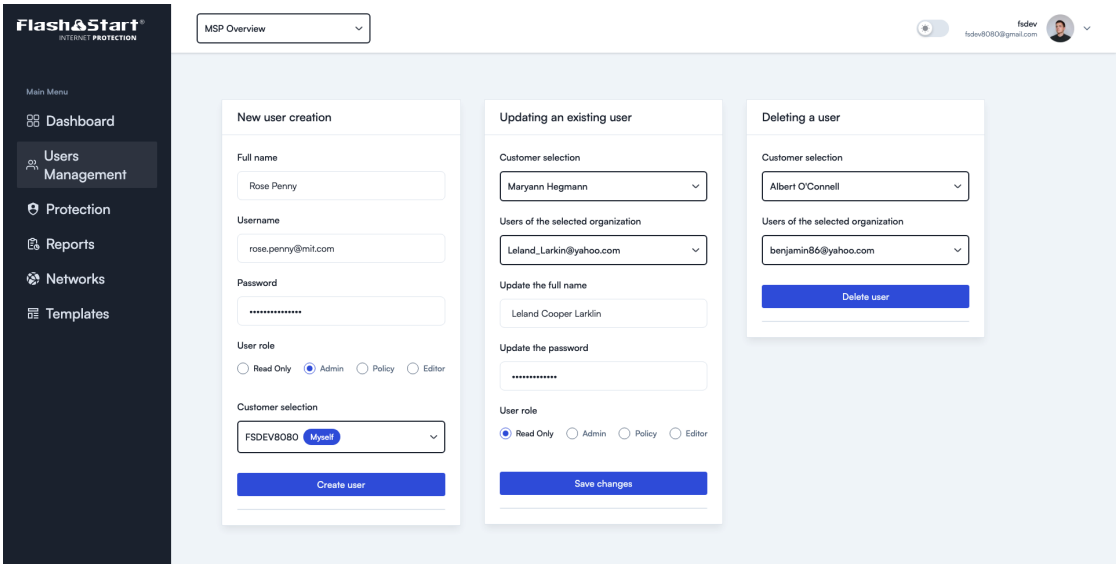

Figura 6.2: Schermata del nuovo sistema per la gestione degli utenti.

sistema è stata progettata per accogliere in modo agevole e graduale soluzioni di autorizzazione avanzate, basate su ruoli e permessi granulari. In questo modo, sarà possibile estendere ulteriormente la sicurezza e la modularità del pannello.

Sul fronte della protezione delle credenziali di accesso, è stato adottato l'algoritmo di hashing `bcrypt`. Quest'ultimo offre non solo una maggiore sicurezza contro attacchi di forza bruta, ma consente anche di aumentare il costo computazionale dell'hashing nel tempo, adeguandone la resistenza alla continua crescita della potenza di calcolo. In questo modo, il sistema può mantenere un elevato livello di protezione delle password anche sul lungo termine.

Per consentire una gestione semplice e immediata della multiutenza, è stata sviluppata un'apposita interfaccia che permette di inserire, aggiornare o rimuovere gli account, oltre che di assegnare a ogni utente un ruolo specifico all'interno dell'organizzazione. Tale funzionalità costituisce le fondamenta su cui verrà costruito in futuro un sistema di controllo degli accessi più sofisticato, in grado di differenziare i privilegi sulle singole risorse applicative.

Nella Figura 6.2 viene mostrata la schermata dedicata alla gestione degli utenti, in cui è possibile osservare le form per l'inserimento, la modifica e l'eliminazione degli account.

Supporto multilingua

Un ulteriore requisito già soddisfatto è il supporto multilingua. Il frontend del nuovo sistema, infatti, possiede già il supporto completo alle lingue italiano e inglese. Sebbene possa sembrare un dettaglio secondario, questa funzionalità è fondamentale per un prodotto destinato a un mercato internazionale, in cui gli utenti parlano lingue diverse. Inoltre, l'architettura progettata consente di aggiungere nuove lingue in modo semplice, senza necessità di modifiche al codice sorgente, ma solo integrando nuovi file di traduzione.

Gestione avanzata degli errori

Un altro elemento progettato e implementato è la gestione avanzata degli errori, che introduce una strategia scalabile e strutturata per il trattamento delle anomalie sia lato backend che frontend. Questa architettura garantisce diversi vantaggi:

- **Robustezza e controllo a tutti i livelli del sistema**: la gestione degli errori è strutturata in modo gerarchico, coprendo dalla validazione degli input nelle API fino ai problemi a livello di database, evitando che errori non gestiti possano propagarsi in modo incontrollato.
- **Standardizzazione e interoperabilità**: gli errori vengono generati in modo schematizzato e dinamico, garantendo una rappresentazione uniforme e facilmente interpretabile da qualsiasi componente del sistema, sia interno che esterno.
- **Compatibilità con il sistema multilingua**: grazie alla struttura modulare degli errori, il frontend può costruire dinamicamente messaggi testuali in più lingue, senza dipendere da stringhe fisse predefinite nel backend.
- **Affidabilità del codice grazie a TypeScript**: il sistema di gestione degli errori sfrutta la tipizzazione statica di TypeScript, garantendo una maggiore sicurezza nella gestione delle eccezioni e facilitando l'individuazione di problemi già in fase di sviluppo.

Rispetto al sistema legacy, che gestiva gli errori in modo frammentato e poco strutturato, questa soluzione garantisce una maggiore affidabilità complessiva, ri-

ducendo il rischio di comportamenti inattesi e migliorando la manutenibilità del codice. Inoltre, la sua scalabilità consente di adattarlo facilmente a nuove esigenze, rappresentando un elemento chiave per la futura evoluzione del sistema.

6.1.2 Elementi solo progettati

In aggiunta alle funzionalità già realizzate, si evidenzia l'esistenza di alcuni aspetti che, sebbene non ancora sviluppati in maniera operativa, risultano già completamente definiti dal punto di vista progettuale. Nonostante la loro efficacia pratica non sia al momento valutabile, è comunque possibile analizzare il potenziale e il valore aggiunto che tali elementi apporteranno rispetto al sistema legacy.

Gestione dei profili condivisi

Uno degli elementi più significativi in questa categoria è l'introduzione dei profili condivisi. Questa novità consente di creare profili di protezione condivisibili tra più organizzazioni all'interno della stessa gerarchia. Essi possono essere visti come template di protezione, utili soprattutto per gli MSP e per i Dealer, che potranno così offrire configurazioni standardizzate e omogenee ai loro clienti.

Un altro vantaggio di questa funzionalità è la semplificazione della gestione della protezione: eventuali modifiche a un profilo condiviso vengono automaticamente propagate a tutte le organizzazioni che lo utilizzano. Questo rappresenta un miglioramento significativo rispetto al vecchio sistema, dove le configurazioni erano gestite separatamente per ogni organizzazione, con difficoltà nella loro manutenzione e uniformità.

L'introduzione dei profili condivisi permette inoltre al sistema di allinearsi con i competitor, alcuni dei quali già dispongono di questa funzionalità. Benché tale caratteristica non sia ancora implementata, la sua progettazione dettagliata consente di avere una chiara roadmap per la sua realizzazione e integrazione futura.

Riprogettazione del database

Il design della funzionalità appena descritta rientra in un più ampio processo di rimodellazione del database, che garantisce maggiore scalabilità e coerenza dei dati.

Inoltre, la nuova struttura permette l'implementazione di funzionalità avanzate che il vecchio database, per sua natura, non poteva supportare. Questa revisione assicura che il sistema possa evolversi senza le limitazioni strutturali della versione legacy, rendendolo più flessibile, scalabile e adatto alle esigenze future.

Per quanto la riprogettazione del database sia stata completata principalmente a livello di schema concettuale, alcune componenti essenziali risultano già integrate nel sistema. In particolare, le nuove tabelle dedicate alla gestione degli utenti e delle organizzazioni sono già operative e costituiscono la base su cui si fonda il meccanismo di multiutenza. Queste modifiche hanno consentito di superare i vincoli del vecchio modello, che non era stato concepito per supportare una gestione avanzata degli accessi e delle gerarchie organizzative.

L'adozione del nuovo schema dati non solo migliora la struttura e la leggibilità del database, ma assicura anche una maggiore coerenza e manutenibilità, agevolando l'implementazione futura di altre funzionalità chiave. Inoltre, grazie al suo design modulare e scalabile, il database può adattarsi con facilità a nuove esigenze, garantendo flessibilità operativa e una gestione più efficiente dei dati.

6.2 Miglioramenti pianificati

Malgrado il nuovo sistema abbia già introdotto numerosi miglioramenti rispetto alla versione legacy, vi sono ancora diverse aree che non risultano complete, così come altre funzionalità che devono essere introdotte ex novo per garantire un'esperienza conforme alle esigenze operative previste. Tutti i punti trattati in questa sezione rappresentano miglioramenti già pianificati o previsti nella roadmap di sviluppo e verranno implementati nelle fasi successive rispetto a questa tesi.

6.2.1 Gestione avanzata dei permessi e autenticazione

L'attuale implementazione della multiutenza consente la creazione e la gestione di account con differenti ruoli, tuttavia, il controllo degli accessi richiede ulteriori sviluppi. In particolare, è prevista l'integrazione di un sistema *Role-Based Access Control* per la gestione dei ruoli utente, combinato con un livello di autorizzazione che regoli i permessi legati all'organizzazione di appartenenza e alle licenze attive.

Per garantire un sistema di permessi strutturato ed efficace, sarà necessaria un'analisi approfondita per individuare tutte le operazioni da regolamentare e codificare in una tabella del database. Successivamente, il sistema dovrà essere integrato a tutti i livelli della piattaforma, assicurando un controllo uniforme sugli endpoint dell'API e sulle rotte del frontend.

In aggiunta, per migliorare ulteriormente la sicurezza e conformarsi agli standard più recenti, è prevista l'implementazione di un sistema di autenticazione multifattore (MFA). Questo elemento è particolarmente richiesto dai clienti operanti in settori sensibili, dove la protezione degli accessi rappresenta un requisito imprescindibile.

6.2.2 Integrazione della brand identity nel frontend

Attualmente, il frontend del sistema è stato sviluppato con un focus sulle funzionalità, senza un'integrazione effettiva della nuova identità visiva dell'azienda. Nelle fasi successive, è già previsto il coinvolgimento del designer responsabile del rinnovamento dell'immagine aziendale, garantendo così un'interfaccia armonizzata con il design system dell'organizzazione. Sebbene questo aspetto non incida direttamente sulle funzionalità, riveste un ruolo cruciale per l'accettazione del prodotto da parte degli utenti finali, che devono percepirlo come un sistema moderno, affidabile e in linea con l'identità del brand.

6.2.3 Implementazione della nuova struttura del database

Come è già stato evidenziato, la riorganizzazione del database risulta già delineata sotto il profilo concettuale e, attualmente, è stata resa operativa soltanto la componente relativa alla gestione della multiutenza. Per le restanti sezioni, invece, è in corso la traduzione in un database di test.

Un elemento cruciale nella roadmap è la definizione di un processo di migrazione dei dati dal sistema legacy alla nuova base dati. Tale passaggio consentirà di validare fin da subito la coerenza e l'integrità delle nuove tabelle, garantendo al contempo la prosecuzione dello sviluppo sulla versione migliorata del database.

6.2.4 Aumento della copertura dei test

Attualmente, l'infrastruttura di testing del frontend è stata predisposta e resa operativa, ma non dispone ancora di specifiche definite per garantire una copertura adeguata. Tuttavia, la *coverage* al 100% del codice sorgente rappresenta un requisito fondamentale per garantire la stabilità e l'affidabilità del sistema, in quanto contribuisce a individuare e correggere eventuali anomalie prima del rilascio in produzione. Per soddisfare questo requisito è necessario incrementare la copertura dei test automatizzati, con particolare attenzione ai componenti critici dell'interfaccia utente. Questo aspetto è già stato pianificato e verrà affrontato nelle prossime iterazioni dello sviluppo, al fine di consolidare la qualità del software e ridurre il rischio di regressioni.

6.2.5 Implementazione delle funzionalità chiave

Oltre ai miglioramenti tecnici e architetturali già discussi, il completamento della prima release del sistema richiede l'integrazione di alcune macrofunzionalità fondamentali per garantirne la piena operatività. In particolare, il sistema dovrà includere:

- **Gestione della protezione**: interfacce e strumenti dedicati alla configurazione delle policy di filtraggio e sicurezza, con un'efficace integrazione dei profili condivisi.

- **Generazione e visualizzazione avanzata dei report**: estensione del modulo di reporting per offrire un'analisi più dettagliata rispetto ai dati attualmente disponibili nella dashboard panoramica, così come la possibilità di esportare i report in formati standard.

- **Analisi del traffico in tempo reale**: strumenti per il monitoraggio live delle richieste DNS, utili per una gestione più dinamica della sicurezza di rete. Dovrà essere inclusa una funzione di ricerca avanzata per filtrare e analizzare i dati in tempo reale.

- **Gestione degli Endpoint remoti**: integrazione del sistema con i dispositivi mobili dotati di *ClientShield*, per consentirne la configurazione e il monitoraggio da parte dei gestori del filtro.

Il nuovo sistema è destinato a sostituire integralmente il precedente, ragion per cui dovrà integrare fin dalla prima release tutte le macrofunzionalità della versione legacy, garantendo continuità nell'erogazione dei servizi e nella gestione della protezione e del traffico DNS. L'implementazione di tali moduli, dunque, non costituisce un semplice avanzamento, bensì un requisito imprescindibile per assicurare una transizione senza compromessi tra il vecchio e il nuovo sistema.

6.2.6 Scrittura della documentazione

Un aspetto fondamentale per garantire la manutenibilità e l'adozione efficace del nuovo sistema è la creazione di una documentazione completa e strutturata. Questo processo coinvolge tre aree principali: la documentazione del codice, la formalizzazione delle API e la stesura di un manuale utente.

Per quanto riguarda il codice sorgente, è essenziale integrare commenti dettagliati nelle parti più rilevanti e articolate, al fine di agevolare la comprensione della codebase da parte di nuovi sviluppatori. La documentazione interna sarà redatta esclusivamente in inglese, seguendo le best practices per i progetti software con una possibile espansione internazionale.

Parallelamente, sarà necessario predisporre una documentazione formale delle API, che descriva in modo chiaro gli endpoint disponibili, i parametri accettati e i formati di risposta previsti. Questa documentazione sarà tradotta nelle principali lingue per garantire un facile accesso agli sviluppatori esterni e ai partner tecnologici.

Infine, la redazione di un manuale utente sarà essenziale per facilitare l'adozione del nuovo sistema sia da parte dei nuovi utenti sia di coloro che effettuano la transizione dal pannello legacy. Il manuale dovrà includere istruzioni dettagliate sulle funzionalità disponibili e sarà tradotto nelle principali lingue, garantendo un supporto chiaro e accessibile a un'utenza internazionale.

L'implementazione di un processo di documentazione strutturato contribuirà significativamente a migliorare l'esperienza degli sviluppatori e degli utenti, garan-

tendo una maggiore efficienza nella gestione del sistema e una più rapida curva di apprendimento.

Capitolo 7

Conclusione

Il presente lavoro di tesi ha avuto come obiettivo la reingegnerizzazione del pannello di configurazione di un sistema legacy per il filtraggio DNS, progettando e implementando un'infrastruttura moderna e scalabile in grado di superare le limitazioni della versione precedente. Il progetto si è focalizzato sull'analisi e la riprogettazione dell'architettura, gettando le basi per una transizione completa verso un sistema più efficiente e flessibile. Questo capitolo riepiloga i principali risultati ottenuti, le sfide affrontate e gli sviluppi futuri previsti per il completamento e l'evoluzione del sistema.

7.1 Sintesi del lavoro svolto e sfide affrontate

Lo sviluppo di un sistema software segue tipicamente un processo articolato in più fasi, tra cui la raccolta dei requisiti, l'analisi e progettazione, l'implementazione e il testing. La presente tesi è intervenuta dopo la fase di raccolta dei requisiti, concentrandosi sull'analisi e la progettazione di un sottoinsieme di caratteristiche chiave della nuova piattaforma, nonché sulla loro prima implementazione. L'obiettivo principale è stato quello di costruire una base solida su cui fondare l'evoluzione del sistema, risolvendo le principali criticità della versione legacy e introducendo soluzioni volte a migliorare scalabilità, sicurezza e manutenibilità. In particolare, le milestone principali raggiunte includono:

- **Multiutenza e gestione degli accessi**: una delle caratteristiche chiave del nuovo sistema rispetto al legacy è il pieno supporto al modello multi-tenant, che permette la gestione di più organizzazioni e utenti all'interno di un'unica istanza. Attualmente, il sistema di autenticazione è stato implementato con successo e consente un accesso sicuro alla piattaforma. Tuttavia, il livello di autorizzazione, essenziale per regolare con precisione i permessi di ciascun utente in base al ruolo e all'organizzazione di appartenenza, è ancora in fase di definizione. La sua integrazione futura consentirà di sfruttare appieno il potenziale della multiutenza, offrendo una gestione più strutturata degli accessi e migliorando la sicurezza del sistema.

- **Dashboard panoramica per MSP**: l'introduzione di una dashboard dedicata ai Managed Service Provider ha reso possibile una visione aggregata delle attività dei clienti, migliorando il controllo e la gestione della protezione DNS. Questo strumento fornisce agli MSP dati centralizzati e statistiche dettagliate, semplificando il monitoraggio delle configurazioni e delle minacce bloccate. Oltre a migliorare l'esperienza degli amministratori, questa funzionalità posiziona strategicamente il prodotto in un mercato più orientato al settore enterprise, che rappresenta uno degli obiettivi primari dell'azienda in termini di crescita e competitività.

- **Nuova architettura del database**: la rimodellazione della base dati ha consentito di superare le rigidità della vecchia struttura, garantendo maggiore coerenza e supporto nativo alle nuove funzionalità. Oltre a risolvere le limitazioni del database legacy, la nuova architettura è stata progettata seguendo le migliori pratiche nel disegno concettuale delle strutture dati. Sono stati adottati nomi significativi per le entità, così come tipi di dato più coerenti e precisi, il tutto per evitare ambiguità e migliorare l'integrità del database. Inoltre, la suddivisione delle tabelle è stata ottimizzata per garantire un'organizzazione più chiara e una gestione più efficiente delle relazioni tra i dati, ponendo le basi per un sistema scalabile e facilmente estendibile.

- **Gestione avanzata e strutturata degli errori**: è stato sviluppato un sistema innovativo per la gestione degli errori, progettato per essere scalabi-

le, facilmente integrabile tra i microservizi e compatibile con la multilingua. Questo sistema è pervasivo, operando a tutti i livelli del backend, dalla validazione degli input nelle API alla gestione delle eccezioni interne e degli errori a livello di database. La sua implementazione sfrutta la tipizzazione avanzata del linguaggio di programmazione utilizzato, garantendo un controllo rigoroso e una maggiore robustezza nel trattamento degli errori. Inoltre, l'approccio scelto lo rende poco invasivo dal punto di vista implementativo, permettendo agli sviluppatori di utilizzarlo in modo efficace senza introdurre complessità eccessive nel codice esistente. Questo aspetto ha rappresentato uno dei contributi più significativi apportati al progetto, migliorando la manutenibilità e l'affidabilità dell'intero sistema.

Sfide affrontate

Durante lo sviluppo del sistema sono emerse diverse criticità, che hanno richiesto un'analisi approfondita e un approccio metodologico mirato. Tra le principali sfide affrontate vi sono:

- **Apprendimento di nuove tecnologie**: la realizzazione del nuovo sistema ha richiesto l'utilizzo di strumenti e paradigmi non presenti nella versione legacy, rendendo necessaria un'iniziale fase di formazione e consolidamento delle conoscenze. In particolare, l'ambiente di sviluppo è stato completamente rinnovato, introducendo linguaggi di programmazione diversi, framework moderni e un'architettura completamente differente rispetto al passato. Questo ha comportato non solo l'acquisizione di nuove competenze tecniche, ma anche la necessità di rivedere le metodologie di lavoro per adattarsi a un paradigma di sviluppo più strutturato e modulare.

- **Strutturazione avanzata degli errori**: una delle principali difficoltà in questo contesto è stata la definizione di un meccanismo di gestione degli errori che fosse al contempo semplice ed efficace. È stato necessario individuare una quantità esaustiva di potenziali errori da codificare, garantendo una categorizzazione chiara e uniforme per facilitarne la gestione. Inoltre, la strutturazione del sistema di error handling ha dovuto assicurare un'intero-

perabilità coerente tra backend e frontend, evitando ambiguità nella comunicazione. Un ulteriore vincolo è stato quello di progettare un meccanismo che permettesse di costruire i messaggi lato frontend in modo semplice e intuitivo, facilitando al contempo la traduzione automatizzata e la gestione multilingua dell'applicazione.

7.2 Evoluzioni e prospettive future

Il lavoro svolto finora rappresenta solo il primo passo verso la realizzazione di un sistema completo e pienamente operativo. Sebbene siano già pianificati sviluppi successivi per il perfezionamento delle funzionalità esistenti, vi sono ulteriori direzioni di evoluzione che potrebbero essere esplorate in futuro per potenziare ulteriormente il sistema. Tra le possibili estensioni e miglioramenti si evidenziano:

- **Integrazione con sistemi di gestione delle identità e directory aziendali**: un'importante evoluzione potrebbe riguardare l'integrazione con servizi come Active Directory o altri Identity Provider aziendali. Questa estensione permetterebbe di applicare il filtraggio DNS in maniera ancora più granulare, assegnando policy di navigazione non solo per rete, ma anche per singolo utente o gruppo definiti all'interno dell'organizzazione. In questo modo, le configurazioni potrebbero essere centralizzate e automatizzate, migliorando la gestione della sicurezza e la conformità aziendale.

- **Implementazione di un sistema di autenticazione centralizzato**: attualmente, l'autenticazione degli utenti è gestita in maniera autonoma dal sistema ma un possibile sviluppo futuro potrebbe prevedere la realizzazione di un servizio dedicato alla gestione centralizzata dell'autenticazione e dei permessi. Questo sistema potrebbe fungere da Single Sign-On (SSO) per tutti i servizi aziendali, permettendo agli utenti di autenticarsi una sola volta per accedere a tutte le applicazioni interne. Oltre a migliorare l'esperienza utente, un simile approccio garantirebbe una gestione più sicura e uniforme delle credenziali e dei livelli di accesso, riducendo i rischi legati alla duplicazione e alla dispersione delle informazioni di autenticazione.

- **Allineamento tra specifiche API e implementazione**: attualmente, la documentazione delle API e l'implementazione delle rotte backend devono essere mantenute manualmente, con il rischio di disallineamenti nel tempo. Un'evoluzione strategica potrebbe essere l'adozione di strumenti come Swagger[1], che consentono di generare automaticamente la documentazione a partire dal codice sorgente, garantendo così una perfetta corrispondenza tra specifiche e implementazione reale. Questo approccio migliorerebbe la manutenibilità del sistema, riducendo gli errori di integrazione con i client esterni e agevolando il lavoro degli sviluppatori nel comprendere e utilizzare le API disponibili.

L'insieme di questi sviluppi contribuirebbe a rendere il nuovo sistema una piattaforma sempre più completa, performante e adattabile alle esigenze operative degli utenti. Oltre a migliorare l'efficienza e la sicurezza, queste integrazioni favorirebbero una maggiore interoperabilità con altri servizi e infrastrutture aziendali, rendendo il sistema più scalabile e competitivo rispetto alle soluzioni attualmente disponibili sul mercato. Nonostante queste evoluzioni non siano ancora state pianificate nei dettagli, rappresentano delle opportunità strategiche che potrebbero essere prese in considerazione per il futuro, in funzione delle esigenze emergenti e delle prospettive di crescita del progetto.

[1] `https://swagger.io`

Bibliografia

[AKA+16] Kamal Alieyan, Mohammed M. Kadhum, Mohammed Anbar, Shafiq Ul Rehman, and Naser K. A. Alajmi. An overview of ddos attacks based on DNS. In *International Conference on Information and Communication Technology Convergence, ICTC 2016, Jeju Island, South Korea, October 19-21, 2016*, pages 276–280. IEEE, 2016.

[BCL21] Will Bonasera, Md Minhaz Chowdhury, and Shadman Latif. Denial of service: A growing underrated threat. In *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, page 1–6. IEEE, October 2021.

[CI90] Elliot J. Chikofsky and James H. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Softw.*, 7(1):13–17, 1990.

[FF14] Gil Fink and Ido Flatow. *Introducing Single Page Applications*, page 3–13. Apress, 2014.

[HH14] Adam Ali Zare Hudaib and EAZ Hudaib. Dns advanced attacks and analysis. *International Journal of Computer Science and Security (IJCSS)*, 8(2):63–74, 2014.

[III99] Donald E. Eastlake III. Domain name system security extensions. *RFC*, 2535:1–47, 1999.

[KR20] Tae Hyun Kim and Douglas Reeves. A survey of domain name system vulnerabilities and attacks. *Journal of Surveillance, Security and Safety*, 1(1), 2020.

[LMM+00] Antonio Lioy, Fabio Maino, Marius Marian, Daniele Mazzocchi, et al. Dns security. In *Terena Networking Conference*, 2000.

[MA08] Steven J. Murdoch and Ross Anderson. *Tools and Technology of Internet Filtering*, page 57–72. The MIT Press, January 2008.

[Mag24] Jonathan Magnusson. Survey and analysis of DNS filtering components. *CoRR*, abs/2401.03864, 2024.

[MJS+00] Hausi A. Müller, Jens H. Jahnke, Dennis B. Smith, Margaret-Anne D. Storey, Scott R. Tilley, and Kenny Wong. Reverse engineering: a roadmap. In Anthony Finkelstein, editor, *22nd International Conference on on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4-11, 2000*, pages 47–60. ACM, 2000.

[Moc87a] Paul V. Mockapetris. Domain names - concepts and facilities. *RFC*, 1034:1–55, 1987.

[Moc87b] Paul V. Mockapetris. Domain names - implementation and specification. *RFC*, 1035:1–55, 1987.

[MQO18] Manar Majthoub, Mahmoud H. Qutqut, and Yousra Odeh. Software re-engineering: An overview. In *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, page 266–270. IEEE, July 2018.

[RH96] Linda H Rosenberg and Lawrence E Hyatt. Software re-engineering. *Software Assurance Technology Center*, 1:2–3, 1996.

[Som11] I. Sommerville. *Software Engineering*. International Computer Science Series. Pearson, 2011.

[Var10] Vijay Varadharajan. Internet filtering issues and challenges. *IEEE Secur. Priv.*, 8(4):62–65, 2010.

Ringraziamenti

Giungere a scrivere questi ringraziamenti è un momento di grande emozione. Questo traguardo non rappresenta solo il conseguimento della laurea, ma la vittoria su difficoltà, insicurezze e momenti di crisi. Significa aver superato esami che sembravano insormontabili e aver portato a termine la stesura di questa tesi, che ho sempre temuto e rimandato. Ora posso dire di avercela fatta e di esserne orgoglioso!

Un ringraziamento speciale alla mia famiglia: grazie *mamma* e *babbo* per avermi sempre sostenuto e per aver reso più leggeri gli anni di studio. Grazie a mio fratello *Samuele* per il supporto e per quelle conversazioni che solo con te posso avere. Un pensiero speciale va alla mia amata *Lallina*, che mi è stata accanto con affetto e incoraggiamento nei momenti più difficili di questo percorso.

Un sentito grazie anche agli amici e ai compagni di corso che hanno contribuito a rendere questo viaggio più stimolante e ricco di momenti indimenticabili. In particolare, grazie *Edoardo* e *Giorgio* per il vostro supporto e la vostra amicizia, *Andrea*, *Gianmarco* e *Thomas* per essere sempre presenti fin dai tempi delle scuole medie. Infine, grazie ai compagni con cui ho affrontato la maggior parte degli esami: *Lorenzo*, *Alex*, *Andrea*, *Marco*, *Fabio* e *Michele*.

Desidero esprimere la mia profonda gratitudine al mio relatore, il professor *Mirko Viroli*, e ai correlatori, *Nicolas Farabegoli* (che da compagno di corso è passato a essere mio correlatore) e *Gianluca Aguzzi*, per il loro prezioso supporto e la loro competenza. Il loro contributo è stato essenziale non solo per il perfezionamento di questa tesi, ma anche per lo sviluppo del sistema che ne è oggetto.

Un ringraziamento speciale va, infine, all'azienda che mi ha ospitato per il tirocinio, in particolare al suo CEO, *Francesco*, e a tutti i colleghi. Senza di loro non avrei avuto l'opportunità di partecipare a un progetto di tale portata, né di crescere professionalmente come ho fatto in questi mesi.

Questo traguardo segna la fine di un lungo percorso e l'inizio di nuove sfide. Ora posso finalmente guardare al futuro con il titolo di *Ingegnere*!