Politecnico di Torino

Corso di Laurea Magistrale
in Ingegneria Informatica

Sicurezza e Protezione dei Dati Sensibili in un Prodotto per Intercettazioni Legali: Servizi di Auditing e Licensing

Relatore
Prof. Giovanni Malnati

Candidato
Luigi Loria

Luglio 2023

Abstract

Questo elaborato si concentra sulla progettazione e realizzazione di un servizio di consultazione ed esportazione delle informazioni di auditing e di un servizio di gestione delle licenze per un prodotto rivolto ad intercettazioni legali. Inoltre, l'elaborato prevede lo studio e l'implementazione di un processo di offuscamento delle due applicazioni, per garantire la massima protezione della proprietà intellettuale e rendere più difficili alterazioni del funzionamento del software da parte di terzi.

In virtù di quanto detto poc'anzi, la necessità di gestire e analizzare dati sensibili rappresenta un tema di grande attualità, soprattutto in un contesto in cui l'utilizzo di strumenti digitali e la quantità di dati generati sono in costante aumento. L'ambito in questione tratta infatti dati particolarmente delicati, che richiedono l'adozione di tutte le misure necessarie per garantirne la protezione e la sicurezza.

In particolare, l'elaborato si concentra su vari aspetti della protezione dei dati, tra cui la protezione del codice sorgente

da parte di malintenzionati, sull'applicazione di algoritmi di cifratura per la protezione dei dati, sull'applicazione di firme digitali per autenticità, integrità e non ripudio, sull'ottimizzazione delle prestazioni in presenza di grandi numeri di dati e sull'ingegnerizzazione della soluzione per garantire manutenibilità e scalabilità del codice prodotto. Tuttavia, l'attenzione va oltre la semplice discussione di questi aspetti, proprio perché la trattazione approfondisce il problema in sé, il processo di progettazione e la sua realizzazione.

Nella soluzione proposta sono state impiegate tecnologie web all’avanguardia, tra cui il framework React con TypeScript per la parte frontend del servizio di gestione delle licenze e il framework Java Spring Boot per la parte backend di entrambe le applicazioni, mentre per la gestione dei dati è stato utilizzato il database non relazionale MongoDB.

Indice

Introduzione

Nell'odierno panorama digitale, la sicurezza e la protezione dei dati costituiscono un aspetto critico, in particolare in aree sensibili come quello legale. Con la crescente dipendenza dalle applicazioni web per la gestione e l'archiviazione di dati confidenziali, garantire la sicurezza di queste applicazioni è diventato fondamentale. Il presente lavoro si propone di affrontare questo problema progettando e implementando due servizi che rispondono alle esigenze del settore delle intercettazioni legali. Il primo servizio consente di gestire e creare in modo sicuro file di licenze per un prodotto specifico di questo settore, mentre il secondo consiste in un'applicazione per la consultazione e l'esportazione di informazioni di auditing prodotte da un sistema investigativo.

Oltre a questi servizi, il presente documento esplora anche l'uso di un processo di offuscamento del codice per migliorare ulteriormente la sicurezza delle applicazioni web. L'offuscamento del codice comporta la modifica deliberata del codice sorgente per renderlo difficile da comprendere o da decodificare. Lo scopo di questa tecnica è ridurre il più

possibile la probabilità di modifiche non autorizzate e proteggere la proprietà intellettuale.

Per il perseguimento di quanto detto, il presente elaborato sfrutta tecnologie all'avanguardia come Spring Boot e React per creare applicazioni web sicure e performanti in grado di soddisfare le esigenze del settore precedentemente menzionato. La loro efficace combinazione e implementazione è in grado di migliorare significativamente la sicurezza e l'affidabilità delle applicazioni.

Nei prossimi capitoli, i servizi implementati saranno esaminati in modo più dettagliato, con particolare attenzione a ciascun componente e al suo contributo alla funzionalità complessiva del sistema.

Il primo capitolo si occupa della progettazione e dello sviluppo del servizio di gestione delle licenze, trattando innanzitutto il backend, caratterizzandone inizialmente la parte architetturale, per poi concentrarsi in dettaglio sulla sfera della sicurezza, in cui vengono trattate le varie tecniche di protezione adottate, come l'uso della crittografia simmetrica e asimmetrica, l'utilizzo di HTTPS per le comunicazioni e l'applicazione di misure di protezione dalle comuni vulnerabilità del web. Successivamente è presente la sezione dedicata al frontend dell'applicazione, in cui si discutono i principali vantaggi del framework React per l'ottimizzazione delle prestazioni e il modo in cui migliorare la sicurezza del frontend utilizzando le pratiche più diffuse nel settore. Il capitolo tratta anche la gestione dei dati sensibili con MongoDB e l'implementazione

dell'autenticazione mediante FreeIPA. Il capitolo si conclude evidenziando l'importanza della sicurezza e della gestione sicura dei dati nelle applicazioni web.

Il secondo capitolo tratta la progettazione e lo sviluppo del servizio di gestione delle informazioni di auditing prodotto da un sistema, concentrandosi sull'ottimizzazione delle prestazioni e sulla gestione delle autorizzazioni in Spring Boot. Vengono descritte tecniche avanzate per l'ottimizzazione delle prestazioni e la gestione di grandi quantità di dati con MongoDB, come l'uso dell'indicizzazione e delle query di aggregazione. Il capitolo tratta anche la localizzazione e la traduzione dei dati richiesti dagli utenti, l'esecuzione di operazioni asincrone e la generazione di documenti PDF in Java con la libreria IText7, compresa l'applicazione della firma digitale con il supporto di BouncyCastle. Il capitolo termina sottolineando l'importanza dell'ottimizzazione delle prestazioni e della gestione delle autorizzazioni per garantire un'applicazione efficiente e sicura.

Il terzo capitolo illustra la ricerca, lo studio e l'implementazione di un processo di offuscamento del codice sorgente dei due servizi presentati in precedenza, fornendo una breve panoramica delle principali tecniche di offuscamento e deoffuscamento, descrivendo la ricerca degli offuscatori e i parametri di valutazione utilizzati, e analizzando i vantaggi e gli svantaggi di ciascuno. Il capitolo culmina ponendo enfasi sull'importanza dell'offuscamento per proteggere la proprietà intellettuale e rafforzare la sicurezza delle applicazioni, e con una valutazione complessiva degli offuscatori visionati.

Il quarto capitolo, infine, conclude il documento riassumendo ed evidenziando gli obiettivi raggiunti e offrendo alcuni spunti per sviluppi futuri.

Capitolo 1

Licensing Generator

Questo capitolo illustra il processo seguito per progettare e realizzare il servizio di gestione e creazione delle licenze. La prima parte riguarda la progettazione e lo sviluppo del backend utilizzando Java Spring Boot, concentrandosi sull'architettura generale e approfondendo in seguito gli aspetti legati alla sicurezza e alla protezione dei dati. La seconda parte esplora l'integrazione di FreeIPA in Spring Boot, con l'obiettivo di autenticare gli utenti dell'applicazione. Infine, il capitolo affronta la progettazione e lo sviluppo del frontend utilizzando React TypeScript, discutendo l'architettura e i modi con cui aumentarne le prestazioni e la sicurezza.

1.1 Introduzione

Il Licensing Generator è un'applicazione composta da un'interfaccia grafica e da un backend che consente agli utenti

di eseguire varie operazioni come l'autenticazione, la gestione dei clienti (creazione, modifica e cancellazione delle loro informazioni) e la generazione nonché il download dei file di licenza.

L'obiettivo principale dell'applicazione risiede nella protezione delle licenze acquistate dai clienti di un applicativo. Ogni licenza è un identificatore univoco di un dispositivo registrato nel sistema e fa parte di un file di licenza in cui sono contenute altre informazioni essenziali come la data di scadenza, il progetto a cui è associata e altro ancora. Per evitare modifiche non autorizzate, i file di licenza in questione vengono crittografati prima della loro memorizzazione, rendendo di fatto impossibile la modifica dei dati al loro interno.

La sicurezza di questa applicazione è di estrema importanza a causa della natura sensibile dei dati che gestisce. Qualsiasi modifica non autorizzata alle licenze avrebbe gravi conseguenze, tra cui perdite di fatturato, responsabilità legali e danni alla reputazione. Pertanto, l'implementazione e l'adozione di solide misure di sicurezza è fondamentale per garantire l'integrità e la riservatezza dei dati gestiti.

1.2 Progettazione e Sviluppo del Backend

Il backend dell'applicazione è realizzato con Java, un linguaggio di programmazione ampiamente adottato e noto per la sua robustezza e portabilità (1). Per sfruttare i vantaggi del linguaggio e semplificare il processo di sviluppo, il backend è stato sviluppato utilizzando Spring Boot, un framework in grado di offrire un supporto immediato per diverse funzionalità, tra cui la presenza

di server integrati, l'integrazione di database e sicurezza, e molto altro.

Nelle sottosezioni seguenti, verranno illustrati i vari componenti del backend e il modo in cui vengono impiegati per garantire la sicurezza e le funzionalità desiderate.

1.2.1 Architettura MVC con Service e Repository

Il pattern architetturale Model-View-Controller (MVC) è diffuso nell'ingegneria del software fin dalla sua nascita, negli anni '70, ed è presto divenuto uno standard per lo sviluppo d'interfacce utente. Lo scopo principale dell'architettura MVC è quello di separare i compiti relativi all'interfaccia utente (View), alla gestione delle richieste (Controller) e all'archiviazione dei dati (Model). La logica applicativa è invece a carico degli ultimi due livelli citati.

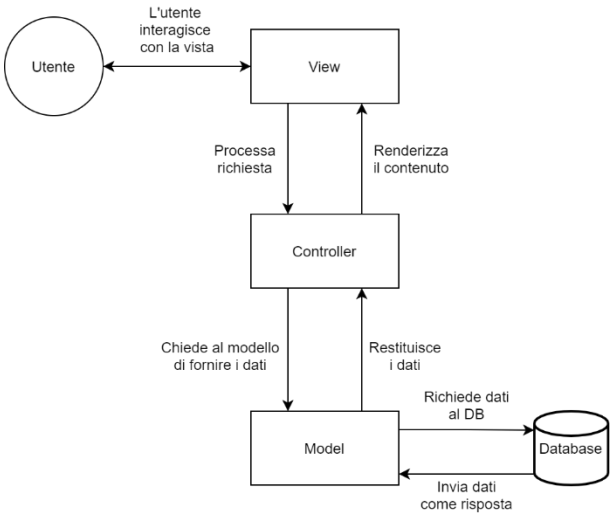


Figura 1: Tipica interazione tra componenti del pattern MVC

Il Model (modello) è responsabile della gestione e della manipolazione dei dati memorizzati nel sistema, oltre a fornire un'interfaccia al Controller per accedere

e modificare questi ultimi. La View (vista) è responsabile della presentazione dei dati all'utente in modo visivo. Essa riceve i dati dal Controller e li formatta per la visualizzazione. Il Controller funge da intermediario tra la vista e il modello, in quanto riceve gli input dalla prima, li elabora utilizzando la logica di business, interagisce con il Model per ottenere o modificare i dati, e aggiorna la vista con i risultati (2).

Tuttavia, con l'aumentare della complessità dell'applicazione, la struttura tende a diventare troppo complessa e difficile da mantenere. Per risolvere questo problema, è possibile introdurre un livello di servizio (Service), il quale rappresenta un modo per incapsulare la logica di business dell'applicazione e fornire un'interfaccia semplificata al Controller per interagire con il livello inferiore. Infatti, il livello di servizio è responsabile della definizione di un insieme di servizi che il Controllor può utilizzare per eseguire operazioni sui dati gestiti dal sistema. Questo favorisce il riutilizzo del codice, la modularità e rende l'applicazione più facile da mantenere e testare. In aggiunta, il tradizionale Model può essere sostituito da un livello chiamato Repository, responsabile della gestione della persistenza dei dati. Esso fornisce un'interfaccia al livello di servizio per interagire con il database e astrae i dettagli di come i dati vengono memorizzati e recuperati. Separando le preoccupazioni relative alla memorizzazione dei dati e alla logica di business, il livello Repository promuove la modularità e rende più facile passare da una tecnologia di memorizzazione dei dati all'altra.

Il framework Spring fornisce diverse funzionalità utili per implementare l'architettura MVC con servizi e repository.

Per implementare il livello di servizio, è possibile definire un insieme di servizi come Spring beans, ossia oggetti creati e configurati dallo "Spring container". Definendo i servizi come bean, si possono sfruttare le funzionalità di iniezione delle dipendenze di Spring per iniettare facilmente questi servizi in altri componenti, come i controller o altri servizi, senza doverli creare direttamente. Per implementare il livello Repository, è possibile utilizzare Spring Data, il quale fornisce un insieme di interfacce e implementazioni per interagire con i database.

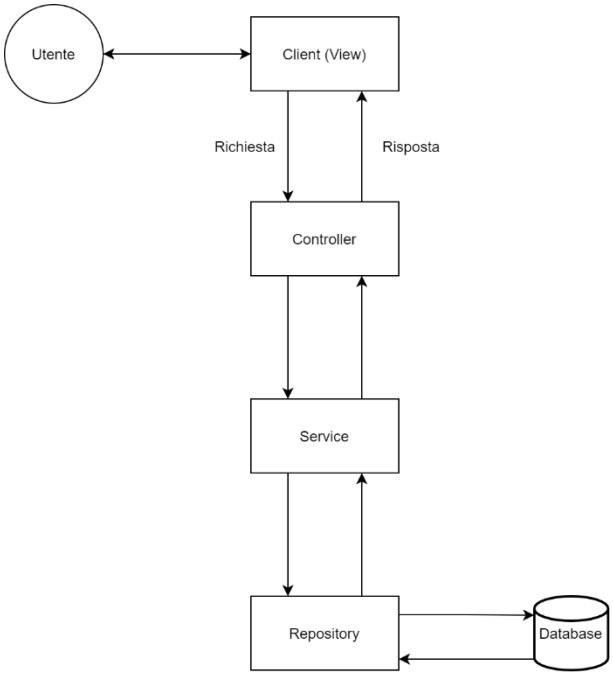


Figura 2: Interazione tra i componenti dell'architettura MVC con Service e Repository

L'interazione tra i componenti dell'architettura precedentemente descritta può essere riepilogata come segue. L'utente interagisce con il componente View, tipicamente implementato come parte di un'applicazione client, inviando una richiesta HTTP al Controller,

il quale la elabora utilizzando la logica di business del Service Layer, eventualmente interagendo con il Repository Layer per recuperare o aggiornare i dati nel database. I risultati vengono quindi restituiti al Controller, il quale provvede a inviare una risposta al client. Quest'ultimo aggiorna la View con la risposta, fornendo all'utente le informazioni o le funzionalità richieste.

1.2.2 Architettura dei Test con JUnit5 e Testcontainers

I test sono una parte essenziale nello sviluppo del software in quanto ne garantiscono la qualità e l'affidabilità. Per raggiungere questo obiettivo, vengono spesso utilizzati framework di testing come JUnit5 e strumenti di testing come Testcontainers. JUnit5 è un framework di testing molto diffuso, dotato di numerose funzionalità, tra cui un supporto per i test parametrizzati, i test dinamici e l'esecuzione parallela dei test. D'altra parte, Testcontainers è tipicamente utilizzato per creare e gestire container Docker a scopo di test, rendendo più semplice la creazione e la rimozione di container "usa e getta" per i test d'integrazione, in modo che questi possano essere eseguiti indipendentemente e partano sempre da uno stato iniziale comune.

L'architettura di test per un'applicazione Spring Boot include test d'integrazione e test unitari per i livelli Repository, Service e Controller. In particolare, i test unitari dovrebbero esclusivamente testare metodi indipendenti, senza dipendenze esterne. I test d'integrazione, invece, devono testare l'interazione tra i diversi livelli dell'applicazione, per garantire che essi funzionino correttamente. Per quest'ultimi, viene consigliato, come accennato in precedenza, l'uso di Testcontainers per creare un contenitore del database e popolarlo con i dati di test. Questo assicura

che i test vengano eseguiti in un ambiente simile a quello di produzione, ma senza il rischio di comprometterlo. Per i test unitari, il mocking non è necessario per i metodi che sono già isolati dalle dipendenze esterne. Tuttavia, quando si testa il livello del Controller, è necessario mockare il framework MVC per simulare le richieste e le risposte HTTP. A questo scopo, è possibile utilizzare la classe MockMVC, la quale permette di chiamare direttamente i metodi del controller senza affidarsi all'ambiente di rete, rendendo i test più veloci e affidabili. Inoltre, questa classe fornisce un insieme di strumenti di verifica che facilitano la verifica della correttezza delle richieste, rendendo il processo di test più efficiente e conveniente.

Come si è visto nella sezione precedente, il livello repository è responsabile dell'interazione con il database e pertanto i test sono necessari per garantire che il livello in questione funzioni correttamente in condizioni reali. In particolare, i test d'integrazione devono verificare che il repository possa eseguire le operazioni, specificate nella sua interfaccia, sul database. Invece, i test unitari del livello repository sono solitamente sconsigliati, poiché il database è un componente critico dell'applicazione e la sua simulazione non sarebbe rappresentativa delle condizioni effettive di utilizzo.

Il livello di servizio contiene la logica di business dell'applicazione. I test unitari per questo livello devono testare i singoli metodi in modo isolato, per garantire che funzionino come previsto. Per quanto riguarda i test d'integrazione, essi devono verificare se l'interazione tra il livello di servizio e il livello di repository funzioni

correttamente, verificando che il servizio possa eseguire correttamente le operazioni specificate sul database e che la logica di business dell'applicazione funzioni come previsto.

Il livello controller è responsabile della gestione delle richieste in arrivo e della trasmissione delle risposte. I test unitari per questo livello devono testare eventuali metodi autonomi e isolati. Un test d'integrazione dovrebbe testare l'interazione tra il livello del controllore e quello del servizio, inviando richieste HTTP agli endpoint del controllore, facendo affidamento sulla classe MockMVC, e verificando che le risposte siano corrette.

In generale, quando si scrivono i test, è essenziale seguire lo schema Arrange-Act-Assert (AAA). Questo schema separa il test in tre parti: l'impostazione (Arrange), l'azione (Act) e la verifica (Assert). In particolare, la prima parte deve preparare i dati di test e impostare l'ambiente, l'azione deve eseguire l'operazione da testare e la verifica deve controllare che il risultato dell'operazione sia del tutto corretto.

È anche importante usare con parsimonia gli oggetti "mock". Sebbene i mock possano essere utili per isolare il codice in fase di test dalle dipendenze esterne, possono anche rendere i test fragili e difficili da gestire. Quando si esegue un mock, si sovrascrive la logica della classe oggetto del mock. La vera logica viene nascosta dietro le quinte ed è lì che i bug amano vivere (3). Pertanto, è importante usare gli oggetti mock con moderazione, ad esempio solo con un qualsiasi servizio esterno che non abbia nulla a che fare direttamente con il dominio aziendale. Seguendo questo approccio, è possibile garantire che i test siano

efficaci, affidabili e controllabili e che l'applicazione svolga il suo lavoro anche nell'ambiente di produzione.

1.2.3 Validazione delle Richieste API

La convalida delle richieste API è un aspetto cruciale nella costruzione di un'applicazione web affidabile e sicura. Quando gli utenti inviano dati al server, è importante assicurarsi che questi soddisfino determinati criteri prima di elaborarli ulteriormente. La mancata convalida dei dati immessi dall'utente potrebbe causare diversi problemi, come l'elaborazione errata dei dati o addirittura l'esposizione di vulnerabilità di sicurezza.

Uno degli aspetti chiave della validazione delle richieste API è la gestione degli errori. Quando un utente invia dati non validi, l'applicazione deve fornire un responso comprensibile all'utente, indicando cosa sia andato storto. Tuttavia, è importante trovare un equilibrio tra il fornire informazioni sufficienti all'utente per capire cosa sia accaduto e il non fornirne troppe, perché altrimenti esse potrebbero essere sfruttate dagli attaccanti. Come regola generale, i messaggi di errore dovrebbero essere chiari e concisi, evitando di rivelare troppe informazioni sul funzionamento interno dell'applicazione. Ad esempio, invece di restituire un messaggio di errore dettagliato che includa una stack trace o altri dettagli tecnici, può essere sufficiente fornire un messaggio di errore generico come "Si è verificato un errore durante l'elaborazione della richiesta".

Altrettanto importante è utilizzare correttamente i codici di stato HTTP per indicare se la richiesta sia stata completata con successo

o meno. Alcuni codici di stato, come “401 Unauthorized” e “404 Not Found”, sono già ampiamente comprensibili e non necessitano di ulteriori precisazioni. Tuttavia, per errori specifici come “422 Unprocessable Entity”, può essere utile fornire ulteriori dettagli su cosa sia andato storto. Invece, quando si verifica un errore inatteso sul server, sarebbe più sicuro restituire un messaggio di errore generico con il codice di stato “500 Internal Server Error”. Questo perché esso può indicare qualsiasi tipo di errore non controllato che potrebbe verificarsi all'interno dell'applicazione e fornire troppi dettagli nel messaggio di errore potrebbe rivelare informazioni sensibili sul funzionamento interno dell'applicazione.

Per implementare la validazione delle richieste API in un'applicazione Spring Boot, si possono usare annotazioni e validatori personalizzati per definire regole di validazione specifiche per diversi tipi di dati e per fornire messaggi di errore personalizzati quando tali regole non sono soddisfatte. Ad esempio, è possibile definire un'annotazione per garantire che un indirizzo e-mail sia in un formato valido, oppure un validatore per verificare che i dati inviati corrispondano a specifici vincoli di formato.

Inoltre, Spring Boot fornisce un supporto integrato per la validazione di strutture di dati, come i DTO o gli oggetti di dominio, utilizzando le annotazioni @Valid e @Validated. Queste annotazioni attivano il processo di validazione per la struttura dati annotata, assicurando che essa soddisfi i vincoli di validazione specificati. Quando una struttura di dati è annotata con @Valid, Spring Boot convaliderà automaticamente

la stessa utilizzando i vincoli di validazione predefiniti definiti per ogni campo. Ad esempio, se un campo è annotato con il vincolo @NotNull, Spring Boot controllerà automaticamente che il campo non sia nullo. L'annotazione @Validated è usata per attivare un processo di validazione personalizzato in Spring Boot. Quando una struttura dati è annotata con @Validated, Spring Boot convaliderà la struttura dati utilizzando i vincoli di validazione personalizzati definiti in un gruppo di validazione separato. Questo può essere utile quando è necessario applicare regole di validazione diverse per casi d'uso o scenari diversi.

Oltre alle annotazioni e ai validatori, Spring Boot fornisce un potente meccanismo per la gestione centralizzata degli errori: il RestControllerAdvice. Questo permette di definire una singola classe per gestire tutte le eccezioni e gli errori che si verificano durante la gestione delle richieste, fornendo un approccio coerente e centralizzato alla gestione degli errori in tutta l'applicazione.

1.2.4 Gestione dei Logs con SLF4J

Quando si sviluppa un'applicazione web, la gestione dei log è un aspetto essenziale per garantire affidabilità, prestazioni e sicurezza dell'applicazione. Per rendere la gestione dei log più semplice per i programmatori, Java fornisce una serie di framework come log4J, logback, ecc.

Spring Boot è dotato di SLF4J, che è un'astrazione di tutti questi framework di log. Esso è l'acronimo di "Simple Logging Façade for Java" e consente di lavorare con qualsiasi framework di logging con un'unica dipendenza. Ogni framework di logging è composto da tre elementi: un Logger che cattura i messaggi,

un Formatter che li formatta e un Handler che li distribuisce stampandoli sulla console o memorizzandoli in un file (4).

Per gestire i log delle richieste HTTP in Spring, è possibile creare un filtro di log che intercetta le richieste HTTP in entrata e registra i dettagli della richiesta e della risposta, come il metodo HTTP, l'URI, il codice di stato e il tempo di risposta. L'implementazione di questo filtro è semplice e può essere facilmente configurato per registrare le richieste su file, database o console. Sebbene la creazione di un filtro di log sia un modo efficace per gestire i log delle richieste HTTP in Spring, esso può anche introdurre alcuni svantaggi. Ad esempio, la memorizzazione di tutte le richieste in arrivo può generare un gran numero di voci di log, che possono influire sulle prestazioni dell'applicazione e consumare spazio su disco, soprattutto se la risposta contiene molti dati. Inoltre, possono essere memorizzate informazioni sensibili presenti nelle intestazioni o nei corpi delle richieste, rendendo necessario filtrare i dati riservati soprattutto se il mezzo di memorizzazione dei log non è sicuro.

In Spring Boot i log dell'applicazione possono essere configurati con uno script di configurazione, chiamato logback-spring, espresso in XML. Esso fornisce un modo flessibile e completo per definire i livelli, ad esempio INFO, WARN, ERROR, gli appender, per specificare dove devono essere inviati i messaggi, e i formati dei log, per specificarne la struttura e il contenuto.

È possibile anche configurare i log per ambienti diversi, sfruttando i profili Spring, i quali consentono di definire impostazioni specifiche per ambiti diversi, come sviluppo, test e produzione. Ogni profilo può avere le proprie impostazioni

di configurazione sia nelle proprietà dell'applicazione che nella configurazione dei log.

Ad esempio, in un ambiente di sviluppo, si potrebbe voler visualizzare i log sulla console per facilitare il debug e la risoluzione dei problemi. Per configurare la stampa dei log sulla console, il file logback-spring.xml può essere modificato per includere un elemento appender di tipo “console”.

Tuttavia, in un ambiente di produzione, la stampa dei log sulla console non è di solito una buona idea per motivi di sicurezza; pertanto, di solito vengono memorizzati in un file. Per configurare la memorizzazione dei log in un file, il file logback-spring.xml può essere modificato per includere un elemento appender di tipo “file-rolling”, il quale fornisce anche la possibilità di definire altre impostazioni, come il percorso in cui memorizzare i file, il limite di dimensione totale, i giorni massimi di cronologia da memorizzare e altro ancora.

1.2.5 Spring Security: Protezione contro CSRF e XSS

Spring Security è la scelta principale per introdurre sicurezza a livello applicativo nelle applicazioni Spring. In generale, il suo scopo è quello di offrire un sistema altamente personalizzabile per implementare autenticazione, autorizzazione e protezione dagli attacchi più comuni (5). Esso ottiene questo risultato attraverso l'uso di filtri per il protocollo HTTP e di tecniche per intercettare e proteggere le chiamate ai metodi interni.

Sfruttando componenti che possono essere configurati tramite specifiche annotazioni e l’utilizzo dello Spring Expression Language (SpEL), è possibile introdurre sicurezza a livello applicativo in modo

più efficiente e meno dispendioso in termini di tempo, evitando la necessità di scrivere codice verboso. Tuttavia, è importante tenere presente che Spring Security, da solo, non protegge un'applicazione o i dati sensibili a riposo o in transito (5). Infatti, in un sistema a livelli, ognuno di essi e le relative comunicazioni sono a rischio e devono essere adeguatamente protetti. Pertanto, spetta allo sviluppatore comprendere e utilizzare correttamente Spring Security e adottare adeguate misure aggiuntive per garantire che l'applicazione e i dati sensibili siano protetti a ogni livello del sistema.

Il modo più efficace per affrontare eventuali minacce informatiche è quello di rimanere consapevoli degli scenari in cui esse si verificano e adottare tempestivamente misure preventive (6). Questo è particolarmente vero quando si tratta di prevenire attacchi lato client, come Cross-Site Request Forgery (CSRF) e Cross-Site Scripting (XSS), che possono rappresentare un rischio significativo per la sicurezza delle applicazioni web. L'Open Web Application Security Project (OWASP) identifica CSRF e XSS come due dei rischi più comuni e pericolosi per la sicurezza delle applicazioni web (7). Pertanto, per proteggere i dati sensibili degli utenti e impedire accessi non autorizzati, è essenziale essere vigili nel tentativo di prevenire tali attacchi.

Nello specifico, CSRF è un attacco in cui una terza parte costringe un utente a eseguire azioni su un sito in cui è attualmente autenticato. Per illustrare il funzionamento di un attacco CSRF, si consideri il seguente scenario (illustrato in Figura 3): un utente connesso al proprio conto bancario visita un sito web malevolo contenente un modulo nascosto che, al caricamento

della pagina, invia una richiesta al sito web della banca per trasferire fondi al conto dell'attaccante. Poiché l'utente ha già effettuato l'accesso al sito originale, la richiesta viene effettuata con il suo token di autenticazione e il trasferimento viene avviato senza che l'utente ne sia a conoscenza o abbia dato il proprio consenso.

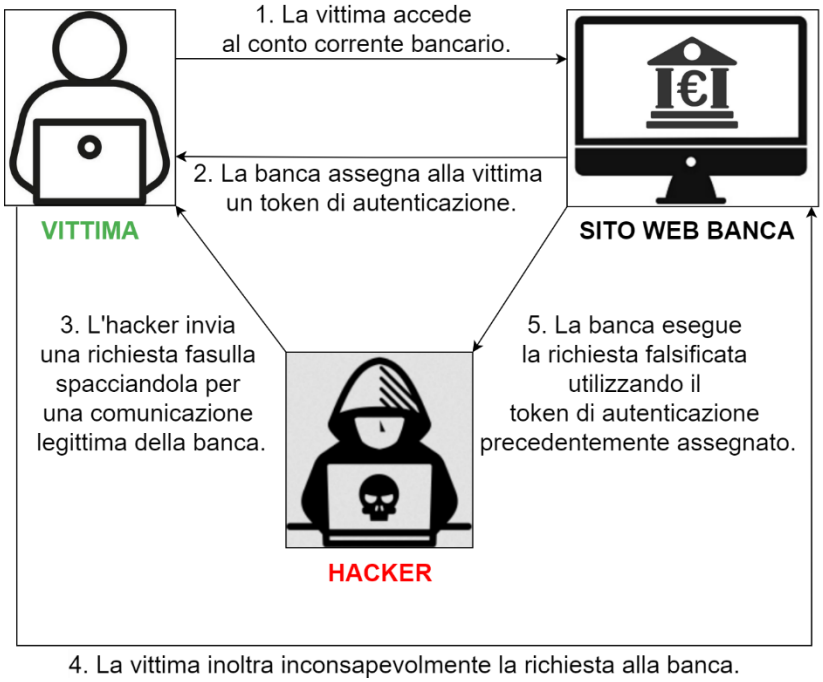


Figura 3: Esempio di attacco CSRF su un sito web bancario

Per prevenire questo tipo di attacco, è possibile utilizzare un sistema di protezione che richiede al client di includere uno speciale token in ogni richiesta inviata al server. In Spring, la classe CookieCsrfTokenRepository consente di gestire i token CSRF generati dal server e di memorizzarli come cookie nel client. Si può inoltre utilizzare la classe CsrfTokenRequestAttributeHandler per aggiungere il token CSRF alle richieste HTTP in uscita, assicurando che il server possa verificare l'autenticità di ciascuna

di esse. In particolare, le classi nominate si occupano di aggiungere il token alla risposta generata dal server, tramite l'attributo "_csrf", rendendolo disponibile al frontend dell'applicazione, che potrà così includerlo nelle richieste successive. Questa protezione può essere facilmente aggiunta alla catena dei filtri di sicurezza di Spring Security, utilizzando la classe CsrfCookieFilter, che intercetta le richieste in arrivo e verifica il token CSRF prima di permettere alla richiesta di procedere.

Tuttavia, è importante notare che i token CSRF possono essere soggetti a furto se un attaccante è in grado di eseguire un attacco Cross-Site Scripting (XSS) con successo. XSS è un attacco in cui un utente malintenzionato introduce uno script lato client in una pagina web con il quale può ad esempio aggirare la protezione CSRF. Ciò può avvenire sia compromettendo il server utilizzando una vulnerabilità nota o sfruttando un input scarsamente protetto per aggiungere uno script malevolo, il quale viene attivato ogni volta che un utente visita il sito. Ad esempio, "><script>alert(1);</script>"@example.org è un indirizzo e-mail potenzialmente valido per un input in quanto conforme allo standard RFC 5321 (8), ma invece contiene uno script “dannoso” che potrebbe essere eseguito e causare danni.

Per mitigare questo problema, è indispensabile seguire le principali pratiche di prevenzione degli XSS, come la sanificazione degli input inseriti dall'utente, sia sul frontend che sul backend, e l'implementazione di criteri restrittivi di sicurezza dei contenuti (CSP). Nello specifico, la sanificazione degli input

comporta la rimozione o la codifica di qualsiasi carattere o script potenzialmente dannoso, mentre il CSP è un meccanismo di sicurezza che limita le origini dei contenuti che possono essere caricati su una pagina web.

Spring Security prevede la possibilità di aggiungere la protezione CSP alla catena dei filtri di sicurezza dell'applicazione, configurando opportunamente le intestazioni aggiunte alla risposta, a cui è possibile abilitare l'integrata protezione XSS fornita dai browser moderni e definire una politica CSP tramite il metodo contentSecurityPolicy, ad esempio come "script-src 'self'; form-action 'self'". Questo CSP crea la cosiddetta "same origin policy" ovvero specifica che gli script possono essere caricati solo dallo stesso dominio dell'applicazione web (script-src 'self') e le azioni dei form possono essere inviate solo allo stesso dominio (form-action 'self'). In altri termini, gli script o le azioni dei moduli caricati da fonti esterne, come quelli iniettati da un utente malintenzionato, verranno bloccati dal browser.

Nei casi in cui sia necessario allentare la politica appena descritta, è possibile introdurre il Cross-Origin Resource Sharing (CORS). CORS è un meccanismo di sicurezza implementato nei browser per impedire a una pagina web di accedere a risorse (ad esempio API) su un altro dominio, a meno che quest'ultimo non abbia specificamente autorizzato l'accesso attraverso l'uso dell'intestazione HTTP "Access-Control-Allow-Origin". Questo può essere particolarmente utile durante la fase di sviluppo di un'applicazione web, in quanto il frontend e il backend sono tipicamente sviluppati su due server separati.

Spring Security fornisce supporto per l'integrazione di CORS in modo simile alle precedenti protezioni analizzate. In particolare, è possibile creare una classe di configurazione che consenta a un filtro di aggiungere l'intestazione "Access-Control-Allow-Origin" alle risposte del server, consentendo al frontend di accedere alle risorse del server. Questo filtro può anche specificare l'origine consentita, i metodi HTTP consentiti e le intestazioni consentite. Nella catena dei filtri di sicurezza, è poi possibile fare riferimento alla classe di configurazione CORS creata e applicarla a tutte le richieste, in modo da consentire la condivisione di risorse di origine diversa nell'applicazione.

1.2.6 Crittografia e Sicurezza Avanzata

Nella sottosezione precedente si è visto come Spring Security fornisca un robusto insieme di strumenti per la protezione delle applicazioni web contro le più comuni minacce alla sicurezza, come gli attacchi CSRF e XSS. Introducendo meccanismi come i token CSRF, le politiche CSP e le configurazioni CORS, è infatti possibile proteggere le applicazioni da attacchi potenzialmente dannosi e garantire la sicurezza dei dati degli utenti. Tuttavia, è anche importante tenere in considerazione il fatto che le minacce possono derivare da vari aspetti di un sistema e, pertanto, la sicurezza va sempre considerata come un processo continuo che richiede attenzione e adattamento costanti. A prescindere dal fatto che si tratti di proteggere l'archiviazione dei dati, la comunicazione di rete o l'accesso degli utenti, ogni aspetto di un sistema deve essere considerato e valutato per individuare potenziali vulnerabilità.

Un passo fondamentale per la sicurezza di un'applicazione web è infatti la crittografia del canale di comunicazione tra il client

e il server tramite l'utilizzo di HTTPS. Ogni richiesta HTTP non cifrata rivela informazioni sul comportamento degli utenti ed è ormai comune che la navigazione in chiaro venga intercettata e tracciata per scopi malevoli. Al giorno d'oggi non esiste un traffico web non sensibile e i servizi offerti non dovrebbero dipendere dalla bontà degli operatori di rete per garantire la sicurezza del canale di comunicazione utilizzato (9). Proprio per queste ragioni, il protocollo HTTPS diventa fondamentale, in quanto fornisce un ulteriore livello di protezione contro le minacce legate alla sicurezza, come gli attacchi man-in-the-middle e l'intercettazione dei dati. Inoltre, esso migliora l'efficacia di altre misure di sicurezza come i token CSRF, complicando il lavoro degli attaccanti nell'intercettare e rubare informazioni sensibili.

Per abilitare il protocollo HTTPS in un'applicazione web, occorre seguire diversi passaggi. Il primo è ottenere un certificato SSL, necessario per stabilire una connessione sicura tra il client e il server. Questo può avvenire generando un certificato auto firmato o utilizzare un certificato esistente emesso da un'autorità di certificazione affidabile. Se si decide di generare un certificato SSL auto firmato, è possibile utilizzare strumenti come OpenSSL o Keytool per crearlo. Tuttavia, bisogna tenere presente che i certificati auto firmati non sono affidabili per impostazione predefinita e sarà necessario configurare le applicazioni client per accettarli. Una volta ottenuto il certificato, è necessario configurare l'applicazione per utilizzare HTTPS. A tale scopo, in Spring è possibile aggiungere opportune proprietà di configurazione dell'applicazione, come "server.ssl.key-store",

relativa al percorso dell'archivio di chiavi che contiene il certificato SSL, "server.ssl.key-store-password", che contiene la password utilizzata per accedere all'archivio precedentemente menzionato e "server.ssl.key-store-type", ovvero il tipo di archivio di chiavi utilizzato (JKS o PKCS12). Quando si utilizza Spring Security, è possibile configurarlo per bloccare automaticamente qualsiasi richiesta proveniente da un canale HTTP non sicuro e per reindirizzarla verso HTTPS. Questo si può ottenere aggiungendo un vincolo di sicurezza che specifichi che il canale di comunicazione debba essere confidenziale. In Spring Boot è possibile utilizzare il bean ServletWebServerFactory per configurare il server web Tomcat con il vincolo di sicurezza appena specificato e aggiungere un connettore aggiuntivo per il traffico HTTP. Questo assicura che tutte le richieste siano reindirizzate correttamente su HTTPS e dunque che il canale di comunicazione utilizzato sia sempre crittografato. Infine, è necessario distribuire il certificato SSL ai clienti. Si noti come in alcuni casi, può essere che sia il cliente stesso a fornire un certificato SSL per la propria applicazione. In ogni caso, durante la fase di sviluppo, è comunque utile generarne uno auto firmato a scopo di test e installarlo localmente per consentire al browser di fidarsi della nostra applicazione.

L'uso della crittografia è un elemento essenziale per proteggere dati sensibili e prevenirne la manomissione, soprattutto nel contesto del servizio in esame, dove i file di licenza rappresentano un asset fondamentale, la cui modifica può comportare danni economici. Per garantire la protezione di tali

file, vengono infatti utilizzati algoritmi di crittografia a chiave simmetrica e asimmetrica, quali RSA e AES GCM.

RSA è un algoritmo di crittografia asimmetrica che utilizza una coppia di chiavi pubbliche e private dove, per garantire riservatezza, la chiave pubblica viene utilizzata per crittografare i dati, mentre la chiave privata viene utilizzata per decifrarli. RSA viene spesso utilizzato per condividere in modo sicuro le chiavi di crittografia simmetriche, come nel caso trattato. AES GCM è invece un algoritmo di crittografia simmetrica utilizzato per la cifratura e la decifratura dei dati. È una modalità di funzionamento della crittografia AES ampiamente adottata per le sue prestazioni e in grado di garantire sia la riservatezza che l'integrità dei dati (10). Essa utilizza una chiave simmetrica sia per la crittografia che per la decifrazione ed è spesso utilizzata per crittografare grandi quantità di dati in modo efficiente.

Quando viene creato un nuovo cliente, viene generata una coppia di chiavi RSA la cui chiave pubblica viene in seguito distribuita al relativo cliente. Quest'ultima è utilizzata per generare un'impronta digitale, la quale rappresenta il marcatore univoco del server a cui deve essere associato il file di licenza. Il formato dell'impronta digitale (come mostrato in Figura 4) è una combinazione di due blocchi, rispettivamente cifrati tramite RSA e AES GCM, dove il primo blocco contiene la chiave di cifratura simmetrica e il vettore di inizializzazione per il blocco successivo, il quale invece contiene la chiave AES GCM da utilizzare per la cifratura del file di licenza. Il tutto è codificato in formato

Base64 per garantire compatibilità con una più ampia gamma di sistemi e per semplificarne la trasmissione.

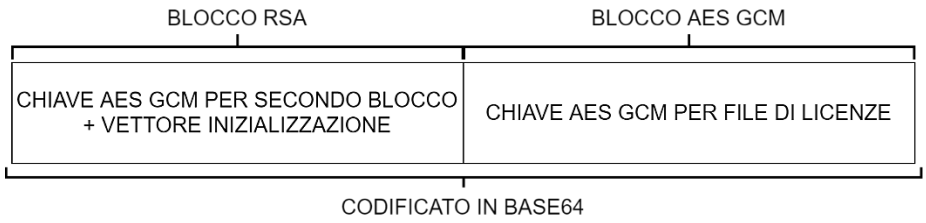


Figura 4: Formato dell'impronta digitale utilizzata per la creazione di un file di licenze

Per creare un file di licenze, è necessario estrarre la relativa chiave dall'impronta digitale innanzitutto decodificando quest'ultima dalla sua rappresentazione in base 64, separando poi i due blocchi di dati di cui è composta, il che può essere effettuato poiché la lunghezza di ciascun blocco dipende dalla lunghezza della rispettiva chiave utilizzata per la cifratura (ad esempio, 512 byte per RSA) e decifrando il primo blocco utilizzando la chiave privata RSA del relativo cliente. Dopo aver recuperato la chiave di cifratura AES e il vettore d'inizializzazione, è possibile ottenere la chiave di cifratura del file di licenze decifrando il secondo blocco dell'impronta digitale. Una volta ottenuta la chiave, il file di licenza può essere creato e protetto mediante crittografia simmetrica.

Entrando nel dettaglio, il file di licenze contiene tutte le licenze acquistate dal cliente e la sua scadenza. Ogni licenza è composta da un identificatore univoco e dall'elenco dei sistemi operativi (ad esempio, Android, IOS, ecc.) supportati e mappati all'interno di un insieme di bit. Come menzionato in precedenza, la chiave simmetrica estratta dall'impronta digitale viene utilizzata per la crittografia del file di licenze generato insieme a un vettore

di inizializzazione generato in modo indipendente per ogni file. Il testo cifrato risultante da tale cifratura sarà poi codificato anch'esso in formato Base64, sempre per usufruire dei benefici precedentemente menzionati. L'effettivo file di licenze memorizzato all'interno del sistema è composto da una concatenazione di tre informazioni: la somma delle lunghezze in termini di byte del testo cifrato e del vettore di inizializzazione appena generati, il vettore di inizializzazione stesso e infine il testo cifrato.

La combinazione di RSA e AES GCM rappresenta un metodo efficace per la condivisione di chiavi di crittografia simmetriche, garantendo al contempo la riservatezza e l'integrità dei file prodotti. Utilizzando questi algoritmi, il servizio è in grado di assicurare che i file di licenze siano sicuri e protetti da accessi non autorizzati, poiché il file viene crittografato utilizzando una chiave nota solo al cliente e al sistema stesso; inoltre, l'algoritmo AES GCM fornisce anche la garanzia riguardo alla non manomissione del file grazie all'utilizzo di un codice di autenticazione del messaggio (MAC). Il MAC viene generato durante il processo di crittografia utilizzando la relativa chiave simmetrica e il contenuto del file di licenze. Quando il file di licenze viene decifrato, il MAC viene verificato utilizzando la stessa chiave e il contenuto del file di licenze decifrato. Se la verifica del MAC non va a buon fine, significa che il file di licenza è stato modificato o manomesso e il processo di decifratura fallirà.

1.2.7 Gestione dei Dati con MongoDB

MongoDB è un popolare database NoSQL ampiamente utilizzato per archiviare e gestire grandi volumi di dati. Una delle ragioni

principali per cui si sceglie MongoDB è la sua caratteristica di scalabilità, cioè di aumentare la capacità per gestire un carico aggiuntivo di dati e richieste, continuando a funzionare in modo ottimale. Invece delle classiche righe e colonne, MongoDB memorizza i dati in documenti flessibili in stile JSON, in modo che i campi possano variare da un documento all'altro e al contempo la struttura dei dati possa essere modificata nel tempo (11). Tutto ciò influisce anche sulle prestazioni, in quanto il processo di lettura e scrittura dei dati è più rapido, dal momento in cui tutte le informazioni di ciascuna entità sono solitamente memorizzate in un unico documento, a differenza dei classici database relazionali che devono scrivere e leggere i dati da molte tabelle per aggiornare o recuperare le informazioni, aumentando il carico del server e diminuendo la velocità (12).

Oltre alla sua scalabilità, MongoDB offre diversi vantaggi per l'archiviazione dei file. In particolare, GridFS è la specifica di MongoDB per fornire una soluzione robusta e scalabile per l'archiviazione di file di grandi dimensioni, come immagini, audio e video. GridFS è in grado di memorizzare file di dimensioni anche superiori al limite di 16 MB imposto per i documenti, memorizzando i dati binari in modo distribuito, ossia suddividendo il file in pezzi (chunk) e memorizzandoli all'interno della stessa collezione. Per impostazione predefinita, utilizza due collezioni "fs.files" e "fs.chunks" rispettivamente per memorizzare i metadati del file e i pezzi di cui è composto. Ogni pezzo è identificato da un campo id univoco, mentre fs.files funge da documento padre ed è referenziato da ogni chunk che lo compone (13). Questo approccio consente a GridFS di gestire file di grandi dimensioni in modo efficiente e senza consumare troppa memoria, poiché esso non viene caricato tutto in una volta.

Quando si integra MongoDB in un'applicazione Spring Boot, è possibile sfruttare la libreria Spring Data MongoDB per accedere ai dati memorizzati al suo interno, compresi i file salvati con GridFS. A tale scopo, la classe GridFsOperations è un potente strumento che fornisce una semplice API per la gestione dei file memorizzati con GridFS. Tramite questa classe, è possibile salvare, caricare e cancellare facilmente i file, oltre a eseguire altre operazioni di base come la ricerca e la modifica dei metadati dei file. Questa caratteristica è stata infatti impiegata per la memorizzazione delle impronte digitali prodotte dai clienti e dei file di licenze generati dal sistema.

Un'altra caratteristica particolarmente utile di MongoDB è il supporto per l'auditing dei dati, che consente di tenere traccia delle modifiche apportate al database, comprese le azioni degli utenti come inserimenti, aggiornamenti e cancellazioni. Questa funzione può essere preziosa per mantenere l'integrità e la sicurezza del database, in quanto consente di monitorare e rilevare modifiche non autorizzate o attività sospette. Infatti, secondo l'OWASP, MongoDB è vulnerabile a una serie di rischi comuni per la sicurezza delle applicazioni web, come attacchi di tipo “injection”, XSS e DoS, in cui gli attaccanti inondano il database di richieste, causandone il rallentamento o il crash. Per ridurre questi rischi, è importante seguire le buone pratiche per la sicurezza, come l'implementazione di autenticazione e di controlli di accesso efficaci (trattati nella sezione successiva), non fidarsi dei dati immessi lato client ed eseguire il cosiddetto “escape”, ovvero di convertire eventuali caratteri speciali o particolari sequenze di caratteri in un formato sicuro, di tutti i dati

sul lato server, la crittografia dei dati sensibili (come fatto per i file di licenze) e monitorare regolarmente i log di audit per individuare eventuali attività sospette.

1.3 Autenticazione SSO con FreeIPA

FreeIPA è una soluzione completa per la gestione delle identità degli accessi (IAM), in grado di fornire autenticazione, autorizzazione e informazioni sugli utenti in maniera centralizzata. Si tratta di un progetto open-source che si basa su molte altre tecnologie, tra cui 389 Directory Server, Kerberos e SSSD (14).

FreeIPA fornisce un meccanismo di autenticazione Single Sign-On (SSO) per consentire agli utenti di autenticarsi una sola volta e di accedere a più applicazioni e servizi senza dover inserire nuovamente le proprie credenziali. A tal fine, agisce come un fornitore d'identità, con il compito di autenticare gli utenti e di emettere token di sicurezza che possono essere utilizzati per accedere a risorse protette.

Esso offre diversi vantaggi legati all'autenticazione SSO, tra cui gestione centralizzata, sicurezza, scalabilità e flessibilità. Più precisamente, fornisce una piattaforma centralizzata per la gestione dei profili utente, dei gruppi e delle autorizzazioni di più sistemi, facilitando la gestione dell'accesso alle risorse e l'applicazione dei criteri di sicurezza desiderati (15). Utilizza inoltre protocolli e algoritmi di crittografia conformi agli standard del settore per proteggere le credenziali degli utenti e contrastare gli accessi non autorizzati. Per di più, FreeIPA è in grado di scalare per supportare ambienti distribuiti e di grandi dimensioni, rendendolo adatto a implementazioni di livello aziendale. In più, supporta un'ampia gamma di meccanismi

di autenticazione, tra cui Kerberos, LDAP e SAML, consentendo alle organizzazioni di scegliere il metodo più adatto alle proprie esigenze.

Nelle sottosezioni seguenti viene descritto come sia possibile configurare un server FreeIPA e come si possa integrare in Spring Boot per offrire un sistema SSO basato sul protocollo di autenticazione LDAP.

1.3.1 Configurazione del Server

Per configurare il server FreeIPA, esistono due opzioni: è possibile installarlo direttamente su un server dedicato, oppure utilizzare un'immagine Docker per eseguirlo in un ambiente containerizzato. L'esecuzione di FreeIPA in un container Docker offre diversi vantaggi rispetto all'esecuzione diretta su un server dedicato. In primo luogo, la sua containerizzazione consente di eseguire tutti i processi che lo compongono in modo isolato, lasciando l'host libero di eseguire altri software senza rischiare conflitti (16). In secondo luogo, utilizzando un container Docker, è possibile eseguire il server FreeIPA su un sistema operativo per il quale potrebbe non essere disponibile (come nel caso di Windows), rendendolo una soluzione più flessibile. Inoltre, Docker facilita la distribuzione e la gestione del server in questione, in quanto fornisce un ambiente coerente e riproducibile per la sua esecuzione.

Durante il primo avvio del container, viene avviato il processo di configurazione del server FreeIPA, il quale fornisce indicazioni per completare con successo quest'ultimo. Una volta completato l'iter di configurazione, è possibile verificare il funzionamento del server effettuando il login dal terminale del container

o accedendo all'interfaccia utente web fornita di default, utilizzando le credenziali specificate durante la creazione del server.

1.3.2 Integrazione con LDAP in Spring

L'autenticazione utente tramite il Lightweight Directory Access Protocol (LDAP) è un processo di convalida di una combinazione di nome utente e password presso un server di directory (FreeIPA in questo caso). Le directory LDAP sono una tecnologia standard per la memorizzazione di informazioni su utenti, gruppi e autorizzazioni e per la loro distribuzione in applicazioni aziendali (17).

Il processo di autenticazione degli utenti con una directory LDAP si svolge in due fasi. Il primo passo è la risoluzione del nome dell'utente in un attributo della voce della directory. Le voci degli utenti in una directory sono identificate da un nome distinto (DN) simile a una struttura gerarchica che parte dalla radice della directory (il segmento più a destra), come ad esempio “cn=luigi, ou=people, dc=polito, dc=it”. Il secondo passo consiste nel convalidare la password dell'utente. Le password sono verificate da un comando LDAP chiamato “bind”, in cui viene aperta una connessione al server della directory, a cui viene inviata una richiesta per autenticare la connessione come un particolare utente, passando il suo DN e la sua password. Se le credenziali sono corrette, il server restituirà un messaggio di successo, altrimenti restituirà un errore “Invalid credentials”.

In Spring Boot, è possibile configurare il server per utilizzare come metodo di autenticazione il protocollo LDAP, aggiungendo un provider di autenticazione alla catena dei filtri di sicurezza menzionata nelle sezioni precedenti. Questo provider è responsabile, dunque, dell'autenticazione degli utenti tramite

un server LDAP, come FreeIPA nel caso in esame. Inoltre, la catena di filtri di sicurezza deve anche essere configurata in modo tale da richiedere l'autenticazione per tutti gli endpoint esposti e abilitare il form di login in cui l'utente può inserire le proprie credenziali.

Il provider LDAP è fornito attraverso la definizione di un componente (bean) che implementa l'autenticazione vera e propria. Quest'ultimo è composto da due parti: un autenticatore che viene creato utilizzando una ricerca basata su specifici filtri per gli utenti presenti sul server LDAP, ad esempio utilizzando l'indirizzo e-mail come criterio di ricerca, e da un oggetto della classe DefaultSpringSecurityContextSource, il quale rappresenta la connessione al server LDAP, impostando l'URL, il DN e la password dell'amministratore per accedere al server e far sì che quest'ultimo autentichi le credenziali fornite dall'utente tramite il form di login. Una volta autenticato, il server inoltrerà, come menzionato in precedenza, un token di sicurezza al client, il quale si occuperà di memorizzarlo in modo sicuro e di inoltrarlo in tutte le richieste successive effettuate al server contenente le risorse protette.

1.4 Progettazione e Sviluppo del Frontend

Il frontend dell'applicazione è stato realizzato con React, una popolare libreria per la creazione d'interfacce utente e TypeScript, una versione estesa di JavaScript che aggiunge la tipizzazione statica opzionale e altre caratteristiche per aiutare gli sviluppatori a scrivere codice più robusto e mantenibile. Grazie alla tipizzazione statica, è possibile individuare gli errori nelle prime fasi dello sviluppo, riducendo il rischio di bug

e migliorando la qualità complessiva del codice. Inoltre, l'architettura modulare di React consente di scomporre interfacce utente complesse in componenti più piccoli e riutilizzabili, rendendo più facile la manutenzione e l'aggiornamento del codice nel tempo.

Nelle sottosezioni seguenti vengono illustrate alcune delle caratteristiche principali derivanti dall'uso di React e TypeScript per la realizzazione dell'interfaccia utente. Verranno trattati argomenti come l'architettura a componenti, la sicurezza, l'organizzazione del codice e la distribuzione dell'applicazione.

1.4.1 Architettura a Componenti

Alla base di un'applicazione React ben strutturata, c'è solitamente un componente centrale che si occupa del routing tra le varie viste, dove con routing si intende il processo di reindirizzamento di un utente a pagine diverse in base alla sua azione o richiesta (18). In genere, è presente una route separata per ogni vista, ma nulla vieta il contrario. Ai fini del servizio in discussione, le viste di riferimento sono: "Homepage", destinata a gestire le informazioni dei clienti in formato tabellare, "Client", dedicata alla gestione dei dettagli di un cliente, e "LicenseFile", che invece mostra i dettagli di un file di licenze, come le licenze di cui è composto e la sua scadenza. Ovviamente ogni vista fornisce opportuni controlli agli utenti per interagire ed eseguire varie azioni, come la creazione, la cancellazione e l'aggiornamento delle informazioni dei clienti, e similmente per i file di licenze, con la possibilità aggiuntiva di poterne effettuare il download.

In React, ogni vista è costituita da componenti riutilizzabili che possono essere condivisi in tutta l'applicazione.

Questi componenti sono essenzialmente blocchi di costruzione che rendono più facile lo sviluppo d'interfacce utente complesse. Quando si lavora con i componenti, è comune passare dati da un componente genitore a uno figlio usando le props (acronimo di proprietà), le quali permettono di definire i parametri che un componente si aspetta di ricevere.

Uno dei principali vantaggi dell'uso di TypeScript risiede proprio nella possibilità di garantire un corretto e sicuro utilizzo di un componente, fornendo annotazioni di tipo per le proprietà passate a esso. Specificando il tipo di ogni proprietà tramite la parola chiave 'type' (come mostrato in Figura 5), è possibile individuare gli errori direttamente durante lo sviluppo, garantendo una maggiore affidabilità del codice. Questo non solo fa risparmiare tempo in fase di debug, ma aiuta anche a prevenire potenziali errori di runtime che potrebbero verificarsi se i tipi di dati non corretti fossero passati come proprietà.

```tsx
type ButtonProps = {
  text: string;
  onClick: () => void;
  primary?: boolean;
};

const Button = ({ text, onClick, primary = false }: ButtonProps) => {
  const buttonStyle = primary ? "primary" : "secondary";
  return (
    <button className={`button ${buttonStyle}`} onClick={onClick}>
      {text}
    </button>
  );
};
```

Figura 5: Esempio di un componente React tipizzato

Pertanto, combinando l'uso di componenti riutilizzabili con le proprietà tipizzate di TypeScript, è possibile creare una libreria di componenti non solo efficiente e coerente, ma anche affidabile e facile da usare.

Oltre ai componenti riutilizzabili, in un'architettura React TypeScript ben strutturata sono coinvolti altri due importanti elementi: i tipi e i servizi. I tipi sono interfacce, enumerativi e altri costrutti forniti da TypeScript in grado di definire la struttura e i tipi di dati gestiti all'interno dell'applicazione. Utilizzando i tipi, è possibile garantire una maggiore robustezza e facilità di gestione del codice, in quanto eventuali errori o incoerenze nei dati possono essere rilevate durante lo sviluppo, come descritto in precedenza per le proprietà dei componenti. Ad esempio, si possono utilizzare le interfacce per rappresentare il corpo delle richieste e delle risposte HTTP, garantendo l'invio e la ricezione dei tipi di dati corretti.

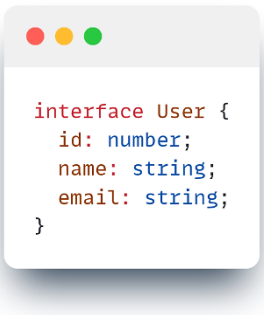
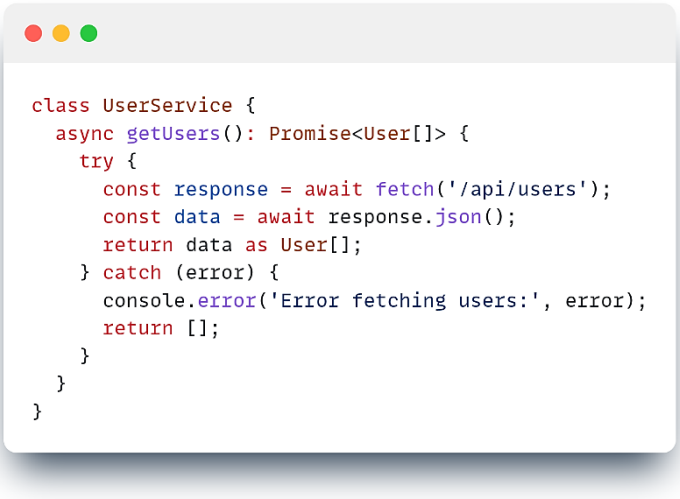


Figura 6: Esempio di interfaccia in TypeScript

I servizi, invece, sono classi in cui viene inserita la logica relativa alle chiamate HTTP e ad altre operazioni remote. Separando questa logica dal resto, è possibile mantenere il codice dell'applicazione organizzato e più modulare. Ad esempio,

è possibile creare una classe UserService che gestisce le richieste API relative ai dati degli utenti. Questa classe può contenere metodi che effettuano richieste all'API del backend e ne gestiscono le risposte, le quali saranno poi utilizzate all'interno di una vista per la loro visualizzazione.



```
class UserService {
  async getUsers(): Promise<User[]> {
    try {
      const response = await fetch('/api/users');
      const data = await response.json();
      return data as User[];
    } catch (error) {
      console.error('Error fetching users:', error);
      return [];
    }
  }
}
```

Figura 7: Esempio di una classe di servizio in TypeScript

Nel complesso, l'architettura React TypeScript descritta consente di separare le attività da svolgere in componenti, tipi e servizi riutilizzabili e di creare una struttura applicativa più flessibile e adattabile.

1.4.2 Gestione delle Chiamate API con Axios

Quando si costruisce un'applicazione React è importante effettuare le chiamate API in modo semplificato per ridurre la complessità del codice e facilitarne la manutenzione. Questa esigenza nasce dal fatto che con ogni probabilità verranno effettuate molte chiamate API per recuperare dati da un server (backend) o aggiornarli. In assenza di un modo semplificato

per gestire queste chiamate, il codice dell'applicazione diventerebbe rapidamente “boilerplate”, ovvero denso di codice ripetitivo. Un ottimo strumento per evitare questi problemi è Axios, un popolare client HTTP che fornisce un'API di semplice utilizzo per effettuare richieste agli API endpoint del backend. Tuttavia, Axios non è solo un semplice client HTTP, poiché offre anche soluzioni avanzate, come per la gestione degli errori e per la modifica automatica delle richieste prima del loro effettivo invio. Ad esempio, è possibile utilizzare gli Axios Interceptor per intercettare tutte le richieste e le risposte HTTP e gestire gli errori in modo centralizzato. Inoltre, gli intercettatori possono anche essere sfruttati per fornire un meccanismo di aggiunta automatica dei token CSRF a ogni richiesta protetta, come POST, PATCH e DELETE, senza dover aggiungere manualmente il token a ogni richiesta. Questo è importante per ragioni di sicurezza, in quanto aiuta a prevenire gli attacchi Cross-Site Request Forgery, come discusso in precedenza in questo elaborato.

Per sfruttare al meglio Axios in un'applicazione React, è opportuno creare una configurazione centralizzata che definisca tutte le informazioni necessarie per effettuare le richieste, compresi i tipi di contenuto (ad esempio, form-data, json, ecc.), le intestazioni (come per il token CSRF) e altri dettagli necessari. Questa configurazione può essere definita in un unico punto e poi utilizzata in tutta l'applicazione, principalmente nelle classi di servizio. Come già menzionato nella sezione precedente, sfruttare queste classi è un modo eccellente per incapsulare la logica delle chiamate API e gestire tutti i dettagli delle richieste con Axios in modo più organizzato e gestibile.

1.4.3 Virtualizzazione delle Tabelle: React Virtual/Table

Essendo uno dei metodi più diffusi per organizzare informazioni complesse, le tabelle sono spesso utilizzate nei prodotti web. Tuttavia, costruire una tabella da zero potrebbe essere un'impresa ardua e in particolare React è noto per dare filo da torcere agli sviluppatori nella sua creazione. Fortunatamente, è disponibile un'ampia gamma di strumenti e librerie che rendono l'esperienza della creazione di una tabella in React decisamente più semplice e conveniente, in particolare TanStack Table, anche noto come React Table (19). Combinando questa libreria con React Virtual, la quale consente di creare componenti virtualizzati, è possibile costruire tabelle di dati in modo efficiente e personalizzabile.

React Virtual impiega la virtualizzazione, ovvero una tecnica che consente di visualizzare solo gli elementi attualmente visibili sullo schermo, invece di mostrare l'intero elenco. Per implementare React Table insieme a React Virtual, è possibile utilizzare l'hook useVirtual fornito da quest'ultimo. Tale hook consente di creare un contenitore virtualizzato per le righe della tabella, il quale può poi essere passato al componente Table di React Table. Ogni riga della tabella sarà contenuta nel componente VirtualItem di React Virtual, in modo da essere visualizzata solo quando è effettivamente visibile sullo schermo. Questo può migliorare notevolmente le prestazioni della tabella, soprattutto quando si tratta di grandi moli di dati. Ad esempio, quando si esegue il rendering di una lista di 10.000 elementi, la virtualizzazione può ridurre il numero di elementi del DOM da 10.000 a circa 40, con un miglioramento di 250 volte delle prestazioni.

Inoltre, è possibile sfruttare le caratteristiche di TypeScript per creare una tabella generica che può poi essere utilizzata con qualsiasi tipo di dato (o quasi). Infatti, TypeScript consente di definire i tipi per le proprietà e gli stati di un componente React, permettendo così di creare componenti che possono essere riutilizzati in tutta l'applicazione anche con tipi di dati eterogenei e fornendo eventualmente modi per personalizzare il loro comportamento e la loro presentazione, rendendo il codice più flessibile e più facile da mantenere.

1.4.4 Utilizzo di Hooks Avanzati e Personalizzati

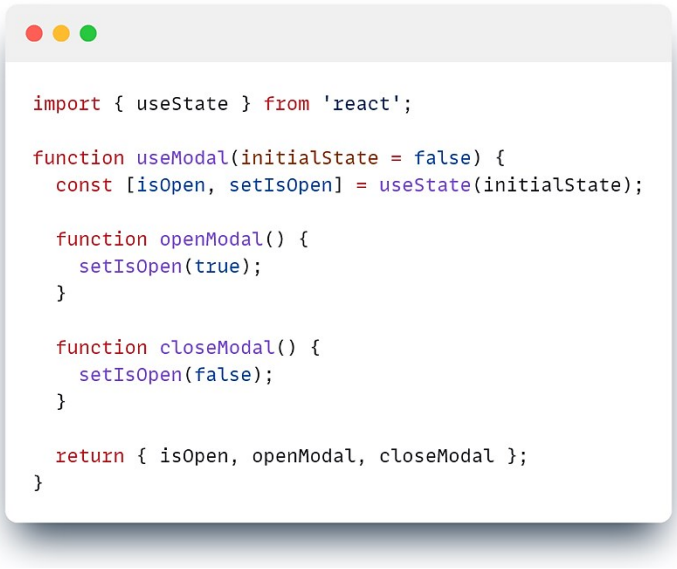
In React, gli hooks sono un potente strumento per gestire lo stato e il comportamento dei componenti. Inoltre, gli hooks possono essere impiegati per riutilizzare la logica di stato in più componenti e possono rendere il codice più modulare, più facile da leggere e meno soggetto a errori. In React sono già disponibili molti hooks pronti per l'utilizzo, tra cui due particolarmente utili per migliorare le prestazioni dell'applicazione: useMemo e useCallback.
Il primo consente di memorizzare il risultato di un calcolo, in modo da doverlo rielaborare solo quando le sue dipendenze vengono modificate (20). Questo può essere utile specialmente per calcoli onerosi o quando si ha a che fare con grande quantità di dati. useCallback è simile a useMemo, ma il suo utilizzo è specifico per la memorizzazione di una funzione, ovvero fare in modo che questa debba essere ricreata solo quando le sue dipendenze subiscono modifiche. Questo può essere utile per ottimizzare le prestazioni quando si passano funzioni tramite props a componenti figli.

In Figura 8 è mostrato un possibile utilizzo dei due hooks appena descritti. In particolare, l'esempio illustra come prevenire inutili re-rendering e di dover ripetere calcoli costosi inutilmente. Il componente in figura utilizza useMemo per memorizzare il risultato di una computazione dispendiosa, assicurando che il calcolo venga eseguito solo quando il parametro necessario cambia e non a ogni rendering del componente. Inoltre, utilizza useCallback in modo da garantire che la relativa funzione memorizzata non venga ricreata a ogni rendering del componente genitore e di conseguenza che non causi il re-rendering (non necessario) anche del componente figlio, il quale ottiene tale funzione tramite props.

```jsx
import React, { useState, useMemo, useCallback } from 'react';

const Parent = () => {
  const [count, setCount] = useState(0);

  const handleClick = useCallback(() => {
    setCount((prevCount) => prevCount + 1);
  }, []);

  const expensiveCalculation = useMemo(() => {
    // Performing expensive calculation...
    return result;
  }, [count]);

  return (
    <div>
      <p>Expensive calculation result: {expensiveCalculation}</p>
      <Child onClick={handleClick} />
    </div>
  );
};

export default Parent;
```

Figura 8: Esempio di utilizzo degli hooks useMemo e useCallback

Oltre agli hooks esistenti, è possibile crearne altri personalizzati per semplificare la logica dell'applicazione. Infatti, è possibile utilizzare hooks per incapsulare parti di logica complessa o per astrarre attività ripetitive, rendendo il codice più leggibile e modulare. Ad esempio, si può creare un hook personalizzato per gestire la validazione dei form o per gestire lo stato di una finestra di dialogo (come mostrato in Figura 9).



```
import { useState } from 'react';

function useModal(initialState = false) {
  const [isOpen, setIsOpen] = useState(initialState);

  function openModal() {
    setIsOpen(true);
  }

  function closeModal() {
    setIsOpen(false);
  }

  return { isOpen, openModal, closeModal };
}
```

Figura 9: Esempio di hook personalizzato

1.4.5 Rischi per la Sicurezza Web: Linee Guida OWASP

Le vulnerabilità lato client possono rappresentare un rischio significativo per le applicazioni nonché esporre informazioni sensibili agli attaccanti. Per affrontare questi rischi, è importante seguire le migliori pratiche di sicurezza, come quelle delineate nella OWASP Top 10. Questa è un elenco dei rischi di sicurezza più critici nonché diffusi per le applicazioni web e fornisce indicazioni su come mitigarli.

Una delle vulnerabilità lato client più comuni è l'XSS, che si colloca infatti al terzo posto della OWASP Top 10 all'interno della categoria più generale "Injection" e in particolare viene dichiarato che il 94% delle applicazioni è stato testato per una qualche forma d'injection (7). Per affrontare questa categoria di rischi, è necessario implementare funzioni di convalida e sanificazione dei dati in ingresso forniti dagli utenti per garantire che essi siano filtrati correttamente per evitare l'esecuzione o l'invio di script e richieste potenzialmente dannosi. Si tratta di un ulteriore livello di sicurezza in aggiunta alla già esaminata protezione offerta dal backend, la quale viene ugualmente applicata per mitigare ulteriormente il rischio di attacchi XSS.

Un'altra vulnerabilità molto diffusa è il CSRF che, sebbene non sia attualmente incluso come categoria distinta nell'ultimo elenco OWASP Top 10 del 2021, è tuttora considerato un rischio significativo per la sicurezza delle applicazioni web ed è comunque menzionato nel testo principale del rapporto. Per prevenire gli attacchi CSRF, un token casuale viene generato dal server e inviato al frontend durante il caricamento iniziale dell'applicazione. Quando viene inviata al server una richiesta a una risorsa protetta, il valore del token viene inviato tramite un'intestazione HTTP dedicata (X-XSRF-TOKEN) e poi confrontato con la versione memorizzata sul server. Se i token corrispondono, la richiesta viene considerata valida e il server la elabora; in caso contrario, essa viene rifiutata. Questo approccio garantisce che solo gli utenti legittimi con un token valido possano accedere alle risorse protette esposte dal backend.

Un'altra vulnerabilità comune è quella dell'autenticazione compromessa che, nel caso in discussione, viene affrontata tramite l'utilizzo di un sistema di autenticazione SSO (FreeIPA), descritto nella sezione precedente. Questa categoria è al settimo posto della OWASP Top 10 ed è fondamentale per garantire che solo gli utenti autorizzati possano accedere alle risorse protette. Il backend gestisce il processo di autenticazione, tramite l'utilizzo di FreeIPA, e reindirizza l'utente al percorso principale dell'applicazione una volta che è stato autenticato. Inoltre, il processo in questione fornisce un token di autenticazione che viene poi allegato, tramite un cookie "HttpOnly", a ogni richiesta effettuata al backend, garantendo così che tutti i percorsi del frontend siano protetti e che non sia consentito alcun accesso non autorizzato.

In generale, la OWASP Top 10 fornisce indicazioni su come mitigare i rischi di sicurezza più critici per le applicazioni web, ed è importante tenerne conto durante lo sviluppo di quest'ultime. Questo rapporto è visto come il primo passo verso una programmazione più sicura e verso la trasformazione della cultura di sviluppo del software all'interno di un'organizzazione in una che produca codice più protetto (7).

1.4.6 Ottimizzare Applicazioni React con Webpack

Quando si sviluppa un'applicazione React, è molto comune usare Webpack come bundler, ovvero come uno strumento in grado di elaborare l'applicazione, costruire internamente un diagramma delle dipendenze da uno o più punti d'ingresso e successivamente combinare ogni modulo del progetto in uno o più bundle,

ossia risorse statiche da cui servire il contenuto (21). Uno dei principali vantaggi dell'uso di Webpack è infatti la sua capacità di raggruppare tutto il codice e le dipendenze di un'applicazione in un unico o in un insieme di file. Ciò consente di velocizzare i tempi di caricamento e di ridurre il numero di richieste di rete necessarie per caricare l'applicazione. Webpack offre anche una serie di plugin che possono contribuire a ottimizzare le prestazioni dell'applicazione, a ridurne le dimensioni e a migliorarne la sicurezza. I plugin sono moduli che possono essere aggiunti alla configurazione di Webpack per eseguire compiti specifici durante il processo di bundling. Tuttavia, l'ottimizzazione di un'applicazione web comporta la configurazione di modalità diverse per lo sviluppo e la produzione, ognuna con il proprio insieme di caratteristiche e necessità.

In modalità di sviluppo, vengono attivate solo alcune funzionalità, come il ricaricamento “a caldo” dei moduli, l'abilitazione del server di sviluppo e l'opzione 'historyApiFallback'. Il primo è una funzione con la quale è possibile visualizzare le modifiche apportate senza dover ricaricare l'intera applicazione. Quando viene apportata una modifica al codice, il server di sviluppo inietterà automaticamente il modulo aggiornato nell'applicazione in esecuzione, consentendo di vedere immediatamente le modifiche senza dover aggiornare la pagina. Questo può far risparmiare molto tempo durante lo sviluppo e può aiutare gli sviluppatori a individuare più rapidamente gli errori. L'opzione "historyApiFallback" è invece usata per abilitare il routing lato client. Quando un utente naviga verso un URL che non corrisponde a nessuna delle rotte definite nell'applicazione, l'opzione "historyApiFallback" reindirizza l'utente

a una pagina di fallback. Questo è utile per le applicazioni che utilizzano il routing lato client e garantisce che gli utenti vedano sempre una pagina valida, anche se navigano verso un URL non esistente.

In modalità di produzione, invece, vengono attivati anche altri plugin e ottimizzazioni per ridurre le dimensioni dei file TypeScript e CSS, aumentare le prestazioni e offuscare il codice. Per ottenere queste ottimizzazioni, vengono adottate diverse specifiche configurazioni e plugins. Nello specifico, il plugin “TerserWebpackPlugin” è utilizzato per la minificazione del codice ed è in grado di rimuovere i commenti e comprimere il codice in modalità di produzione. Con il plugin “WebpackObfuscator” si può invece ottenere l'offuscamento del codice, offrendo numerose opzioni per trasformarlo in una forma illeggibile in modo da offrire una protezione da parte di malintenzionati che potrebbero manipolarlo o emularlo al fine d’impadronirsene. Il plugin “HtmlWebpackPlugin” è utilizzato per la generazione automatica di un file HTML in cui inserire i vari file JavaScript prodotti dalla compilazione, facilitando la gestione dell'inclusione di script e fogli di stile (CSS). I plugin "ProvidePlugin" e "CleanWebpackPlugin" vengono invece utilizzati per rimuovere il codice inutilizzato dalla compilazione finale, importando automaticamente i moduli quando utilizzati nel codice e ripulendo la cartella di output prima di ogni compilazione. Infine, il plugin "SplitChunks" è usato per creare bundle separati per il codice delle dipendenze, in modo da poterlo caricare separatamente per velocizzare i tempi di caricamento.

Utilizzando questi plugins e ottimizzando il processo di compilazione, si può garantire una migliore esperienza utente, tempi di caricamento più rapidi e una maggiore sicurezza per l'applicazione sviluppata.

1.4.7 Distribuzione dell'Applicazione

Per la distribuzione di un'applicazione web sono disponibili diverse opzioni, come la distribuzione tradizionale via server web dedicato, quella basata su cloud e quella tramite container (Docker). Una possibilità che offre diversi vantaggi è quella di offrire il frontend come risorsa statica tramite Spring Boot e poi utilizzare un sistema di distribuzione basato su container per facilitarne l'utilizzo in ambienti diversi senza problemi di compatibilità. Utilizzando Spring è infatti possibile fornire risorse statiche, come il bundle del frontend generato tramite Webpack, senza la necessità di avere un server web separato per gestirle. Questo semplifica di molto la distribuzione e la gestione dell'applicazione, poiché il backend e il frontend possono essere distribuiti insieme e gestiti come un'unica entità.

Per attuare questa configurazione, occorre innanzitutto specificare il percorso delle risorse statiche all'interno del file di configurazione dell'applicazione del backend. Questo indica a Spring dove cercare queste risorse per poi renderle disponibili per la loro fruizione. Inoltre, è necessario definire un controller per gestire le richieste a queste risorse. In questo modo si assicura che quest'ultime siano tra l'altro autenticate e autorizzate in modo appropriato, poiché Spring applicherà a esse qualsiasi aspetto trasversale, come la sicurezza, originariamente configurato per il backend. Questo fornisce un ulteriore livello di protezione per l'applicazione, in quanto le stesse misure di sicurezza vengono

applicate alle richieste sia del backend che del frontend in automatico, dato che ora condividono lo stesso ambiente.

Oltre a offrire il frontend come risorsa statica, la configurazione di un ambiente di produzione per la distribuzione tramite Docker è in grado di offrire ulteriori vantaggi. Infatti, Docker consente di creare ambienti containerizzati che possono essere facilmente replicati su diversi sistemi, anche eterogenei, garantendo una distribuzione coerente e riducendo il rischio di errori di configurazione. Tra l'altro, Docker fornisce uno strumento chiamato Docker Compose per la creazione di file di configurazione che permettono di distribuire più container come un'unica entità. Si tratta infatti di uno strumento con il quale è possibile definire ed eseguire applicazioni Docker multi-container semplicemente definendo all'interno di un singolo file tutti i servizi dell'applicazione, con relative configurazioni. Questo strumento permette inoltre di creare e avviare, con un solo comando, tutti i servizi presenti nella configurazione (22). In altri termini, l'intera applicazione, compresi il backend, il frontend, il database, il server di autenticazione e qualsiasi altro servizio necessario, può essere resa disponibile in modo semplificato attraverso la definizione di un singolo file di configurazione, rendendo la distribuzione e la gestione di un'applicazione web più semplice ed efficiente.

1.5 Conclusioni

In questo capitolo si è visto come sia possibile creare un servizio in grado di generare file di licenze in modo sicuro ed efficiente. La creazione di un'applicazione containerizzata composta da tre

servizi, in particolare da MongoDB per la gestione dei dati, FreeIPA per l'autenticazione SSO degli utenti e un'applicazione Java Spring Boot con un frontend React TypeScript offerto come risorsa statica, fornisce una solida base per la costruzione di un'applicazione web sicura, in quanto offrono tutti gli strumenti necessari per creare un sistema robusto, scalabile e sicuro.

In particolare, in questo capitolo è stata data particolare enfasi a come poter mitigare e prevenire i principali rischi e vulnerabilità comunemente diffusi nelle applicazioni web, tra cui quelle descritte nel rapporto OWASP Top 10 del 2021, che pone particolare attenzione ai problemi legati agli attacchi CSRF, XSS, al problema delle comunicazioni insicure (risoltesi attraverso l'uso di HTTPS), a quello delle injection e alla compromissione dell'autenticazione.

In aggiunta a ciò, si è visto come la crittografia rappresenti un ottimo strumento di protezione in grado di offrire integrità e riservatezza dei dati, nella fattispecie per i file di licenze. Per di più esso rappresenta uno strumento facilmente integrabile all'interno di un framework molto versatile come quello offerto da Java Spring Boot, che insieme a tutte le librerie di cui è composto, offre la possibilità di gestire ogni singola esigenza all'interno della propria applicazione web.

Anche per la parte relativa al frontend sono stati analizzati diversi aspetti, a partire dalla progettazione di un'architettura modulare in grado di rendere l’applicazione più fluida e facile da gestire, all'introduzione di protezioni di sicurezza, al miglioramento delle prestazioni con l'uso di hooks personalizzati e no, all’utilizzo di specifiche librerie per la virtualizzazione

(React Virtual/Table) e per semplificare la gestione delle chiamate HTTP (Axios), fino alla sua produzione e distribuzione come risorsa statica tramite Spring Boot.

Il capitolo sottolinea inoltre l'importanza della distribuzione tramite container e fornisce nozioni pratiche su come distribuire un'applicazione containerizzata in modo appropriato. Questo aspetto è particolarmente importante visto il crescente utilizzo della containerizzazione e dei microservizi nello sviluppo di applicazioni web moderne.

In sintesi, questo capitolo fornisce preziose indicazioni sullo sviluppo e sulla distribuzione di un'applicazione web sicura. Seguendo le migliori pratiche e le linee guida delineate nel capitolo, è possibile realizzare applicazioni web robuste ed efficienti, in grado di rispondere alle esigenze del settore.

Capitolo 2

Auditing System

Questo capitolo espone il processo seguito per progettare e implementare il servizio di consultazione ed esportazione delle informazioni di auditing prodotte da un sistema all'interno di un contesto investigativo. Il capitolo inizia con una discussione delle principali possibilità disponibili per aumentare le prestazioni relative all'utilizzo di MongoDB e della protezione dei suoi dati, compresa l'integrazione con Spring Security per la gestione dell'autenticazione e dell'autorizzazione degli utenti. Infine, si analizza il modo in cui i documenti PDF possono essere prodotti in modo asincrono in Java. Sarà inoltre dimostrato come si possa applicare una firma digitale ai PDF, per garantirne la sicurezza e la conformità agli standard legali e normativi.

2.1 Introduzione

L'Auditing System è un servizio in cui gli utenti possono effettuare diverse operazioni in base ai permessi loro assegnati o in base al gruppo di appartenenza. Le operazioni possibili sono essenzialmente tre: ottenere una lista paginata di entry di audit eventualmente filtrate e/o ordinate, programmare un'operazione di esportazione delle entry di audit risultanti dall'operazione precedente e ottenere il documento PDF generato al termine di quest'ultima.

Lo scopo e gli obiettivi di questo servizio sono diversi da quello discusso nel primo capitolo. Infatti, in questo caso, si tratta di un applicativo utilizzato direttamente dal cliente finale, a differenza del Licensing Generator utilizzato internamente. Questo ovviamente comporta requisiti di funzionamento leggermente diversi, anche se rimangono assolutamente valide e da impiegare tutte le linee guida e le buone pratiche per la realizzazione di un'applicazione web discusse nel primo capitolo, come l'impiego del pattern architetturale MVC, le protezioni contro gli attacchi più diffusi, come CSRF e injections, la gestione dei logs, l'uso di canali di comunicazione sicuri, ecc. Le quali non saranno ovviamente ritrattate all'interno di questo capitolo. Lo stesso vale per la parte di distribuzione del servizio, la quale avverrà sostanzialmente nello stesso modo della prima, cioè tramite containers Docker. Tuttavia, è importante notare come in questo caso il servizio consista solo in un backend Java Spring Boot senza alcuna interfaccia grafica.

Nel complesso, il sistema gestisce le entry di audit prodotte da un sistema legato a un ambito investigativo. In molti contesti

legali e normativi, come quello in essere, i record di audit sono considerati prove legali e possono essere utilizzate nei procedimenti giudiziari. Questo perché le entry di audit forniscono una descrizione dettagliata e completa di tutte le azioni intraprese da un sistema o da un individuo, compresa la data, l'ora e la natura dell'azione. Ad esempio, in una indagine giudiziaria, i record di audit possono essere utilizzati per tracciare i movimenti dei sospetti, raccogliere prove di attività illegali e identificare potenziali testimoni.

Questo capitolo, rispetto al precedente, si concentrerà principalmente sulle modalità di autorizzazione delle richieste e sull'ottimizzazione delle prestazioni di un sistema chiamato a gestire elevati volumi di dati, visto che i record di audit raggiungono i 10 milioni al giorno, nonché sulla produzione di documenti in formato PDF che offrano autenticità, integrità e validità dei dati contenuti.

2.2 Spring Data MongoDB: Utilizzo Avanzato

Come accennato nell'introduzione, il servizio deve gestire una notevole quantità d'informazioni e, in combinazione con l'esigenza di ottenere elevate prestazioni, ne consegue la necessità di sfruttare specifiche funzionalità offerte sia direttamente da MongoDB e sia da Spring Data per perseguire tali obiettivi. In particolare, il requisito di ottenere prestazioni elevate è necessario per l'operazione di recupero delle informazioni di audit paginate, in cui l'utente autorizzato ha anche la possibilità di applicare vari filtri e di ordinare i risultati con diversi livelli di ordinamento.

Nelle sottosezioni che seguono, verranno illustrate le modalità con cui è possibile soddisfare le suddette richieste, mantenendo al contempo prestazioni elevate anche in presenza di un numero considerevole di dati.

2.2.1 Ottimizzazione delle Prestazioni

Quando si tratta di migliorare le prestazioni relative all'uso di un database, una metrica comune è quella di ottenere i risultati delle interrogazioni in meno di un secondo. Raggiungere questo livello elevato di prestazioni può essere impegnativo, ma ci sono alcune tecniche in grado di fornire un aiuto.

Uno dei modi più efficaci per migliorare le prestazioni in MongoDB è l'indicizzazione. Gli indici consentono di trovare e recuperare rapidamente i dati, invece di dover scansionare l'intero database. Creando indici sui campi interrogati regolarmente, è possibile ridurre significativamente i tempi di risposta e migliorare le prestazioni complessive. Ad esempio, si supponga di avere una collezione di dati relativi agli utenti e di eseguire spesso interrogazioni in base al tipo di utente. Creando un indice su quest'ultimo campo, è possibile ridurre il tempo d'interrogazione da diversi secondi a millisecondi. In alcuni casi, la creazione di un indice può portare a un miglioramento delle prestazioni da 10 a 100 volte.

Gli indici possono essere creati direttamente tramite annotazioni specifiche fornite da Spring Data MongoDB, ma in genere non è una soluzione consigliata. Il motivo è che la creazione d'indici è un'attività di configurazione una tantum e non dovrebbe essere eseguita potenzialmente a ogni avvio dell'applicazione. Gli indici dovrebbero invece essere creati direttamente

in MongoDB, utilizzando gli strumenti appropriati. Fortunatamente, questo non costituisce un problema per quanto riguarda la distribuzione del servizio, poiché Docker Compose consente di specificare un meccanismo di migrazione con la quale poter inizializzare il database. Definendo uno specifico Docker entrypoint come volume, è possibile indicare alcuni script JavaScript da eseguire quando il container di MongoDB viene inizializzato. Tali script possono contenere, ad esempio, il codice necessario per la creazione di collezioni, l'aggiunta d'indici ed eventuali dati predefiniti. Questo assicura che il processo di inizializzazione venga effettuato solo una volta, al momento della creazione, e non a ogni avvio dell'applicazione.

Uno dei problemi principali dell'indicizzazione è rappresentato dal fatto che essa può aumentare i requisiti di archiviazione e incidere sulle prestazioni di scrittura, in particolare quando si tratta di volumi elevati di dati (23). Difatti, gli indici devono essere aggiornati ogni qualvolta un documento viene aggiunto, aggiornato o cancellato, il che può comportare operazioni di scrittura aggiuntive e un aumento della latenza di scrittura. Inoltre, non tutte le query possono beneficiare dell'indicizzazione. Alcune query potrebbero anche non utilizzare alcun indice, il che significa che l'overhead del mantenimento del medesimo non è necessario e può anzi peggiorare le prestazioni. Pertanto, è importante essere strategici nella creazione degli indici e crearli solo dove necessario. Questo significa dover analizzare le query da effettuare e i log, identificare i campi a cui si accede di frequente e creare indici solo su di essi (24).

Oltre all'indicizzazione, un'altra tecnica da considerare è l'uso di documenti incorporati o referenziati. Il primo permette di evitare l'utilizzo di join, riducendo al minimo le query e gli aggiornamenti effettuati, ma può causare altri tipi di problemi come, ad esempio, la duplicazione delle informazioni. Si dovrebbe prendere in considerazione la referenziazione quando un documento viene consultato di frequente ma contiene dati usati raramente. L'inclusione in questo caso aumenterebbe solo i requisiti di memoria; quindi, il riferimento potrebbe essere più sensato (24). Inoltre, se una parte di un documento viene aggiornata di frequente e continua ad allungarsi, mentre il resto del documento è relativamente statico, il riferimento può aiutare a evitare di aggiornare ogni volta l'intero documento.

2.2.2 Mongo Template

MongoTemplate è una classe fornita da Spring Data in grado di fornire un'API di alto livello per interagire con MongoDB. È un'alternativa più flessibile e personalizzabile rispetto all'implementazione predefinita di un repository, il quale utilizza un insieme predefinito di operazioni CRUD. Uno dei principali vantaggi dell'utilizzo di MongoTemplate è la possibilità di creare repository personalizzati che offrono la possibilità d'implementare requisiti speciali che potrebbero essere difficili o impossibili da ottenere con un normale Mongo repository.

Infatti, rispetto a quest'ultimo, MongoTemplate offre maggiore flessibilità e controllo sul processo d'interrogazione. Con MongoTemplate è possibile scrivere query personalizzate utilizzando una serie di criteri, tra cui restrizioni, proiezioni e opzioni di ordinamento. Tutto ciò può essere particolarmente utile quando

si ha a che fare con parametri variabili o scenari di query complesse, come nel caso in questione, dove un insieme predefinito di metodi di repository potrebbe non essere sufficiente. Un altro vantaggio dell'uso di MongoTemplate è la possibilità di ottenere prestazioni migliori per alcune tipologie di query. Ad esempio, se si devono eseguire specifiche operazioni di aggregazione o di paginazione, MongoTemplate può essere un'opzione più efficiente rispetto all'implementazione predefinita del repository.

D'altra parte, MongoTemplate risulta essere più complesso e richiedere una maggiore quantità di codice rispetto all'utilizzo dei classici repository. Inoltre, esso potrebbe non essere sicuro dal punto di vista dei tipi, poiché lavora con documenti grezzi con la necessità di convertirli manualmente in oggetti Java, il che potrebbe portare a errori se non eseguito correttamente.

Pertanto, il ricorso a MongoTemplate deve essere ben ponderato prima del suo effettivo utilizzo, poiché, sebbene sia molto potente e fornisca un grande controllo sulle query effettuate, comporta una maggiore responsabilità e gestione per il programmatore con il conseguente rischio d'incorrere in errori o bug.

2.2.3 Filtraggio, Ordinamento e Paginazione

Come descritto nella sottosezione precedente, MongoTemplate può essere impiegato per l'implementazione di un repository personalizzato, che nel servizio in esame è particolarmente adatto alla realizzazione dell'operazione di ottenimento dell'elenco paginato delle entry di audit, in quanto per essa gli utenti possono specificare diverse tipologie di filtri, più livelli di ordinamento

e indicare il numero di elementi per pagina, nonché quale pagina in particolare ottenere.

Con MongoTemplate, è possibile creare una query personalizzata utilizzando la classe Query, la quale rappresenta una query MongoDB che può poi essere eseguita su una collezione. La classe Query consente di aggiungere un elenco di criteri per filtrare i risultati in base a condizioni specifiche. I criteri sono rappresentati dalla classe Criteria, con la quale è possibile specificare le condizioni che devono essere soddisfatte affinché un documento sia incluso nei risultati della query. I criteri possono essere combinati utilizzando vari operatori, come "and", "or", "not", "in", "nin" e molti altri. Questi operatori possono essere facilmente combinati per creare query complesse che filtrano i risultati in base a più condizioni. Si noti come questa capacità d'inserire parametri in modo condizionale e di creare query complesse non sia possibile con l'implementazione predefinita offerta da Spring Data MongoDB, la quale richiede un insieme fisso di parametri.

L'ordinamento può essere effettuato utilizzando la classe Sort, la quale consente di definire uno o più criteri di ordinamento. Ogni criterio di ordinamento è definito utilizzando un nome di un campo e una direzione (ascendente o discendente). Ad esempio, è possibile ordinare il campo "nome" in ordine decrescente creando un oggetto Sort con i parametri "nome" e "DESC".

Infine, la paginazione può essere ottenuta utilizzando l'interfaccia Pageable, con la quale è possibile specificare con precisione il numero di pagina richiesto e il massimo numero di elementi

presenti in ciascuna. Si può quindi utilizzare il metodo find di MongoTemplate per concatenare le query applicando i criteri di paginazione e ordinamento, indicando il tipo di classe dei documenti ottenuti. In particolare, il metodo find restituisce un oggetto generico della classe Page, che rappresenta una pagina di risultati della query eseguita e contiene gli elementi della pagina richiesta, il numero di pagina corrente, il numero totale di pagine e il numero totale di elementi.

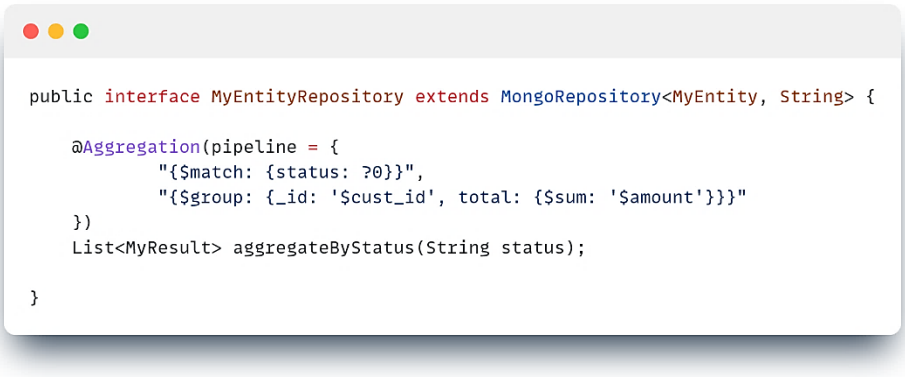
2.2.4 Query di Aggregazione

Il sistema di aggregazione di MongoDB è progettato per raggruppare documenti e trasformarli in un risultato aggregato. La query di aggregazione consiste nella definizione di diverse fasi che vengono poi eseguite in sequenza (25). Spring Data offre una buona integrazione con il framework di aggregazione di MongoDB, tramite l'annotazione “Aggregation”.

Questa annotazione può essere aggiunta a un metodo di una qualsiasi interfaccia “MongoRepository” e consente di definire le fasi della pipeline utilizzando la classica sintassi di aggregazione definita da MongoDB. Ad esempio, è possibile definire una pipeline a due fasi, come mostrato nella Figura 10, in cui nella prima i documenti vengono filtrati e nella seconda i risultati vengono raggruppati in base a un determinato campo, eseguendo al contempo un calcolo relativo al gruppo.

L'uso dell'annotazione Aggregation presenta diversi vantaggi. In primo luogo, consente di definire query di aggregazione utilizzando una sintassi e un'API familiare, riducendo la curva di apprendimento per i nuovi sviluppatori. In secondo luogo,

consente l'uso di query sicure dal punto di vista del tipo, il che può aiutare a individuare gli errori in fase di compilazione piuttosto che in fase di esecuzione. Infine, può migliorare le prestazioni dell'applicazione riducendo la quantità di dati recuperati dal database e sfruttando le capacità di aggregazione native di MongoDB.



```
public interface MyEntityRepository extends MongoRepository<MyEntity, String> {

    @Aggregation(pipeline = {
            "{$match: {status: ?0}}",
            "{$group: {_id: '$cust_id', total: {$sum: '$amount'}}}"
    })
    List<MyResult> aggregateByStatus(String status);

}
```

Figura 10: Esempio di pipeline di aggregazione in Spring Boot

Tuttavia, è importante progettare attentamente la pipeline di aggregazione per ottimizzare le prestazioni. Ciò può essere ottenuto riducendo al minimo il numero di fasi, assicurandosi che ogni fase filtri o modifichi i dati in modo efficiente e ricorrendo eventualmente agli indici per velocizzarne l'esecuzione.

2.3 Gestione delle Autorizzazioni

Nel servizio in analisi, è fondamentale gestire i permessi attribuiti ai diversi utenti per garantire che questi ottengano risultati differenti in base al loro ruolo e alle loro autorizzazioni. Ad esempio, gli utenti amministrativi non hanno restrizioni sia sull'accesso alle operazioni offerte dal servizio sia sulle entry di audit visibili, mentre agli utenti privi di qualsiasi permesso dev'essere negato completamente l'accesso. Questa necessità di gestire gli utenti

in modo specifico crea l'esigenza di memorizzare tutte le loro informazioni all'interno di una o più collezioni in MongoDB.

L'autenticazione non è più fornita tramite SSO, come descritto nel primo capitolo, ma piuttosto gli utenti si autenticano con un classico meccanismo di nome utente e password, con quest'ultima memorizzata tramite hash per garantire una memorizzazione sicura. Lo scopo è quello di evitare la duplicazione delle informazioni relative agli utenti tra il database e il server di autenticazione, semplificando così il processo di gestione delle autorizzazioni.

Per implementare l'autenticazione tramite nome utente e password, si può utilizzare un processo di autenticazione personalizzato che coinvolge UserDetailsService, CustomUserDetails e un AuthenticationProvider personalizzato. Il servizio UserDetailsService recupera le credenziali dell'utente da una fonte di dati, in questo caso la collezione degli utenti. La classe CustomUserDetails estende l'interfaccia UserDetails per includere ulteriori dettagli sull'utente, utili per comprenderne ruoli e permessi. Invece, la classe AuthenticationProvider verifica la password dell'utente utilizzando un apposito codificatore e crea un oggetto Authentication, contenente i dati dell'utente e le autorità concesse.

Infatti, quando un utente esegue l'autenticazione con successo, i suoi permessi e/o ruoli sono mappati come "granted authority". Quest'ultimo è un concetto di Spring Security che definisce ciò che un utente può fare all'interno di un'applicazione. Utilizzando la granted authority, diventa facile introdurre protezioni

e restrizioni in base al ruolo o ai permessi posseduti. Ad esempio, l'annotazione "PreAuthorize" può essere anteposta ai metodi di un Controllor per limitare l'accesso a utenti specifici con un insieme definito di ruoli e/o permessi.

Un'altra possibilità per gestire le autorizzazioni è quella di accedere al contesto di sicurezza di Spring per ottenere i dati dell'utente attualmente autenticato ed eventualmente associare ad esso una tipologia di restrizione da applicare. Ad esempio, queste limitazioni possono poi essere trasformate in criteri, più o meno filtranti, da utilizzare per limitare o meno i risultati di una query eseguita tramite MongoTemplate, come descritto in una sezione precedente. Utilizzando questi meccanismi, diventa facile poter garantire che gli utenti possano accedere solo ai dati a cui sono autorizzati, oltre a prevenire eventuali accessi non autorizzati.

2.4 i18n: Localizzazione e Traduzione

L'internalizzazione (i18n) è un processo di progettazione e sviluppo di software che consente di adattare il sistema in modo da poter essere utilizzato in paesi e lingue diverse. L'internazionalizzazione è importante perché consente ai produttori di software di raggiungere un pubblico globale e di far fronte alle sfide della globalizzazione. La sua implementazione richiede la creazione di un'architettura flessibile e modulare che possa essere facilmente adattata a diverse culture linguistiche e convenzioni locali (26). In altre parole, la tecnologia i18n viene utilizzata per fornire un supporto software multilingue, consentendo la traduzione di testi e stringhe di sistema in diverse lingue. In particolare, i18n si occupa della gestione delle traduzioni,

della localizzazione dei formati di date, numeri e valute e della gestione dei set di caratteri.

In Spring Boot, l'implementazione dell'i18n può essere effettuata attraverso l'uso del modulo Spring Boot Internationalization, anche noto come Spring Boot i18n. Questo modulo fornisce un meccanismo di supporto multilingue per le applicazioni Spring Boot, consentendo la traduzione di testi e stringhe di sistema in diverse lingue. La sua realizzazione può avvenire attraverso l'uso di librerie o database di traduzione. Nel primo caso, le librerie consentono di separare i testi e le traduzioni del software e di gestire diverse versioni dei file di traduzione. Nel secondo caso, i database consentono di mantenere un'unica versione del testo e delle traduzioni, semplificandone la gestione.

Nel servizio in questione, come fonte di traduzione è stato utilizzato un database MongoDB. I documenti della sua collezione "i18n" rappresentano gli oggetti di traduzione, che contengono la lingua di destinazione e un elenco di “Namespace”. Quest’ultimo rappresenta lo spazio dei nomi di traduzione, cioè un insieme di costanti di traduzione associate a un determinato contesto, come “audit” nel caso in discussione.

Per implementare la traduzione vera e propria, è necessario innanzitutto definire un repository per recuperare gli oggetti di traduzione dalla collezione i18n. Successivamente, in tutte le classi di servizio in cui è necessario tradurre i risultati, le costanti di traduzione possono essere recuperate in base alla lingua di destinazione, tramite il repository precedentemente menzionato, e queste costanti possono essere poi utilizzate

per effettuare le traduzioni delle varie proprietà dei risultati prima di essere restituiti al ricevente. Per facilitare la traduzione, le costanti vengono memorizzate all'interno di una mappa, dove la chiave è la parola chiave da tradurre e il valore è la traduzione nella lingua desiderata. Si noti che in questo processo di traduzione si deve comunque prestare attenzione alla possibilità, ben concreta, che la lingua di destinazione, o addirittura una traduzione specifica, non sia presente, e quindi si deve offrire un possibile valore predefinito per evitare errori o incongruenze.

2.5 CompletableFuture: Operazioni Asincrone

In un applicativo che offre la possibilità di schedulare determinate operazioni, come l'esportazione delle entry di audit per il servizio in esame, potrebbe essere necessario doverle eseguire in modo asincrono per evitare che l'utente debba attendere per un periodo prolungato. Infatti, l'operazione di esportazione delle entry di audit può richiedere anche molte ore per essere completata a causa della necessità di generare un documento PDF con firma digitale. Per questo motivo, è possibile utilizzare la classe CompletableFuture di Java per eseguire l'operazione in modo asincrono, consentendo all'utente di continuare a utilizzare l'applicazione mentre l'operazione viene eseguita in background.

CompletableFuture è una potente classe di Java che fornisce un'interfaccia per la programmazione asincrona e concorrente. Essa rappresenta un'operazione asincrona che può essere completata esplicitamente con un valore o un'eccezione. Uno dei principali vantaggi di questa classe è la capacità di eseguire operazioni asincrone e concorrenti in modo efficiente

e sicuro, senza bloccare il thread principale. Ciò consente al codice di continuare a eseguire altre operazioni mentre le operazioni asincrone sono in esecuzione, migliorando l'efficienza complessiva del sistema. Inoltre, essa supporta la combinazione di operazioni asincrone in maniera modulare e flessibile, consentendo di concatenare le varie operazioni per creare una pipeline di elaborazione dei dati. Le operazioni asincrone possono anche essere combinate per eseguire più operazioni contemporaneamente, con un conseguente utilizzo più efficiente delle risorse. Un altro vantaggio è il supporto per la gestione degli errori. Infatti, essa fornisce un semplice meccanismo per gestire gli errori e le eccezioni in modo efficiente e sicuro, garantendo che, se un'operazione asincrona genera un'eccezione, il sistema posso gestirla in modo appropriato.

Quando si eseguono operazioni di lunga durata in modo asincrono usando CompletableFuture, uno dei problemi da affrontare è che il contesto di sicurezza di Spring non viene automaticamente inoltrato al nuovo thread, poiché si tratta di una variabile locale a un thread specifico. Questo può portare a problemi di sicurezza o a errori, poiché il detentore del contesto di sicurezza potrebbe non essere disponibile durante le operazioni asincrone. Per gestire questo problema, è possibile creare un esecutore personalizzato definendo un thread pool con il contesto di sicurezza ereditario (come mostrato in Figura 11). Questo garantisce che ogni nuovo thread creato all'interno del contesto dell'esecutore appena definito, erediti il contesto di sicurezza dal thread genitore. Successivamente è possibile utilizzare il metodo "runAsync" della classe CompletableFuture

per eseguire operazioni in maniera asincrona, passando l'esecutore appena creato per propagare il contesto di sicurezza al nuovo thread (come illustrato in Figura 11).

```
// Creare un thread pool con il contesto di sicurezza ereditato
ExecutorService executor = new DelegatingSecurityContextExecutorService(Executors.newFixedThreadPool(10));

// Eseguire l'operazione di esportazione in modo asincrono, utilizzando il metodo runAsync di CompletableFuture
CompletableFuture.runAsync(() -> {
        // Produzione PDF...
        // Firma digitale...
        // Memorizzazione file...
}, executor).whenComplete((result, exception) -> {
    if (exception != null) {
        // Gestire eventuali eccezioni che si verificano durante l'operazione
        logger.error("Errore durante l'esportazione delle entry di audit", exception);
    } else {
        // Gestire completamento operazione con successo
        logger.info("Audit entries esportate con successo");
    }
});
```

Figura 11: Utilizzo di CompletableFuture per operazioni asincrone e di lunga durata in Java

Questo metodo non restituisce un valore, quindi è utile solo per le operazioni in cui non è necessario attendere un valore di ritorno. Tuttavia, è possibile utilizzare il metodo “whenComplete” per gestire eventuali eccezioni che possono verificarsi durante l’esecuzione o semplicemente per eseguire azioni collaterali quando le operazioni asincrone sono completate con successo.

2.6 IText7: Generazioni di Documenti PDF

iText 7 è una libreria Java open source per la creazione e la manipolazione di documenti PDF. È una delle librerie PDF più popolari e supporta molte funzionalità, tra cui la creazione di documenti PDF/A, la gestione dei font, l'inserimento d’immagini e tabelle, la creazione di campi compilabili, la gestione dell'accessibilità e molto altro.

Uno dei principali vantaggi di questa libreria è la possibilità di avere un controllo completo sulla creazione del documento PDF attraverso il ricorso a classi specializzate a seconda delle esigenze. Ad esempio, una volta creata un'istanza del documento PDF, rappresentata dalla classe PdfDocument, è possibile aggiungervi contenuti come paragrafi (Paragraph), tabelle (Table), immagini (Image) e molto altro.

Per garantire una gestione efficiente della RAM, è preferibile creare il PDF direttamente su un flusso di output di un file, invece di mantenere l'intero documento in memoria. Questo perché i documenti PDF possono essere molto grandi e mantenere l'intero documento in memoria può consumare una quantità significativa di RAM, causando potenzialmente problemi di prestazioni o addirittura crash. Per scrivere il documento PDF direttamente in un flusso di output di un file, è possibile utilizzare la classe “FileOutputStream” o qualsiasi altra classe che implementi l'interfaccia OutputStream. In questo modo, il documento PDF viene scritto nel flusso di output in parti più piccole, riducendo la quantità di memoria utilizzata dall'applicazione.

2.6.1 Documenti in Formato PDF/A

PDF/A è uno standard ISO per i documenti PDF archiviabili che garantisce la conservazione e il recupero accurato del documento nel tempo (27). La conformità a tale formato è particolarmente importante per i documenti giudiziari, come le esportazioni di entry di audit generate dal sistema in analisi, poiché questi documenti devono essere conservati

e recuperati con precisione nel tempo. Questi documenti sono spesso soggetti a requisiti legali per la conservazione a lungo termine e la conformità PDF/A garantisce che il contenuto, la struttura e l'aspetto rimangano intatti e leggibili nel tempo.

Per creare un documento PDF/A in Java, è necessaria una libreria che supporti appunto tale standard, e iText è una di queste. La classe PdfADocument di iText può essere utilizzata per creare un documento conforme al suddetto formato. Questa classe richiede tre parametri: un PdfWriter, il livello di conformità desiderato e un oggetto PdfOutputIntent che specifica le informazioni di output. Nella fattispecie, PdfWriter è l'oggetto che scrive il documento PDF su disco o su un flusso di output. È essenziale utilizzare un PdfWriter che supporti la conformità PDF/A, come PdfAWriter. Il livello di conformità a PDF/A specifica il grado di compatibilità con tale standard. Esistono diversi livelli, tra cui PDF/A-1a, PDF/A-1b, PDF/A-2a, PDF/A-2b, ecc. Ovviamente, il livello di conformità appropriato deve essere specificato in base ai requisiti del progetto. Infine, l'oggetto PdfOutputIntent specifica le informazioni di output, come l'identificatore della condizione di output, il nome del registro, le informazioni sull'intento e il profilo ICC (International Color Consortium) per la gestione del colore. Tutte queste informazioni sono necessarie per garantire la conformità del documento alle specifiche PDF/A.

È inoltre fondamentale includere le informazioni sui font e sui colori direttamente nel documento. I caratteri devono infatti essere incorporati nel documento per garantire la corretta visualizzazione su tutti i dispositivi. Il documento deve anche includere

informazioni sul colore, come i profili ICC, per garantirne una riproduzione accurata e per consentire al documento di essere conforme ai requisiti e di essere conservato e recuperato con precisione nel tempo.

2.6.2 Template e Modelli

In iText 7, i modelli e i template vengono utilizzati per creare e gestire documenti PDF con un layout e una struttura coerenti.

Un modello rappresenta la struttura logica di un documento PDF. Contiene i metadati e i dati che lo compongono, come titolo, autore, testo, immagini e altri contenuti. Il modello viene creato utilizzando le varie classi disponibili, come Paragraph, Table e Image. Queste classi forniscono un modo per aggiungere contenuti a un documento PDF in modo strutturato, garantendo che il documento sia ben formato e conforme alle specifiche PDF.

Un template è una struttura o un layout riutilizzabile per un documento PDF. Esso definisce aspetti quali la posizione delle intestazioni e dei piè di pagina, le dimensioni e la posizione delle aree di contenuto e lo stile generale del documento. I template possono essere utilizzati per creare documenti dall'aspetto uniforme su più pagine o addirittura su più documenti.

Pertanto, per creare un documento PDF con un layout e una struttura coerenti, è possibile utilizzare una combinazione di modelli e di template. Si può definire un template che specifichi la struttura del documento, e poi utilizzare i modelli per aggiungere contenuti al documento, come paragrafi, tabelle e immagini.

I modelli possono essere aggiunti al template in punti specifici, assicurando che il documento sia conforme al layout specificato.

2.7 Firma Digitale con iText7 e BouncyCastle

La firma digitale è una tecnica matematica utilizzata per verificare l'autenticità e l'integrità dei documenti digitali (28). È un tipo di firma elettronica che utilizza un codice univoco generato da un algoritmo crittografico per aggiungere un livello di sicurezza a un documento.

Quando si tratta di documenti PDF, l'applicazione della firma digitale è particolarmente utile per scopi legali. La firma digitale aggiunta a un documento PDF assicura infatti che il documento non sia stato manomesso, garantisce l'identità del firmatario e fornisce una prova verificabile del processo di firma. Questa caratteristica la rende uno strumento fondamentale per le aziende, le organizzazioni e i privati che trattano informazioni sensibili e necessitano di documenti sicuri e legalmente vincolanti.

Per applicare una firma digitale a un documento PDF, si possono utilizzare librerie come BouncyCastle e iText7. La prima è una libreria crittografica basata su Java che fornisce protocolli e algoritmi di sicurezza per varie esigenze. La seconda, invece, è una libreria per la generazione di documenti PDF in Java, già presentata nella sezione precedente.

Affinché si possa applicare una firma digitale a un documento PDF utilizzando tali librerie, è necessario innanzitutto caricare il keystore che contiene la chiave privata da utilizzare per firmare il documento. Un keystore è una struttura di archiviazione sicura

utilizzata per memorizzare certificati digitali e chiavi private. È un file che contiene una o più chiavi crittografiche, insieme ai corrispondenti certificati digitali che identificano il proprietario di tali chiavi. In genere esso viene caricato da un file locale in formato PCKS12, che è un formato standard per la memorizzazione di chiavi private e certificati. Per accedere al keystore è necessaria una password, solitamente fornita e recuperabile dalle proprietà dell'applicazione.

Una volta caricato il keystore, è possibile recuperare la chiave privata e la catena di certificati. In particolare, la prima è una chiave segreta utilizzata per firmare il documento PDF, mentre la seconda è una sequenza di certificati digitali in grado di identificare l'entità che firma il documento PDF e di verificarne l'identità. La chiave privata viene mantenuta segreta e non viene mai condivisa, mentre la catena di certificati contiene certificati pubblici che possono essere condivisi con altri per verificare l'autenticità del documento PDF firmato.

Dopo aver recuperato la chiave privata e la catena di certificati, è possibile creare la firma utilizzando la classe PrivateKeySignature fornita da iText7 e fornendo la chiave privata e l'algoritmo di hashing da utilizzare per firmare il documento PDF, ad esempio SHA256. La chiave privata impiegata per la firma è una chiave privata a curva ellittica, ovvero un tipo adoperato in ECDSA, che si riferisce all'Elliptic Curve Digital Signature Algorithm, un tipo di algoritmo di firma digitale che utilizza la crittografia a curva ellittica. Per creare una firma digitale, è necessario fornire anche un digest, che può essere ottenuto creando un'istanza

di BouncyCastleDigest che implementa l'interfaccia IExternalDigest, la quale offre la possibilità di creare un hash del documento PDF al quale verrà apposta la firma.

L'ultimo passo consiste nell'impostare l'aspetto della firma digitale utilizzando la classe PdfSignatureAppearance fornita da iText, che consente di personalizzare l'aspetto visivo del campo della firma. Utilizzando questa classe è infatti possibile impostare il modello, la posizione, il creatore e il nome del campo della firma.

Una volta impostato l'aspetto, è possibile utilizzare la classe PdfSigner per apporre la firma sul documento. Questa classe consente anche di specificare se la firma deve essere separata o meno. Nel primo caso, la firma può essa essere inclusa in un file separato, tipicamente un file CMS, che può essere utile per l'archiviazione a lungo termine e la verifica del documento firmato. Infatti, questa è la scelta applicata per il servizio in discussione, in quanto si adatta alle sue esigenze.

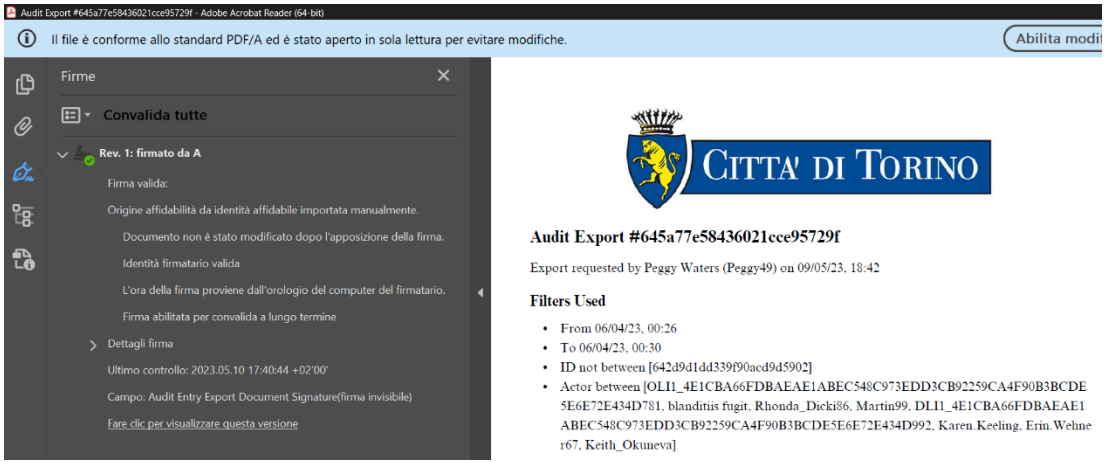


Figura 12: Esempio di un PDF con firma digitale valida generato dall'Auditing System

2.7.1 Test di Manomissione

Per verificare la correttezza della firma digitale applicata a un documento PDF, si possono effettuare delle prove di manomissione principalmente in due modi differenti: modificando direttamente il PDF con un software come Adobe Reader e modificandolo tramite codice.

Nel primo caso, un documento PDF che è stato firmato digitalmente può essere semplicemente aperto in Adobe Reader e modificato. Ad esempio, è possibile aggiungere una frase al documento o modificare un'immagine. Dopo aver apportato la modifica, l'utente può verificare la firma attraverso il pannello correlato di Adobe Reader, che visualizzerà un messaggio di avviso indicante che il documento è stato modificato e che la firma non copre più l'intero documento. L'utente può quindi visualizzare i dettagli della firma per vedere le modifiche esatte apportate al documento dopo l'ultima firma, come mostrato in Figura 13.

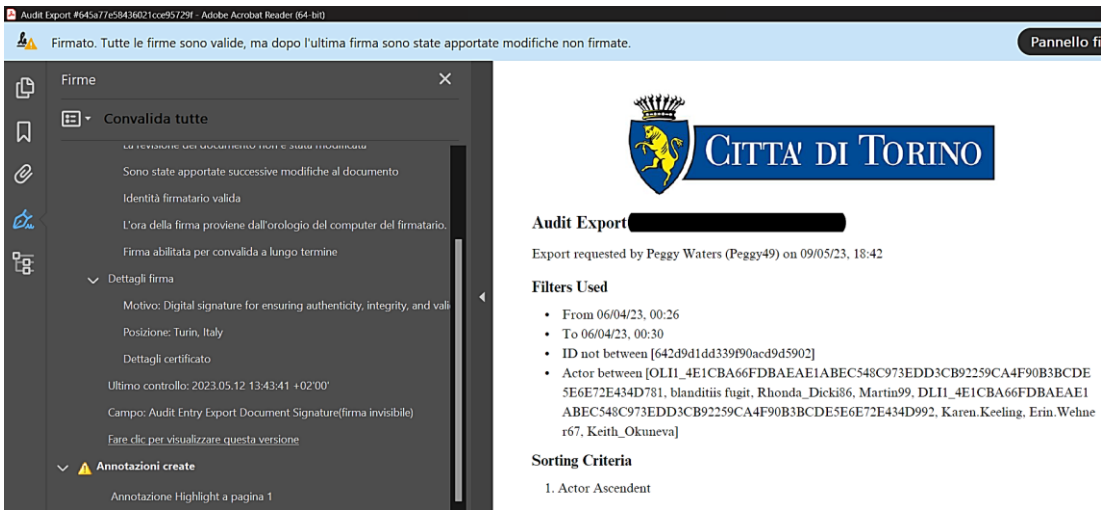


Figura 13: Il pannello di convalida della firma di Adobe Reader notifica che il documento PDF è stato modificato

Per la seconda tipologia di test, il documento PDF deve essere aperto a livello di codice utilizzando un flusso di output per poter apportare modifiche a esso. Ad esempio, è possibile modificarne il contenuto o i metadati utilizzando le classi fornite da iText. Dopo aver effettuato la modifica, è possibile verificarne l'integrità utilizzando la classe SignatureUtil d'iText per ottenere i nomi delle firme dal PDF. Per ogni firma trovata, è possibile verificare se la firma in questione non copre l'intero documento, il che indicherebbe che il documento è stato modificato. È anche possibile leggere i dati della firma come file PKCS7, mappati da un oggetto PdfPKCS7, e verificarne l'integrità e l'autenticità utilizzando un metodo fornito da iText per tale classe. Se la firma non è valida, il metodo restituisce un valore falso, indicando che il documento è stato manomesso.

Nel complesso, questi test di manomissione sono un modo per verificare che la firma digitale, applicata ai documenti PDF generati dall'applicativo, sia corretta e che qualsiasi modifica non autorizzata possa essere efficacemente rilevata.

2.8 Conclusioni

In questo capitolo si è discusso di come sia possibile creare un servizio che consenta la consultazione e l'esportazione delle informazioni di audit prodotte da un sistema in ambito investigativo. Come accennato in precedenza, questo servizio possiede molti aspetti in comune con quello descritto nel primo capitolo; infatti, essi condividono la stessa struttura modulare e in generale tutti gli aspetti legati alla sicurezza. Tuttavia, si è visto come gli obiettivi e i requisiti principali siano diversi, in quanto in questo caso il servizio necessita principalmente

di prestazioni elevate, dato l'alto volume di dati che deve gestire, e di proteggere i documenti prodotti da modifiche. Infatti, a differenza del requisito di confidenzialità necessario per il file di licenze generati dal Licensing Generator, in questo caso si tratta di rendere i documenti PDF generati protetti da eventuali manomissioni e di garantire che essi possano essere conservati in modo sicuro nel tempo.

Per soddisfare questi requisiti, sono state inizialmente esplorate diverse tecniche per ottimizzare le prestazioni con MongoDB, dato che in questo caso il volume di dati da gestire è dell'ordine della decina di milioni al giorno. In particolare, si è parlato dell'utilizzo di Mongo Template, il quale fornisce un livello di astrazione di alto livello rispetto all'implementazione predefinita di Spring Data, offrendo un controllo a maggiore precisione sulle modalità d'interazione con il database. Si è anche illustrato come utilizzare le funzionalità di paginazione, aggregazione, indicizzazione e localizzazione per migliorare le prestazioni e ridurre il tempo necessario per recuperare e gestire i dati provenienti da MongoDB.

Sebbene l'attenzione iniziale fosse rivolta principalmente alle prestazioni, si è poi discussa l'importanza di creare un sistema sicuro per gestire le autorizzazioni e garantire che le informazioni di audit siano protette da accessi e modifiche non autorizzati. Sono stati analizzati infatti i metodi per utilizzare le "granted authority" per gestire i ruoli e i permessi degli utenti.

Infine, la parte più significativa del capitolo è stata incentrata sull'esportazione delle informazioni di audit, tra cui la generazione di documenti PDF utilizzando la libreria iText7. La discussione

ha riguardato anche l'utilizzo della classe CompletableFuture per avviare la generazione dei documenti PDF in modo asincrono, consentendo al richiedente di ricevere il documento in futuro senza dover attendere a tempo indeterminato il completamento del processo di generazione. Si è anche illustrato il modo in cui applicare una firma digitale a questi documenti utilizzando iText7 e BouncyCastle, assicurando la loro protezione da qualsiasi potenziale modifica.

In sintesi, questo capitolo ha evidenziato l'importanza delle prestazioni, della gestione delle autorizzazioni e della sicurezza nella creazione di un sistema robusto ed efficiente per la consultazione e l'esportazione delle informazioni gestite. Implementando le tecniche illustrate, è possibile creare un sistema in grado di garantire che le sue informazioni siano sempre accurate, complete e a prova di manomissione, in modo da assicurarne anche la validità legale.

Capitolo 3

Code Obfuscation

Questo capitolo descrive la ricerca, lo studio e l'implementazione di un processo di offuscamento del codice sorgente di un'applicazione Java realizzata con il framework Spring Boot. Prima di approfondire gli strumenti di offuscamento specifici per le applicazioni Java Spring Boot, verrà fornita una breve panoramica delle principali tecniche di offuscamento e deoffuscamento. Dopo aver acquisito una comprensione fondamentale di questi concetti, si analizzeranno i migliori strumenti per l'offuscamento delle applicazioni Java. In particolare, verranno analizzate le caratteristiche e le capacità di questi strumenti, nonché la loro efficacia nel proteggere il codice sorgente delle applicazioni.

3.1 Introduzione

Al giorno d'oggi, le applicazioni software sono diventate parte integrante della nostra vita quotidiana. Con la crescente complessità del software, è diventato essenziale proteggere il codice sorgente di queste applicazioni da accessi e modifiche non autorizzate. Uno dei modi per raggiungere questo obiettivo è l'offuscamento.

L'offuscamento consiste nel trasformare il codice sorgente di un'applicazione in una forma oscura e difficile da comprendere, pur preservandone la funzionalità. Il processo di offuscamento ha infatti tre scopi principali. In primo luogo, rende più difficile la modifica del codice, il che a sua volta aiuta a prevenire l'elusione dei controlli dell'applicazione, come la verifica delle licenze all'interno del sistema. In secondo luogo, rende più difficile a terzi alterare il funzionamento del software. Infine, protegge la proprietà intellettuale rendendo più complesso il reverse engineering e/o il riutilizzo del codice sorgente da parte di terzi (29).

Tuttavia, l'offuscamento delle applicazioni Spring Boot può presentare sfide uniche rispetto ad altri tipi di applicazioni Java. Questo è dovuto alla natura dinamica della configurazione dei bean e delle funzioni di autoconfigurazione di Spring Boot, che può rendere difficile determinare quali classi e metodi debbano essere offuscati. Inoltre, le applicazioni Spring Boot si affidano spesso a librerie e framework di terze parti, che possono avere dipendenze e requisiti di configurazione propri. L'offuscamento di queste dipendenze può essere impegnativo, in quanto può richiedere la modifica del codice sorgente o dei file

di configurazione, e pertanto esse sono solitamente lasciate intatte.

Nonostante queste sfide, l'offuscamento rappresenta comunque un elemento importante per proteggere le applicazioni Spring Boot da accessi e modifiche non autorizzati. Infatti, l'obiettivo finale di questa attività è soprattutto quello di proteggere i servizi descritti e analizzati nei primi due capitoli dell'elaborato.

3.2 Offuscamento e Deoffuscamento del Codice

Come anticipato nell'introduzione, l'obiettivo principale dell'offuscamento è quello di proteggere il codice sorgente da accessi e modifiche non autorizzati, nonché di prevenire il reverse engineering e il furto di proprietà intellettuale.

In questa sezione, verranno approfondite alcune delle tecniche più diffuse e utilizzate per l'offuscamento e la deoffuscamento del codice sorgente.

3.2.1 Tecniche di Offuscamento

Esistono diverse tecniche di offuscamento che possono essere applicate al codice sorgente e la scelta di queste dipende dalle esigenze specifiche dell'applicazione e dal livello di protezione richiesto. Alcune delle tecniche di offuscamento più comunemente utilizzate sono la ridenominazione, la crittografia delle stringhe e l'offuscamento del flusso di controllo.

La ridenominazione consiste nel cambiare i nomi di variabili, classi e metodi nel codice sorgente per renderlo più difficile da comprendere. La ridenominazione può essere eseguita

manualmente o con strumenti automatici e può aumentare significativamente la complessità del codice.

La crittografia delle stringhe implica invece la cifratura delle stringhe presenti nel codice sorgente per renderle più difficili da leggere. Le stringhe vengono poi decifrate solo in fase di esecuzione, il che può rendere più difficile la comprensione del codice da parte degli eventuali malintenzionati.

Infine, l'offuscamento del flusso di controllo consiste nel modificare la logica delle istruzioni presenti nel codice sorgente per rendere più difficile la comprensione del programma. Tale offuscamento può essere effettuato con diverse tecniche, come la suddivisione del codice, lo srotolamento dei cicli e l'inserimento di codice morto. La prima consiste nel suddividere il codice in parti più piccole e riorganizzarle in modo da rendere più difficile la comprensione della logica generale del programma. Questa tecnica può essere applicata a singole funzioni o a interi moduli di codice e può rendere più difficile per gli attaccanti seguire il flusso di controllo del programma. Lo srotolamento dei cicli riguarda la trasformazione dei costrutti di ciclo (come for e while) nel codice in modo che non siano più riconoscibili come cicli, rendendo più difficile la comprensione dell'algoritmo sottostante. Invece, inserire codice morto consiste nell'introdurre delle diramazioni di codice fittizie e delle dichiarazioni condizionali che non vengono mai eseguite. Questo può rendere più difficile la comprensione della logica e del flusso di controllo del programma, oltre a rendere più difficile l'identificazione delle parti del codice che sono effettivamente rilevanti per la funzionalità del programma.

Tuttavia, è importante tenere conto del fatto che l'offuscamento del codice può fornire un falso senso di sicurezza e presenta diversi potenziali svantaggi che devono essere attentamente considerati prima dell'implementazione. Uno degli svantaggi principali è l'aumento delle dimensioni dell'applicazione e dell'overhead di runtime, poiché il codice offuscato può essere di dimensioni maggiori rispetto a quello non offuscato e l'implicare l'esecuzione di un numero di istruzioni più elevato, con conseguente deterioramento delle prestazioni dell'applicazione.

3.2.2 Tecniche di Deoffuscamento

Il deoffuscamento è il processo d'inversione delle tecniche di offuscamento applicate al codice sorgente per recuperare il codice originale o anche solo per capirne il funzionamento. Esso è solitamente un processo complesso e possibilmente lungo, ma è spesso necessario in situazioni come la verifica delle vulnerabilità, la rimozione della protezione "anticopia", l'elusione delle restrizioni di accesso e la personalizzazione di sistemi integrati (30).

Alcune delle tecniche più comunemente utilizzate sono la decompilazione, l'analisi dinamica e l'analisi statica. La prima prevede l'utilizzo di un decompilatore per convertire il bytecode offuscato nel codice sorgente originale. L'analisi dinamica consiste invece nell'eseguire il codice offuscato in un debugger o in un altro ambiente di esecuzione per comprenderne il comportamento. Viceversa, l'analisi statica consiste nell'analizzare il codice offuscato senza eseguirlo, utilizzando tecniche quali il disassemblaggio o l'analisi binaria. L'analisi statica può essere utilizzata in particolare per identificare

modelli e costrutti specifici nel codice, nonché per comprendere la struttura e il flusso di controllo del programma.

Nonostante la sua utilità nel recuperare il codice sorgente originale, presenta anch'essa alcuni potenziali svantaggi che devono essere presi in considerazione. Uno degli inconvenienti principali è che le tecniche di deoffuscamento non sono sempre in grado di recuperare completamente o accuratamente il codice sorgente originale. Questo può dipendere dalla complessità del codice e dalle tecniche di offuscamento utilizzate, oltre che dalle limitazioni delle tecniche di deoffuscamento stesse. Un altro potenziale inconveniente è la possibilità di introdurre errori o bug nel codice. Essa comporta la modifica del codice originale offuscato e, se non viene eseguita con attenzione, può portare a cambiamenti non voluti nel codice che possono influire negativamente sulla sua funzionalità e sulle sue prestazioni. Di conseguenza, è importante valutare attentamente i limiti delle tecniche di deoffuscamento prima di utilizzarle, per garantire che i risultati siano accurati e affidabili.

3.3 Parametri e Strumenti di Valutazione

Per valutare l'efficacia dei processi di offuscamento del codice, che verranno analizzati nella prossima sezione, è importante considerare diversi parametri e utilizzare gli strumenti appropriati. Nel presente studio, ogni software di offuscamento sarà valutato principalmente sulla base di cinque parametri di valutazione, quali la reputazione, basata sulle recensioni e sui feedback degli utenti che lo hanno utilizzato in precedenza, la facilità d'uso, in quanto un software facile da usare e ben documentato può essere conveniente da utilizzare, l'impatto sulla funzionalità

e sulle prestazioni dell'applicazione, la qualità, per verificare se il codice risultante è comprensibile o meno, e l'efficacia, per capire quanto gli strumenti di reverse engineering siano in grado di ripristinare il codice originale.

A supporto del processo di valutazione vengono naturalmente utilizzati diversi strumenti di reverse engineering, come JD-GUI, Procyon, FernFlower, Bytecode Viewer, Java Decompiler e Java-Deobfuscator. Questi rappresentano attualmente gli strumenti di decompilazione e deoffuscamento open source più diffusi e conosciuti e vengono utilizzati per decompilare il bytecode dell'applicazione e visualizzarlo come codice sorgente, al fine di analizzarne il flusso di controllo e verificare l'efficacia dell'offuscamento. Una eccezione è rappresentata da Deobfuscator, il quale rappresenta uno strumento open source per il deoffuscamento del codice di un'applicazione Java e sarà lo strumento principale per valutare l'efficacia dei vari software di offuscamento analizzati.

3.4 Analisi degli Strumenti di Offuscamento

La scelta degli strumenti di offuscamento è stata effettuata considerando diverse fonti e prendendo in considerazione sia offuscatori gratuiti che a pagamento. In particolare, sono stati consultati vari blog, articoli e documenti specializzati nel campo dello sviluppo software. Sulla base delle informazioni raccolte, sono stati poi selezionati i cinque offuscatori più utilizzati nonché consigliati dalla comunità.

Tra gli offuscatori selezionati vi sono dunque sia prodotti open source che commerciali. Per i primi sono stati selezionati ProGuard e yGuard, due programmi open source molto popolari

per l'offuscamento del codice Java. Per gli offuscatori a pagamento, invece, sono stati individuati DashO, Allatori e Zelix Klassmaster, tutti e tre molto popolari e considerati tra i migliori strumenti di offuscamento commerciali attualmente disponibili sul mercato.

L'elenco originale comprendeva anche altri due offuscatori, Bozar e Stringer, che sono stati successivamente esclusi per diversi motivi. Bozar è stato escluso a causa di un debole algoritmo impiegato nel suo processo di offuscamento, il che lo rende facilmente deoffuscabile con diversi strumenti in circolazione. Per Stringer invece, sebbene sia un altro celebre offuscatore commerciale per Java, non è stato possibile ottenere una licenza di prova.

Nelle seguenti sottosezioni, i cinque offuscatori identificati saranno quindi analizzati e valutati in base ai cinque parametri descritti nella sezione precedente.

3.4.1 ProGuard

ProGuard è uno strumento open source per la riduzione delle dimensioni, l'ottimizzazione, l'offuscamento e la preverifica del bytecode Java (31). Esso è sviluppato dall'azienda Guardsquare, la quale si occupa di sicurezza informatica, ed è stato utilizzato con successo in molte applicazioni Java tant'è che da diversi anni è incluso nella suite di strumenti di sviluppo Android di Google. Questo gli ha permesso ovviamente di crearsi una buona reputazione negli anni ed è infatti riconosciuto come il miglior offuscatore open source attualmente disponibile.

ProGuard offre alcune caratteristiche interessanti per l'offuscamento del codice, tra cui la possibilità di spostare tutte le classi offuscate nello stesso package per aumentare la protezione e personalizzare il dizionario usato per l'offuscamento dei nomi. Tuttavia, non fornisce ufficialmente un'integrazione con Maven e la parte di configurazione non risulta essere di facile realizzazione, tanto che è necessario procedere per tentativi ed errori. Sebbene la documentazione sia adeguata e dettagliata riguardo alle possibili configurazioni e personalizzazioni disponibili, non vi sono esempi concreti per comprenderne il funzionamento complessivo. Sono disponibili diverse personalizzazioni per l'offuscamento dei nomi e per ulteriori misure di protezione, ma non vi sono configurazioni per l'attivazione dell'offuscamento del flusso di controllo o della crittografia delle stringhe. Inoltre, il processo di esclusione di alcune classi, metodi e campi dall'offuscamento è più macchinoso che in altri offuscatori esaminati.

Dopo aver utilizzato ProGuard per offuscare il codice sorgente dei servizi precedentemente descritti in questo elaborato, si è constatato che tutte le funzionalità del sistema sono rimaste intatte e la dimensione degli eseguibili è rimasta invariata. Tuttavia, è stato rilevato un impatto negativo sulle prestazioni, con un aumento dei tempi di risposta, soprattutto per le azioni più onerose. Ad esempio, la generazione di PDF nel Auditing System e la generazione di file di licenze nel Licensing Generator hanno subito un peggioramento delle prestazioni del 30%.

Nel complesso, sebbene l'offuscamento dei nomi di ProGuard offra una buona protezione, è solo una piccola parte della sicurezza del codice e non garantisce una protezione completa. Nonostante che con la configurazione scelta sia stato possibile introdurre un certo grado di confusione all'interno del codice, richiedendo così un po' più di tempo per apportare modifiche al funzionamento del software, la logica del programma è rimasta intatta e quindi un malintenzionato, dopo aver decompilato il codice, sarebbe sicuramente in grado di comprendere l'intento di una specifica classe o funzione.

Se il progetto richiedesse solo un livello di protezione relativamente basso, l'offuscamento dei nomi offerto da ProGuard potrebbe essere sufficiente. Tuttavia, nel caso in cui il progetto richieda un livello di protezione più elevato, potrebbe essere necessario utilizzare altre tecniche di offuscamento oltre all'offuscamento dei nomi offerto da questo software.

3.4.2 yGuard

yGuard è un software di offuscamento per applicazioni Java che mira a proteggere il codice sorgente rendendolo più difficile da comprendere o decompilare. Esso è progettato per essere utilizzato con applicazioni Java indipendentemente dal framework utilizzato (32).

La reputazione di YGuard per quanto riguarda l'offuscamento del codice è generalmente ritenuta non all'altezza, con recensioni contrastanti. Mentre alcuni suggeriscono il suo buon funzionamento e la sua relativa facilità di configurazione, altri indicano che potrebbe essere meno efficace in termini

di ottimizzazione e dimensione del codice rispetto ad altri software di offuscamento come ProGuard.

In effetti, la reputazione è stata confermata dal suo utilizzo. Nonostante la documentazione adeguata e dettagliata per l'integrazione con vari strumenti di compilazione, tra cui Maven, il software offre un'ampia gamma di funzioni esclusivamente per quanto riguarda la ridenominazione di classi e variabili, il che può far pensare che non sia potente come altri strumenti di offuscamento presenti sul mercato. In aggiunta, la configurazione per mantenere univoci i nomi dei bean può essere più complessa rispetto ad altre soluzioni e questo causa non pochi problemi nell'offuscamento di un'applicazione Spring Boot. Infatti, richiede l'uso di funzionalità di mappatura degli attributi delle classi o di non offuscare i nomi di queste ultime, mentre invece altri software di offuscamento forniscono direttamente una configurazione predefinita per mantenere unici i nomi dei bean (requisito di qualsiasi applicazione Spring).

L'offuscamento eseguito da YGuard è molto semplice e si limita a una debole ridenominazione di classi, metodi e variabili senza aggiungere alcuna complessità alla logica. Questo tipo di offuscamento potrebbe non essere sufficiente per contrastare eventuali attacchi e pertanto potrebbe essere facile per un attaccante esperto ricostruire il codice originale esaminando il codice decompilato.

Nel complesso, l'offuscamento dei nomi utilizzato da YGuard può essere considerato una forma molto limitata di offuscamento del codice. Senza protezioni aggiuntive, il codice è minimamente

più difficile da leggere e comprendere da parte di un potenziale malintenzionato. Di conseguenza, l'efficacia di YGuard nel proteggere il codice sorgente è decisamente bassa.

3.4.3 DashO

DashO è uno strumento commerciale in grado di offrire offuscamento professionale e una protezione completa per applicazioni Java e Android. È presente sul mercato da oltre 20 anni ed è costantemente aggiornato per stare al passo con l'evolversi delle minacce alla sicurezza affrontate dalle squadre di sviluppo delle applicazioni (33). DashO non fornisce solo funzionalità di offuscamento, ma include anche il watermarking, ovvero l'incorporazione d'informazioni sul copyright nel jar, la segnalazione di errori e la "chiamata a casa" per segnalare eventuali manomissioni.

Esso ha globalmente ricevuto recensioni positive per i suoi strumenti di offuscamento, tra cui la protezione antimanomissione e la possibilità di configurare diversi livelli di offuscamento a un prezzo ragionevole. Tuttavia, alcuni utenti hanno riscontrato difficoltà di utilizzo a causa della complessità nel comprendere le funzionalità fornite e della mancanza di materiale formativo online.

DashO non è uno strumento integrato nel processo di compilazione, ma piuttosto un'applicazione desktop che offre una GUI dall'aspetto semplice per applicare le protezioni desiderate tra cui l'offuscamento. Sebbene offra tutte le funzionalità necessarie per offuscare un'applicazione Spring Boot, la sua configurazione risulta essere faticosa in quanto

non offre scorciatoie per specificare quali classi, metodi o proprietà mantenere intatte.

L'attivazione di tutte le protezioni disponibili, tra cui l'offuscamento del flusso di controllo, l'offuscamento dei nomi e la crittografia delle stringhe, ha comportato un notevole aumento della complessità del codice. Inoltre, dopo l'offuscamento, tutte le funzionalità sono rimaste intatte e la dimensione dei file eseguibili è rimasta praticamente invariata. Tuttavia, si è verificato un leggero e accettabile rallentamento dei tempi di risposta delle applicazioni, con un aumento di circa il 10%.

L'offuscamento del flusso di controllo ha introdotto un numero significativo di costrutti aggiuntivi e ha aumentato il numero di variabili locali, rendendo il codice meno comprensibile e difficile da alterare. In questo modo il codice sorgente offuscato è diventato anche più difficile da decompilare rispetto al codice originale, fornendo un ulteriore livello di protezione contro eventuali attacchi. Tuttavia, l'offuscamento dei nomi non è stato del tutto efficace, poiché i nomi di alcuni pacchetti e classi/metodi non sono stati offuscati, diminuendo la sicurezza complessiva delle applicazioni.

Lo strumento di deoffuscamento impiegato è stato in grado d’identificare DashO come il software di offuscamento utilizzato nonché il tipo di offuscamento eseguito, senza però riuscire a ricostruire completamente il codice sorgente originale. Inoltre, le stringhe crittografate non sono state decifrate, contribuendo a mantenere una buona efficacia complessiva.

3.4.4 Allatori

Allatori è un offuscatore commerciale Java di seconda generazione che offre una protezione completa; infatti, non si limita solo a nascondere il codice sorgente, ma incorpora anche funzioni aggiuntive che rendono più difficile la decompilazione del codice (34).

La reputazione di Allatori per l'offuscamento del codice è contrastante. Alcuni lodano la facilità di configurazione e d'integrazione, altri criticano la scarsa efficacia del solo offuscamento e la facilità con cui la protezione può essere rimossa con strumenti pubblici, con il rischio di ridurre le prestazioni dell'applicazione. Tuttavia, molti utenti hanno apprezzato la buona capacità di personalizzare le impostazioni e la presenza di protezioni aggiuntive.

Il prodotto fornito è corredato da una documentazione dettagliata e da esempi pratici che aiutano a comprendere le diverse alternative disponibili. La struttura dei file di configurazione è organizzata in modo gerarchico e comprende diversi tag, ognuno dei quali ha specifici sotto-tag per ulteriori impostazioni.

I principali vantaggi di questo strumento sono la facilità d'integrazione nell'ambiente di sviluppo e il fatto che dopo aver effettuato l'offuscamento non è stato osservato alcun degrado delle prestazioni o delle funzionalità dell'applicazione. Infatti, i servizi protetti continuano a funzionare completamente come prima, con tempi di risposta sostanzialmente identici, e la dimensione dei file eseguibili dell'applicazione è rimasta quasi identica a quella originale.

Sono disponibili tutte le principali protezioni legate all'offuscamento, tra cui la crittografia delle stringhe, l'offuscamento del flusso di controllo e la ridenominazione. Il risultato è che il codice generato presenta un buon grado

di offuscamento, soprattutto grazie a un forte offuscamento dei nomi delle classi e dei relativi metodi e proprietà e alla crittografia delle stringhe. Tuttavia, è stato necessario escludere dall'offuscamento alcune classi, in particolare quelle relative ai DTO e ai modelli, per garantire il corretto funzionamento delle applicazioni.

Invece, l'offuscamento del flusso di controllo ha fornito buoni risultati esclusivamente nelle classi con una logica considerevole, mentre in tutte le altre classi ha apportato solo miglioramenti marginali. Tuttavia, nonostante il deoffuscatore non sia stato in grado di rimuovere completamente l'offuscamento del flusso di controllo, ha purtroppo segnalato che la crittografia delle stringhe impiegata è piuttosto elementare e ha identificato una classe in cui sono presenti tutte le chiavi per decifrare le stringhe presenti.

Complessivamente, il sistema di offuscamento del codice offerto da Allatori si è rivelato efficace nel rendere difficile la comprensione del codice, ma l'importante vulnerabilità presente per decifrare le stringhe rappresenta un grosso svantaggio.

3.4.5 Zelix KlassMaster

Zelix KlassMaster è uno strumento commerciale nato per proteggere il codice sorgente Java dalla decompilazione e dall'analisi statica. Il prodotto offerto offre una vasta gamma di funzionalità di offuscamento e possiede una reputazione solida e affidabile con una storia che risale addirittura al 1997 (35). Infatti, i clienti che hanno utilizzato il prodotto hanno espresso grande soddisfazione, lodando la solidità della protezione e l'efficacia dell'offuscamento fornito.

La documentazione di Zelix può risultare inaccessibile a causa della grafica obsoleta del sito, che rende difficile individuare i passaggi necessari per configurare il sistema. Tuttavia, una volta appreso il funzionamento, la configurazione diventa semplice, grazie all'uso di alcune parole chiave e selettori per aggiungere le proprietà e le personalizzazioni desiderate. Zelix offre diverse opzioni di configurazione per l'offuscamento, tra cui l'offuscamento del flusso di controllo, la crittografia delle stringhe e altre caratteristiche aggiuntive come la gestione automatica delle riflessioni, la possibilità di comprimere tutto in un unico pacchetto, la forzatura dei nomi unici e così via.

La configurazione predefinita di Zelix KlassMaster attiva già numerose protezioni, tra cui l'offuscamento del flusso, la rimozione delle informazioni di debug e la protezione della tabella delle eccezioni, che contribuiscono ad aumentare la complessità del codice e a renderlo più difficile da analizzare. Aggiungendo a queste configurazioni predefinite altre opzioni personalizzabili, come la gestione della riflessione, la crittografia delle stringhe e la personalizzazione dei nomi di classi e metodi, è possibile aumentare notevolmente la protezione del codice.

L'impatto di Zelix sull'esecuzione del codice è minimo, in quanto non vi è alcun degrado significativo delle prestazioni e la funzionalità del codice rimane intatta, senza modifiche alle dimensioni del JAR. Inoltre, si è dimostrato molto efficace nella protezione del codice sorgente, garantendo un elevato livello di sicurezza e rendendo il codice sorgente più difficile da analizzare e alterare.

Per quanto riguarda la capacità di resistere al processo di deoffuscamento, lo strumento impiegato è stato in grado d'indicare solo la possibilità di decifrare le stringhe cifrate, ma non è stato poi in grado né di svolgere tale compito né di completare il deoffuscamento del flusso di controllo. Questo dimostra la capacità di Zelix KlassMaster di offrire una protezione molto efficace nel proteggere il codice sorgente, rendendolo più difficile da analizzare e alterare.

3.5 Conclusioni

Sulla base dell'analisi effettuata, si può concludere che Zelix KlassMaster è da considerarsi il miglior offuscatore tra quelli esaminati, in quanto offre un'ampia gamma di protezioni personalizzabili, un'ottima qualità di offuscamento, una buona efficacia nella protezione del codice sorgente e in generale una maggiore sicurezza rispetto ad altri offuscatori. Difatti, è stato l'unico caso in cui gli strumenti open-source non sono stati in grado di generare una versione deoffuscata del codice, a riprova dell'efficacia della soluzione da esso offerta.

Tra gli altri offuscatori esaminati, yGuard si è rivelato essere il peggiore, offrendo una protezione di bassa qualità, inefficace e pressoché inutile. Diversamente, ProGuard e Allatori offrono indubbiamente una buona capacità di offuscamento, ma presentano al contempo alcune limitazioni, come una protezione del codice sorgente meno efficace e la possibilità di deoffuscare il codice protetto.

Tuttavia, ProGuard rappresenta ancora una scelta valida ed economica per la protezione del codice sorgente, soprattutto per i progetti a basso budget, sebbene sia importante considerare

che l'offuscamento dei nomi da solo non basta a garantire un'efficace protezione del codice sorgente.

Menzione d'onore per DashO, il quale dispone di alcune interessanti funzionalità per la personalizzazione dell'offuscamento del flusso di controllo e delle opzioni di offuscamento dei nomi, ma la sua mancata integrazione nel processo di compilazione e l'inefficacia nella protezione del codice sorgente lo rendono meno competitivo rispetto alle altre soluzioni esaminate.

Capitolo 4

Conclusioni

La realizzazione dei due servizi utilizzando il framework Java Spring Boot ha permesso di esaminare differenti modalità di protezione e di comprendere l'importanza della sicurezza nelle applicazioni web moderne. Da questo studio emerge la necessità di comprendere i requisiti applicativi prima d'implementare le opportune misure di sicurezza di cui esso necessita.

In particolare, si sottolinea l'importanza di rimanere vigili sul fatto che tutto può diventare una possibile vulnerabilità, per cui è importante ponderare ogni scelta fatta perché dietro di essa si possono nascondere possibili punti di attacco ma, allo stesso tempo, dare troppo peso a un elemento può portare ad abbassare la guardia su altre frontiere.

Le vulnerabilità più comuni, come l'injection, il cross-site scripting (XSS) e l'errata gestione dell'autenticazione e della sessione,

sono ben note e possono essere sfruttate dai malintenzionati per ottenere l'accesso non autorizzato a dati sensibili. E proprio perché sono ben note, esistono molte soluzioni e approfondimenti, e pertanto esse rappresentano sempre il punto di partenza per la protezione di un'applicazione web. In questo modo, implementando solide misure di sicurezza e rimanendo aggiornati sui rischi più frequenti, ad esempio seguendo i rapporti forniti dall'Open Web Application Security Project (OWASP), è possibile garantire la protezione dell'applicazione e di tutti i componenti che la circondano.

L'elaborato ha anche dimostrato come requisiti specifici richiedano proprietà differenti. Ad esempio, il primo servizio descritto doveva produrre file di licenze in modo confidenziale, ossia tale da non poter essere letto o modificato da terze parti, mentre il secondo doveva fornire principalmente l'autenticazione e l'integrità di documenti legali, in modo che potessero essere mantenuti per lungo tempo e non manomessi da nessuno. Infatti, per soddisfare tali esigenze, è stato necessario ricorrere a diversi meccanismi, in particolare nel primo caso alla crittografia simmetrica e asimmetrica, e nel secondo caso all'applicazione di una firma digitale utilizzando algoritmi a curva ellittica.

Infine, è stata sottolineata l'importanza dell'offuscamento del codice sorgente come misura fondamentale per salvaguardare la proprietà intellettuale e prevenire le manomissioni. Oggigiorno sono disponibili molti strumenti per l'offuscamento del codice sorgente, ognuno con le proprie caratteristiche e capacità. Pertanto, è essenziale valutare i diversi strumenti e selezionare quello che meglio soddisfa i requisiti

dell'applicazione e, in generale, del progetto. Tuttavia, è importante notare che, sebbene l'offuscamento possa offrire una protezione significativa al codice sorgente dell'applicazione, non costituisce una soluzione infallibile. Gli attaccanti esperti possono comunque trovare il modo di effettuare il reverse engineering o di manomettere il codice dell'applicazione, anche quando si utilizzano tecniche avanzate di offuscamento. Pertanto, è essenziale implementare più livelli di misure di sicurezza, per garantire una protezione completa.

Come per qualsiasi altra cosa, ci sono sempre opportunità per ulteriori esplorazioni e miglioramenti. Nel caso in discussione, si possono individuare diverse aree su cui concentrare il lavoro futuro.

In primo luogo, per quanto riguarda il Licensing Generator, si ritiene che vi sia spazio per un miglioramento dell'interfaccia grafica, in particolare per offrire una migliore esperienza utente, e che nel backend sia possibile migliorare la gestione degli errori nell'ambito dei vari processi di crittografia dei file di licenze.

In secondo luogo, nell'Auditing System, è opportuno esplorare la possibilità di suddividere la generazione dei documenti PDF in sotto documenti più piccoli basati su intervalli di tempo regolari, come ad esempio i giorni, sia per accelerare l'operazione di apertura del singolo file che per velocizzare il processo di generazione stesso, in quanto queste sotto generazioni potrebbero essere effettuate in parallelo. Questo approccio potrebbe comportare un calcolo più oneroso e possibilmente un'occupazione di spazio maggiore a causa della creazione

di un maggior numero di PDF, ma potrebbe migliorare significativamente la velocità di apertura del file, che è inversamente proporzionale al numero di pagine del PDF.

Infine, per quanto riguarda l'offuscamento, sarebbe interessante testare lo strumento di offuscamento Zelix KlassMaster con software di deoffuscamento e reverse engineering professionali per valutarne ulteriormente le qualità, in quanto si ricorda che esso è stato l'unico in grado di generare un codice sorgente protetto non deoffuscabile dagli strumenti open source utilizzati. Questo test potrebbe aiutare a identificare eventuali punti deboli o vulnerabilità nel suo processo di offuscamento e a determinare la vera efficacia dello stesso nel proteggere il codice sorgente da accessi o modifiche non autorizzati.

Bibliografia e Sitografia

1. **Nazarevich, Dmitry.** Vantaggi e svantaggi di Java. *Innowise.* [Online] 20 Dicembre 2021. https://innowise-group.com/it/blog/benefits-and-drawbacks-of-java/.

2. **Cheema, Ajaypal.** What is Model-View-Controller? *Ajaypal Cheema Dot Com.* [Online] 30 Dicembre 2022. https://www.ajaypalcheema.com/mvc/.

3. **Reales, Jose Maria Valera.** To Mock or Not to Mock. *Medium.* [Online] 11 Gennaio 2021. https://chemaclass.medium.com/to-mock-or-not-to-mock-af995072b22e.

4. **Rao, Divya.** Understanding logging in Spring Boot. *Medium.* [Online] 25 Luglio 2022. https://medium.com/javarevisited/understanding-logging-in-spring-boot-ac0fd79177b4.

5. **Spilca, Laurentiu.** *Spring Security in Action, Second Edition.* s.l. : Manning, 2023. 9781633437975.

6. **Beschokov, Mukhadin.** What is the difference between CSRF and XSS? *Wallarm.* [Online] 20 Settembre 2022. https://www.wallarm.com/what/what-is-the-difference-between-csrf-and-xss.

7. **Open Web Application Security Project.** OWASP Top Ten. *OWASP.* [Online] 2021. https://owasp.org/www-project-top-ten/#.

8. **Naik, Varun.** CSRF, CORS, and HTTP Security headers Demystified. *Vnaik.com.* [Online] 2019. https://blog.vnaik.com/posts/web-attacks.html.

9. **Governo degli Stati Uniti.** Why HTTPS for Everything? . *The HTTPS-Only Standard .* [Online] https://https.cio.gov/everything/.

10. **Community, Wikipedia.** Galois/Counter Mode. *Wikipedia.* [Online] 15 Maggio 2023. https://en.wikipedia.org/wiki/Galois/Counter_Mode.

11. **MongoDB, Inc.** Cos'è MongoDB? *MongoDB.* [Online] [Riportato: 6 Luglio 2023.] https://www.mongodb.com/it-it/what-is-mongodb.

12. **Rahim, Javeria.** MongoDB vs MySQL: Which One Should You Choose? *Astera.* [Online] 5 Settembre 2022. https://www.astera.com/type/blog/mongodb-vs-mysql/.

13. **Tutorials Points.** MongoDB - GridFS. *Tutorials Points.* [Online] [Riportato: 6 Luglio 2023.] https://www.tutorialspoint.com/mongodb/mongodb_gridfs.htm#.

14. **Fevoryx.** Centralized Authentication Using FreeIPA Directory Server. *Linux System Administrator's Blog.* [Online] 1 Dicembre 2018. https://linuxgurublog.wordpress.com/2017/10/18/centralized-authentication-using-freeipa-directory-server-part-1/.

15. **Team, FreeIPA.** What is FreeIPA? *FreeIPA Documentation.* [Online] 2023. https://freeipa.org/page/About.

16. —. Docker. *FreeIPA Documentation.* [Online] 2023. https://freeipa.org/page/Docker.

17. **connect2id.** LDAP user authentication explained. *connect2id.* [Online] [Riportato: 6 Luglio 2023.] https://connect2id.com/products/ldapauth/auth-explained.

18. **Olawanle, Joel.** A Complete Guide to Routing in React. *Hygraph.* [Online] 6 Settembre 2022. https://hygraph.com/blog/routing-in-react.

19. **Harrison, Paramanantham.** React Table: A complete guide with updates for TanStack Table. *LogRocket.* [Online] 3 Aprile 2023. https://blog.logrocket.com/react-table-complete-guide/.

20. **Murtaza, Ahmed.** React Performance Optimization: useMemo vs useCallback . *DEV.* [Online] 8 Gennaio 2023. https://dev.to/ahmedgmurtaza/react-performance-optimization-usememo-vs-usecallback-2p2a.

21. **Webpack.** Concepts. *Webpack.* [Online] [Riportato: 8 Luglio 2023.] https://webpack.js.org/concepts.

22. **Docker Inc.** Docker Compose overview. *Docker Docs.* [Online] 2023. https://docs.docker.com/compose/.

23. **Ingo, Mat Keep e Henrik.** Performance Best Practices: Indexing. *MongoDB.* [Online] 26 Aprile 2022.

https://www.mongodb.com/blog/post/performance-best-practices-indexing.

24. **MongoDB Inc.** Best Practices Guide for MongoDB Performance . *MongoDB.* [Online] [Riportato: 9 Luglio 2023.] https://www.mongodb.com/basics/best-practices.

25. **Pedro, Xavier.** Data aggregation with Spring Data MongoDB and Spring Boot. *Java Code Geeks.* [Online] 14 Aprile 2016. https://www.javacodegeeks.com/2016/04/data-aggregation-spring-data-mongodb-spring-boot.html.

26. **Community, Wikipedia.** Internationalization and localization. *Wikipedia.* [Online] [Riportato: 9 Luglio 2023.] https://en.wikipedia.org/wiki/Internationalization_and_localization.

27. —. PDF/A. *Wikipedia.* [Online] [Riportato: 9 Luglio 2023.] https://it.wikipedia.org/wiki/PDF/A.

28. **Admin.** Cryptography. *Coders Helpline.* [Online] 7 Febbraio 2023. https://www.codershelpline.com/courses/theorypapers/cyber-security/cryptography/.

29. **Community, Wikipedia.** Offuscamento del codice. *Wikipedia.* [Online] [Riportato: 10 Luglio 2023.] https://it.wikipedia.org/wiki/Offuscamento_del_codice.

30. —. Reverse engineering. *Wikipedia.* [Online] [Riportato: 10 Luglio 2023.] https://it.wikipedia.org/wiki/Reverse_engineering.

31. **Contributors, ProGuard.** ProGuard. *GitHub.* [Online] [Riportato: 10 Luglio 2023.] https://github.com/Guardsquare/proguard.

32. **Contributors, yGuard.** yGuard. *GitHub.* [Online] [Riportato: 11 Luglio 2023.] https://github.com/yWorks/yGuard.

33. **PreEmptive.** DashO. *PreEmptive.* [Online] [Riportato: 11 Luglio 2023.] https://www.preemptive.com/products/dasho/.

34. **Allatori.** ALLATORI JAVA OBFUSCATOR. *Allatori.* [Online] [Riportato: 11 Luglio 2023.] https://allatori.com/.

35. **Zelix.** Java Obfuscation with Heavy Duty Protection. *Zelix Klassmaster.* [Online] [Riportato: 11 Luglio 2023.] https://www.zelix.com/klassmaster/index.html.

Elenco delle Figure

Ringraziamenti

Prima di tutto, vorrei ringraziare sinceramente Antonio e Giuliano per avermi dato l'opportunità di affrontare questo lavoro di tesi e per aver condiviso con me le loro competenze, fornendomi utili suggerimenti e accompagnandomi in questa esperienza.

Vorrei esprimere la mia sincera riconoscenza a Domenico e Anna per il loro prezioso sostegno durante il mio soggiorno in questa nuova città. Mi hanno accolto a braccia aperte e mi hanno fatto sentire parte della loro famiglia. Sarò per sempre grato per la loro gentilezza e generosità.

Voglio anche ringraziare i miei cari amici Antonio, Andrea e Cristian per i meravigliosi due anni di esperienza che abbiamo condiviso. Abbiamo studiato insieme, affrontato sfide insieme e festeggiato insieme i nostri successi. Non avrei mai potuto chiedere compagni migliori per questo viaggio.

Sono profondamente debitore alla mia famiglia per il loro costante incoraggiamento e la loro motivazione durante il mio percorso accademico. A mia madre Franca, a mio padre Giuseppe e a mia sorella Teresa: grazie per avermi permesso di perseguire i miei sogni e per aver sempre creduto in me.

Grazie a tutti voi ancora una volta per il vostro sostegno.