ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea in Ingegneria e Scienze Informatiche

Sviluppo di un pannello Web a supporto di un filtro DNS

Tesi di laurea in:
Programmazione ad Oggetti

Relatore
Prof. Mirko Viroli

Candidato
Alessandro Valmori

Correlatore
Dott. Nicolas Farabegoli

II Sessione di Laurea
Anno Accademico 2024-2025

Sommario

Questa tesi descrive la progettazione, lo sviluppo e l'analisi architetturale di una applicazione web full-stack, realizzata in collaborazione con l'azienda FlashStart Group. Il progetto nasce dalla necessità di fornire una dashboard in sola lettura per la visualizzazione e l'analisi dei dati provenienti dal servizio di filtraggio DNS offerto ai clienti dell'azienda.

L'obiettivo del lavoro è duplice: da un lato, la realizzazione di una piattaforma software funzionale, sicura e manutenibile; dall'altro, l'analisi critica delle scelte architetturali e dei design pattern della programmazione orientata agli oggetti (OOP) che ne hanno guidato lo sviluppo.

La metodologia si basa su un'architettura a servizi containerizzata con Docker. Il backend è stato sviluppato in Java, adottando il paradigma di programmazione reattiva con Spring WebFlux per garantire scalabilità ed efficienza. Il frontend è un'applicazione single-page (SPA) costruita con React e TypeScript. La gestione dei dati è affidata a un database PostgreSQL, mentre la sicurezza è implementata tramite un sistema di autenticazione basato su token JWT con rotazione.

Il risultato è un'applicazione capace di interfacciarsi con gli endpoint esterni di FlashStart e di presentare i dati in modo intuitivo attraverso componenti grafici. La trattazione approfondisce l'applicazione pratica di design pattern fondamentali come il Factory Method, utilizzato per la creazione di filtri dinamici nel gateway, e analizza come i principi SOLID e le tecniche OOP siano stati il fondamento per la strutturazione dei componenti sia del backend che del frontend.

Questo lavoro rappresenta un'analisi di come i principi teorici dell'ingegneria del software e i pattern OOP trovino applicazione concreta per risolvere problemi industriali, evidenziando benefici e compromessi delle scelte implementative in un contesto aziendale reale.

Indice

Capitolo 1

Introduzione

1.1 Contesto Aziendale e Motivazione del Progetto

Il presente lavoro di tesi si inserisce in un contesto industriale specifico, frutto della collaborazione con FlashStart SRL, un'azienda italiana con sede a Cesena, specializzata nello sviluppo e nella fornitura di soluzioni di filtraggio dei contenuti e protezione da minacce informatiche basate su tecnologia DNS (Domain Name System). I servizi offerti da FlashStart si rivolgono a una clientela diversificata, che include aziende, istituzioni educative e pubbliche amministrazioni, fornendo loro strumenti per garantire una navigazione sicura e controllata.

Al momento dell'inizio del percorso di tirocinio, nel mese di giugno 2025, l'azienda si trovava in una fase di significativa evoluzione tecnologica e strategica. Era infatti in corso un processo di completa reingegnerizzazione della propria piattaforma di gestione, la dashboard utilizzata dai clienti per configurare e monitorare il servizio di filtraggio. Questo processo, unito a un'operazione di rebranding aziendale, mirava a modernizzare l'infrastruttura e l'esperienza utente, con un rilascio previsto per novembre 2025.

In questo scenario di transizione, è emersa una criticità tanto specifica quanto urgente. La piattaforma allora in uso, pur essendo efficace per la gestione delle policy di protezione, presentava una notevole lacuna funzionale: l'assenza di una modalità di consultazione dei dati in sola lettura (readonly). Gli utenti, in par-

ticolare gli amministratori di rete e i responsabili IT, manifestavano la crescente necessità di poter analizzare i report e le statistiche di navigazione senza avere i permessi di modifica, per evitare alterazioni accidentali delle configurazioni di sicurezza.

Il progetto di tesi nasce per rispondere a questa precisa esigenza. Data l'impossibilità di attendere il rilascio della nuova piattaforma, si è optato per lo sviluppo di una soluzione tattica e mirata: un'applicazione web temporanea, concepita come "ponte" (bridge) tecnologico. Lo scopo primario di questa applicazione è fornire ai clienti un pannello di controllo readonly per le funzionalità standard di analisi dei dati, garantendo continuità operativa e soddisfacendo le richieste del mercato fino alla migrazione sulla nuova infrastruttura. Questo lavoro di tesi documenta pertanto non solo la realizzazione di un prodotto software, ma anche l'approccio ingegneristico adottato per sviluppare una soluzione efficace e affidabile in un contesto agile e con vincoli temporali definiti.

1.2 Obiettivi della Tesi

A partire dal contesto delineato, questo elaborato si pone obiettivi che trascendono la semplice descrizione di un prodotto software, per configurarsi come un'analisi approfondita delle metodologie di ingegneria del software applicate a un caso di studio reale. Il fine principale della tesi è, pertanto, analizzare e documentare come i paradigmi della Programmazione Orientata agli Oggetti (OOP) e le relative pratiche di progettazione, quali i design pattern, vengano impiegati per realizzare un'applicazione web complessa in un contesto industriale. Si intende dimostrare come tali principi, spesso affrontati in ambito teorico, rappresentino strumenti pratici e fondamentali per guidare le scelte architetturali e per garantire qualità essenziali come la manutenibilità, la scalabilità e la robustezza del software, specialmente quando si opera con vincoli di tempo e risorse.

Per supportare tale analisi, un primo passo fondamentale sarà la documentazione completa e dettagliata dell'architettura full-stack dell'applicazione. Verrà illustrato il flusso di dati e le interazioni tra il frontend, sviluppato in React con TypeScript, e il backend, basato su un modello di programmazione reattiva in Java. Questo esame non si limiterà a una descrizione statica, ma includerà un'analisi

critica delle decisioni tecniche prese durante lo sviluppo, giustificando la scelta dello stack tecnologico e approfondendo l'implementazione di componenti chiave come il sistema di autenticazione sicuro basato su token JWT e l'integrazione con le API esterne di FlashStart.

Il cuore analitico della tesi si concentrerà sull'applicazione pratica dei Design Pattern. Verranno identificati e discussi esempi concreti di pattern implementati nel codice sorgente, spiegando per ciascuno il problema che risolve e i benefici apportati in termini di flessibilità e organizzazione del codice. In questo modo, si intende creare un collegamento esplicito tra i concetti teorici dell'ingegneria del software e la loro implementazione pratica. Particolare attenzione sarà data a come le scelte di design abbiano promosso un'architettura dinamica e facilmente manutenibile; un requisito fondamentale dato che l'applicazione doveva interfacciarsi con servizi interni di FlashStart a loro volta in piena fase di reingegnerizzazione. Verrà quindi evidenziato come le decisioni di design non siano state casuali, ma guidate da principi consolidati volti a massimizzare l'efficienza e la qualità, nel pieno rispetto della natura strategica ma temporanea del progetto.

1.3 Panoramica dello Stack Tecnologico

La realizzazione di un'applicazione robusta, scalabile e sicura, pur in un contesto di sviluppo agile e a breve termine, ha richiesto un'attenta selezione delle tecnologie costitutive. L'architettura del sistema è stata progettata per essere moderna e reattiva, garantendo un'esperienza utente fluida e un uso efficiente delle risorse server. Ogni componente dello stack, dal backend al frontend, fino all'infrastruttura di deployment, è stato scelto per rispondere a precise esigenze di progetto.

Il cuore pulsante dell'applicazione risiede nel backend, sviluppato in Java 17 e fondato sul framework Spring Boot. Per questo progetto è stato adottato un paradigma di programmazione completamente reattivo, utilizzando Spring WebFlux anziché il tradizionale Spring MVC. Questa scelta strategica si basa sulla natura dell'applicazione, intrinsecamente legata a operazioni di I/O (input/output) come le chiamate verso API esterne e l'accesso al database. Il modello non bloccante di WebFlux permette di gestire un elevato numero di connessioni concorrenti con un numero limitato di thread, ottimizzando le risorse e garantendo bassa latenza.

Inoltre, il backend assume un ruolo centrale di API Gateway grazie all'integrazione con Spring Cloud Gateway. Questa componente orchestra il traffico di rete, indirizzando le richieste del client verso le API interne dell'applicazione o fungendo da proxy sicuro verso i due distinti endpoint di FlashStart, applicando filtri personalizzati per l'autenticazione e l'arricchimento delle chiamate.

Per l'interfaccia utente è stata realizzata una Single Page Application (SPA) utilizzando React, una delle librerie JavaScript più diffuse per la creazione di interfacce complesse e performanti. L'adozione di TypeScript ha introdotto la tipizzazione statica nel codice frontend, un elemento cruciale per aumentare la robustezza, la manutenibilità e la chiarezza del codice, riducendo la probabilità di errori a runtime. La comunicazione asincrona con il backend avviene tramite chiamate HTTP gestite dalla libreria axios, configurata con intercettori specifici per aggiungere automaticamente i token di autenticazione alle richieste e per gestire in modo trasparente la logica di rinnovo dei token in caso di sessione scaduta, come definito in axiosInstance.ts.

La persistenza dei dati, limitata alle informazioni su utenti e token per la gestione delle sessioni, è affidata a PostgreSQL, un database relazionale open-source rinomato per la sua stabilità e le sue funzionalità avanzate. Per mantenere la coerenza con l'approccio reattivo del backend, l'accesso al database è gestito tramite Spring Data R2DBC (Reactive Relational Database Connectivity). Questo permette di estendere il paradigma non bloccante fino al livello di accesso ai dati, evitando colli di bottiglia e garantendo che l'intera catena di elaborazione di una richiesta, dalla ricezione alla risposta, rimanga asincrona.

La sicurezza dell'applicazione è un pilastro fondamentale dell'architettura. Il sistema di autenticazione è stateless e si basa su JSON Web Tokens (JWT), che vengono trasmessi dal client nell'header di ogni richiesta autorizzativa. Per mitigare i rischi legati a token di lunga durata, è stato implementato un meccanismo di sicurezza avanzato noto come rotazione dei refresh token. Come visibile nel AuthenticationService, ogni volta che un refresh token viene utilizzato con successo per ottenere un nuovo access token, esso viene immediatamente invalidato e sostituito da uno nuovo. Questa strategia riduce drasticamente la finestra temporale in cui un token compromesso potrebbe essere sfruttato, aumentando significativamente la sicurezza complessiva delle sessioni utente.

Infine, l'intera infrastruttura è gestita tramite un approccio DevOps moderno. L'applicazione è interamente containerizzata utilizzando Docker, e la sua architettura multi-servizio (backend, frontend web server e database) è definita tramite file docker-compose.yaml. Questo garantisce la portabilità e la riproducibilità dell'ambiente su macchine diverse. Il ciclo di vita dello sviluppo è automatizzato da una pipeline di Continuous Integration e Continuous Deployment (CI/CD) ibrida, che si avvale di GitHub Actions per l'orchestrazione dei flussi di lavoro e di un runner self-hosted, installato direttamente sul server di destinazione, per eseguire in modo sicuro le operazioni di build e deployment.

Capitolo 2

Background

Il presente capitolo si prefigge di delineare il contesto teorico e tecnologico che funge da fondamento per lo sviluppo del pannello web oggetto di questa tesi. Per giustificare le decisioni architetturali e tecnologiche adottate, verranno esplorate le aree di ricerca che hanno guidato le scelte progettuali e implementative. L'analisi spazierà dai fondamenti della Programmazione Orientata agli Oggetti (OOP) e dei design pattern, fino alle architetture reattive e alle metodologie DevOps, per concludere con uno sguardo allo stato dell'arte dei sistemi di filtraggio DNS in cui il progetto si inserisce.

2.1 Pattern Architetturali per Sistemi Reattivi

I design pattern sono soluzioni collaudate e riutilizzabili a problemi comuni di progettazione. Nello sviluppo di applicazioni web moderne, caratterizzate da elevata interattività e flussi di dati complessi, la loro applicazione è cruciale. Un pattern fondamentale è l'Observer, che definisce una dipendenza uno-a-molti tra oggetti, in modo che quando un "soggetto" cambia stato, i suoi "osservatori" vengono notificati automaticamente.

Tuttavia, l'Observer classico può mostrare limiti in contesti reattivi complessi, come la potenziale incapacità di aggiornare correttamente gli oggetti, l'esecuzione di aggiornamenti ridondanti o in ordine errato (causando "glitch") e la creazione di dipendenze circolari. Per superare queste criticità, sono emersi pattern più so-

fisticati come il pattern REACTOR [MGCS21]. Questo pattern utilizza un grafo di dipendenze per gestire le reazioni a catena, assicurando che gli aggiornamenti avvengano in modo ordinato e consistente tramite ordinamento topologico. È pensato per scenari in cui il cambiamento di stato di un oggetto richiede una reazione automatica in tutti gli oggetti dipendenti, situazione comune in interfacce grafiche, fogli di calcolo e animazioni. Sebbene la sua implementazione di riferimento sia in C#, i concetti sono perfettamente applicabili al contesto Java di questo progetto.

A livello di interfaccia utente, il pattern DCRC (Dynamically Coalescing Reactive Chains) [OMCJL24] affronta problemi simili, riducendo i "glitch" visivi nelle GUI web raggruppando catene di eventi per garantire aggiornamenti ordinati. La progressione da Observer a pattern come REACTOR (per il backend) e DCRC (per il frontend) evidenzia una maturazione nel modo in cui le applicazioni moderne gestiscono cambiamenti di stato complessi, un tema centrale per questo lavoro di tesi che integra OOP e programmazione reattiva.

2.2 Architetture Reattive e Programmazione Non-bloccante

Le architetture reattive sono diventate fondamentali per costruire applicazioni web capaci di gestire alta concorrenza e grandi volumi di dati mantenendo responsività. La programmazione reattiva è un paradigma focalizzato sui flussi di dati (data streams) e sulla propagazione del cambiamento [Wik24]; le applicazioni "reagiscono" ai dati man mano che diventano disponibili. I suoi benefici principali sono una migliore responsività (le operazioni non bloccanti evitano che l'applicazione si fermi in attesa di I/O), resilienza (gestione degli errori integrata nei flussi), elasticità e un uso efficiente delle risorse.

Spring WebFlux è il modulo di Spring che permette di creare applicazioni web reattive in Java. È costruito su Project Reactor, che fornisce i tipi `Mono<T>` (per 0 o 1 elemento) e `Flux<T>` (per 0 o N elementi). Le sue caratteristiche chiave sono l'I/O non bloccante, che permette di servire molte connessioni con pochi thread, il supporto alla "backpressure" per evitare sovraccarichi, e la possibilità di comporre logica asincrona in modo dichiarativo.

Un aspetto interessante è la distinzione tra Object-Oriented Reactive Programming (OORP), dove si usano oggetti per comporre flussi reattivi, e Reactive Object-Oriented Programming (ROOP), dove gli oggetti stessi sono flussi di stato reattivo [BPVdWDM13]. Spring WebFlux tende a favorire un approccio OORP, dove i servizi e i controller orchestrano e manipolano i flussi `Flux` e `Mono`.

2.3 Sicurezza nelle Applicazioni Web

La sicurezza è un aspetto non negoziabile, specialmente in un'applicazione che gestisce il monitoraggio di un filtro DNS. Il meccanismo di autenticazione scelto per questo progetto è basato su JSON Web Tokens (JWT), uno standard aperto per creare token di accesso che contengono "claim" (asserzioni) su un utente. Un JWT è composto da un Header, un Payload e una Signature, che ne garantisce l'integrità e l'autenticità.

Seguendo le best practice di sicurezza, l'architettura adotta token di accesso a breve scadenza (access token) e token di aggiornamento a lunga scadenza (refresh token). Una misura di sicurezza cruciale implementata è la rotazione dei refresh token: quando un refresh token viene usato, viene invalidato e sostituito con uno nuovo. Questa tecnica aiuta a rilevare eventuali furti e limita drasticamente i danni in caso di compromissione del token. Questo approccio, sebbene aumenti la sicurezza, introduce un leggero grado di "statefulness" in un sistema altrimenti stateless, poiché il server deve mantenere traccia dei token validi, un compromesso progettuale importante discusso in questa tesi.

2.4 Metodologie DevOps e Containerizzazione

Le pratiche DevOps e la containerizzazione sono standard moderni per aumentare l'efficienza e l'affidabilità del ciclo di vita del software. Per questo progetto è stato utilizzato Docker, una piattaforma che automatizza il deployment di applicazioni all'interno di container software. I container incapsulano l'applicazione e tutte le sue dipendenze, garantendo che funzioni in modo coerente in qualsiasi ambiente ed eliminando il classico problema del "funziona sulla mia macchina". I

benefici includono consistenza, isolamento, gestione semplificata delle dipendenze e scalabilità.

Il ciclo di vita del software è automatizzato tramite una pipeline di Continuous Integration/Continuous Deployment (CI/CD). Questo progetto sfrutta una soluzione ibrida con GitHub Actions per l'orchestrazione e runner self-hosted per l'esecuzione dei job di deployment. Questo approccio combina la comodità della definizione delle pipeline su GitHub con la sicurezza e la flessibilità di eseguire il deployment su macchine controllate direttamente dall'azienda.

2.5 Stato dell'Arte: Sistemi di Filtraggio DNS

Per comprendere appieno il contesto applicativo, è essenziale analizzare lo stato dell'arte del filtraggio DNS. Il Domain Name System (DNS) è il sistema che traduce i nomi di dominio (es. 'www.google.com') in indirizzi IP. Il filtraggio DNS sfrutta questo meccanismo per bloccare l'accesso a domini malevoli o indesiderati, intercettando le richieste e rispondendo con l'indirizzo di una pagina di blocco o facendo fallire la risoluzione. Le moderne soluzioni di filtraggio, come quelle di FlashStart, utilizzano blacklist, whitelist e, sempre più, sistemi basati su Intelligenza Artificiale (AI) e Machine Learning (ML) per classificare dinamicamente i domini e proteggere dalle minacce zero-day.

Il pannello web sviluppato in questa tesi non è il filtro stesso, ma un'interfaccia di gestione e analisi per il sistema di filtraggio. Questo implica che la sua architettura deve essere progettata per interagire efficacemente con un'infrastruttura di sicurezza critica e ad alte prestazioni, presentando i dati in modo chiaro e supportando la configurazione delle policy che verranno poi applicate dai motori di filtraggio DNS.

Capitolo 3

Analisi

Capitolo 4

Design

Capitolo 5

Implementazione

Bibliografia

[BPVdWDM13] E. G. Boix, K. Pinte, S. Van de Water, and W. De Meuter. Object-oriented Reactive Programming is Not Reactive Object-oriented Programming. Technical Report vub-soft-tr-13-16, Vrije Universiteit Brussel, Brussel, 2013.

[dLGM+13] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, David T. Garlan, Matthew B. dwyer, Robert M. France, et al. Programming and reasoning about actors that share state. *Journal of Functional Programming*, 23(4):343–393, 2013.

[Mak16] Toshiaki Maki. Data Microservices with Spring Cloud Stream, Task and Data Flow JSUG SpringDay. SlideShare, 2016.

[MGCS21] Marko Mijač, Antonio García-Cabot, and Vjeran Strahonja. Reactor Design Pattern. *TEM Journal*, 10(1):18–30, February 2021.

[MGCS23] Marko Mijač, Antonio Garcia-Cabot, and Vjeran Strahonja. REFRAME-A software framework for managing reactive dependencies in object-oriented applications. *SoftwareX*, 24:101571, 2023.

[OMCJL24] João Pedro Oliveira Marum, H. Conrad Cunningham, James A. Jones, and Yanyan Liu. Following the Writer's Path to the Dynamically Coalescing Reactive Chains Design Pattern. *Algorithms*, 17(2):56, 2024.

[Wik24] Wikipedia. Reactive programming. `https://en.wikipedia.org/wiki/Reactive_programming`, 2024.

Acknowledgements

Optional. Max 1 page.