**UNIVERSITÀ DEGLI STUDI DELLA BASILICATA**

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

**RELAZIONE PROGETTO FINALE TAV**

**ASFJ: A Simple Firewall in Java**

Docente:

Ch.mo Prof. Gianvito Summa

Studente:

Michael Pio Stolfi 68787

**ANNO ACCADEMICO 2024-2025**

Sommario

[Sommario 2](#_Toc201860146)

[Introduzione 3](#_Toc201860147)

[Test 4](#_Toc201860148)

[Conclusioni 5](#_Toc201860149)

[Bibliografia 6](#_Toc201860150)

# Introduzione

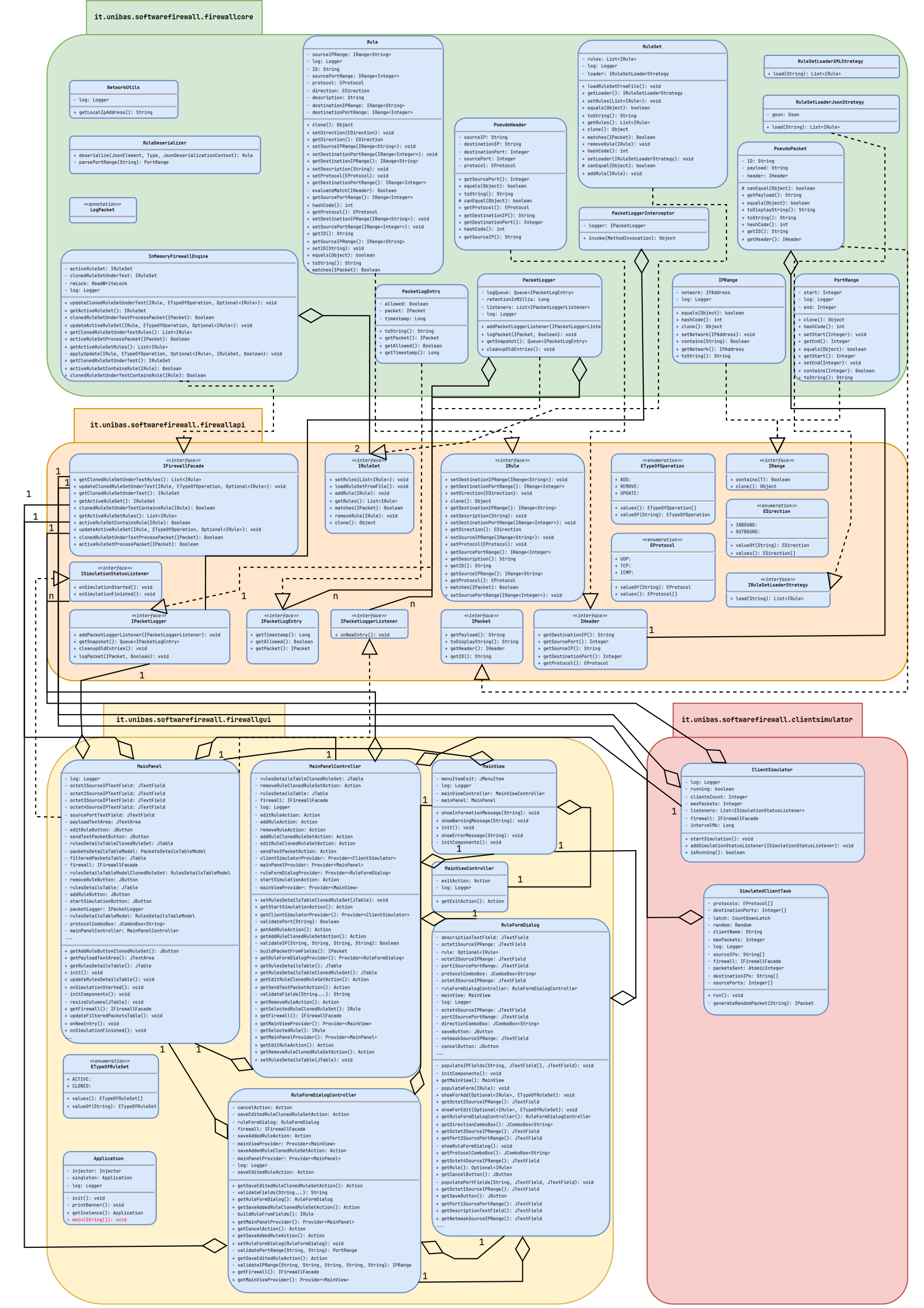
A Simple Firewall in Java (ASFJ) è un’applicazione Java desktop modulare che simula il comportamento di un firewall software. Il progetto permette di testare, monitorare e modificare dinamicamente regole di filtraggio su pacchetti generati da client concorrenti. L'interfaccia utente, costruita con Swing, consente un'interazione in tempo reale con il traffico e le regole, supportando anche la simulazione di ambienti di rete controllati per fini didattici o sperimentali.

ASFJ nonostante abbia fini didattici è stato sviluppato con un'attenzione particolare a: tenere alto l’isolamento aumentando la coesione (tramite la separazione dei moduli), all’estendibilità riducendo l’accoppiamento (tramite l’utilizzo estensivo delle interfacce), e all'applicazione dei principi avanzati dell’ingegneria del software, come l’Inversion of Control (IoC), i Design Pattern (DP), e la personalizzazione del comportamento applicativo tramite file properties. Per tale applicativo come in tutti gli ambienti professionali non si è ignorato anche la importante fase finale dello sviluppo applicativo ovvero la fase di deploy. A tale scopo esso è stato impacchettato sotto forma di uber-JAR; un uber-JAR, noto anche come fat JAR, è un file JAR che contiene non solo un programma Java, ma anche le sue dipendenze. Ciò significa che il JAR funziona come una distribuzione "all-in-one" del software, senza bisogno di altro codice Java.

Ovviamente questo approccio estremamente professionale allo sviluppo ha un chiaro inconveniente ovvero che la struttura applicativa stessa si complica e diviene per un osservatore esterno di difficile comprensione. Questo documento ha come fine: l’integrazione del README di progetto, la spiegazione della struttura applicativa, l’esposizione dei dettagli applicativi complessi e infine la discussione di come sono stati implementati gli argomenti centrali del progetto ovvero: Dependency Injection (DI), Aspect Oriented Programming (AOP), Thread e Sincronizzazione, e Clonazione.

# Test

Ciao ciao ciaio



# Conclusioni

L'analisi condotta ha evidenziato come l'uso dei design pattern possa migliorare significativamente la sicurezza di un'applicazione web, mitigando vulnerabilità come Broken Access Control, SQL Injection, Identification and Authentication Failures e Security Logging and Monitoring Failures . Il confronto tra il progetto senza design pattern e quello con un'implementazione sicura ha dimostrato che l'adozione di pattern strutturati permette di prevenire attacchi e ridurre le superfici di rischio.

Tuttavia, è importante sottolineare che i pattern da soli non garantiscono la sicurezza. Devono essere integrati all'interno di un approccio più ampio di security by design e defense in depth, che preveda una gestione attenta delle credenziali, logging sicuro, monitoraggio continuo e aggiornamenti costanti. Solo attraverso un'architettura ben progettata e una strategia di sicurezza completa è possibile garantire un sistema robusto e resiliente agli attacchi.

In sintesi, l'utilizzo dei design pattern rappresenta una best practice fondamentale nello sviluppo software sicuro, ma deve essere affiancato da una consapevolezza continua sulle minacce emergenti e da una strategia di protezione che combini più livelli di difesa.

# Bibliografia

1. OWASP. “OWASP Top Ten: 2021”. <https://owasp.org/Top10/>
2. AGID. “Linee guida per lo sviluppo del software sicuro”. <https://www.agid.gov.it/it/sicurezza/cert-pa/linee-guida-sviluppo-del-software-sicuro>
3. Microsoft. “Microsoft Security Development Lifecycle (SDL)”. <https://learn.microsoft.com/it-it/compliance/assurance/assurance-microsoft-security-development-lifecycle>
4. Port Swigger. “Authentication vulnerabilities”. <https://portswigger.net/web-security/authentication>
5. Port Swigger. “SQL injection”. <https://portswigger.net/web-security/sql-injection#what-is-sql-injection-sqli>