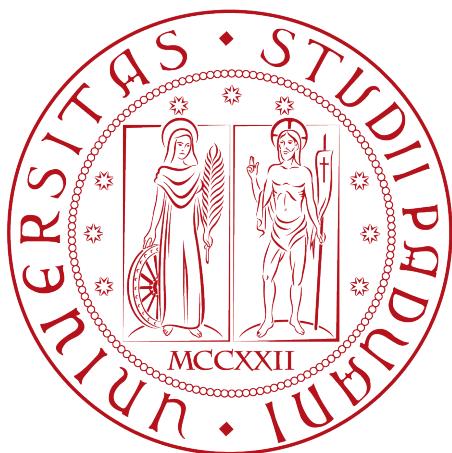


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO
LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



**Messa in sicurezza di una web app
nell'ambito di contact tracing tramite il
framework Spring Security**

Tesi di laurea triennale

Relatore

Prof. Gilberto Filè

Laureando

Alberto Cherobin

Alberto Cherobin: *Messa in sicurezza di una web app nell'ambito di contact tracing tramite il framework Spring Security*, Tesi di laurea triennale, © Settembre 2020.

L'informatica non riguarda i computer più di quanto l'astronomia riguardi i telescopi

— Edsger Dijkstra

Sommario

Il presente documento descrive il mio lavoro svolto durante il periodo di stage dal 29/06/2020 al 21/08/2020, della durata di circa trecento ore, presso l'azienda Sync Lab.

Gli obiettivi da raggiungere erano molteplici. In primo luogo era richiesto lo studio teorico dei maggiori protocolli di sicurezza informatica nell'ambito web, andando ad analizzare pro e contro di ciascuno di essi. In secondo luogo era richiesta la messa in sicurezza di una web app_G tramite l'utilizzo del framework Spring Security. Quest'ultimo è un framework del progetto Spring_G che consente di gestire in modo semplice e trasparente l'autenticazione (ovvero chi sei) e la profilazione (ovvero cosa sei autorizzato a fare) degli utenti che accedono ad una applicazione web.

Il documento è stato suddiviso in 5 capitoli:

- **Capitolo 1:** Descrizione dell'azienda e obiettivi concordati;
- **Capitolo 2:** Riassunto teorico dei principali protocolli di sicurezza informatica studiati;
- **Capitolo 3:** Descrizione progettazione sicurezza dell'app Sync Trace;
- **Capitolo 4:** Riepilogo dei test di unità effettuati;
- **Capitolo 5:** Descrizione degli strumenti e delle tecnologie adottate;
- **Capitolo 6:** Resoconto conclusivo con valutazione del percorso di tirocinio.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Gilberto Fileè, relatore della mia tesi, per l'aiuto e il sostegno fornитоми durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori, Elio e Nadia, per il sostegno, il grande aiuto e per avermi dato il tempo e la possibilità di realizzarmi in ciò che mi piace fare.

Desidero ringraziare mia sorella Ilenia, per il grande aiuto e sostegno in questa mia carriera universitaria.

Desidero ringraziare la mia fidanzata, Chiara, per essermi sempre stata vicino e avermi sostenuto in tutti questi anni.

Desidero ringraziare Daniele e Margherita, per tutte le ore di studio e divertimento passate insieme.

Desidero infine ringraziare tutta la Corte Wifi per tutti i bellissimi giorni passati insieme, di aiuto e sostegno reciproco, auguro a tutti voi il meglio.

Padova, Settembre 2020

Alberto Cherobin

Indice

1	Introduzione	1
1.1	L'azienda	1
1.2	Scelta dell'azienda	2
1.3	L'idea	2
1.4	Obiettivi	3
1.5	Way of working	3
1.6	Strumenti software di comunicazione	4
2	Protocolli di sicurezza	5
2.1	Protocollo HTTPS	5
2.2	Protocollo OAuth	7
2.2.1	OAuth 1.0	7
2.2.2	OAuth 2.0	7
2.3	Protocollo Kerberos	10
2.4	Protocollo SAML 2.0	12
3	Progettazione sicurezza per Sync Trace	15
3.1	Architettura backend dell'app	15
3.2	Il modulo Spring Security di Spring	16
3.3	Autenticazione di servizi REST con JWT	17
3.3.1	Introduzione	17
3.3.2	Autenticazione Stateless	17
3.3.3	JSON WEB TOKEN	18
3.3.4	Flusso logico di autenticazione JWT	19
3.3.5	Vantaggi e svantaggi dei JWT	20
3.4	JWT applicato a Spring Security	21
3.4.1	Configurazione Spring Security	21
3.4.2	Diagrammi di sequenza di generazione e validazione del token	25
3.4.3	Perché utilizzare i JWT?	26
3.5	Alternativa ai JWT: Paseto Token	27
3.5.1	Cos'è un Paseto Token	27
3.5.2	Struttura Paseto Token	27
3.5.3	Come funzionano i Paseto token	27
3.5.4	Che problemi risolvono?	29
3.5.5	Paseto Token VS JSON Web Token	31
4	Test effettuati	33

5	Tecnologie e strumenti	37
6	Conclusioni	39
6.1	Raggiungimento degli obiettivi	39
6.2	Conoscenze acquisite	39
6.3	Valutazione personale	40
	Glossario	41
	Bibliografia	45

Elenco delle figure

1.1	Logo azienda Sync Lab	1
1.2	Logo Sync Trace	2
1.3	Trello	4
1.4	Google Sheets	4
2.1	Protocollo HTTP	5
2.2	Protocolli HTTP e HTTPS a confronto	6
2.3	Authorization flow	8
2.4	Implicit flow	9
2.5	Client credential flow	9
2.6	Key Distribution Center (KDC)	10
2.7	Primo step protocollo Kerberos)	10
2.8	Secondo step protocollo Kerberos	11
2.9	Terzo step protocollo Kerberos	11
2.10	Logo protocollo Kerberos	11
2.11	Logo protocollo SAML 2.0	13
3.1	API gateway	15
3.2	Spring Security	17
3.3	API gateway	19
3.4	Flusso autenticazione JWT	19
3.5	Configurazione Spring Security	21
3.6	Metodo doFilterInternal	22
3.7	Autenticazione	23
3.8	Flusso authentication manager	24
3.9	Flusso generazione JWT	25
3.10	Flusso validazione JWT	25
3.11	Generazione Paseto locale	28
3.12	Generazione Paseto pubblico	28
3.13	Flusso Paseto locale	29
3.14	Flusso Paseto pubblico	30
4.1	Test con JUnit	35

Elenco delle tabelle

4.1 Elenco test di unità effettuati	33
4.1 Elenco test di unità effettuati	34

Capitolo 1

Introduzione

Organizzazione del testo

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: parola_G;
- i termini in lingua straniera o facenti parte del gergo tecnico sono evidenziati con il carattere *corsivo*.

1.1 L’azienda

Sync Lab nasce come Software house tramutatasi rapidamente in System Integrator attraverso un processo di maturazione delle competenze tecnologiche, metodologiche ed applicative nel dominio del software. L’azienda, propone sul mercato interessanti quanto innovativi prodotti [software](#), nati nel loro laboratorio di ricerca e sviluppo. Attraverso questi prodotti Sync Lab ha gradualmente conquistato significativamente fette di mercato nei seguenti settori: mobile, videosorveglianza e sicurezza delle infrastrutture informatiche aziendali.



Figura 1.1: Logo azienda Sync Lab

Il loro obiettivo è quello di supportare il Cliente nella Realizzazione, Messa in Opera e Governance di soluzioni IT, sia dal punto di vista Tecnologico, sia nel Governo del Cambiamento Organizzativo.

1.2 Scelta dell'azienda

Il primo contatto avuto con l'azienda Sync Lab è avvenuto allo StageIT 2020, effettuato interamente in modalità telematica. Il presentatore dell'azienda è stato l'ingegnere, il mio attuale tutor, Fabio Pallaro, il quale ha spiegato in maniera fluida e decisa le caratteristiche dell'azienda. Ha inoltre espresso la tendenza ad accontentare gli stagisti nelle loro preferenze nell'ambito informatico. Effettuare uno stage presso quest'ultima significava andare ad approfondire il [framework_G Spring_G](#), da me già utilizzato all'interno del progetto di ingegneria del software, andando ad approfondire il lato della sicurezza informatica.

1.3 L'idea

In seguito alla pandemia del virus COVID-19 scoppiata in Italia nel Marzo del 2020 i cittadini sono stati costretti a quarantena forzata per circa 3 mesi. Successivamente alla diminuzione dei casi riscontrati, è stata permessa la libera circolazione. In casi come questi è comodo avere a portata di mano qualcosa che possa segnalare al cittadino un eventuale rischio di contagio. L'idea dunque è stata quella di creare un'[app_G](#) in modo da poter segnalare un eventuale contatto con soggetti positivi al virus. Il tutto è molto semplice: una volta installata l'applicazione sul proprio telefono essa farà uso della tecnologia [bluetooth_G](#) per effettuare una scansione dell'area circostante al soggetto proprietario. Nel momento in cui 2 o più applicazioni vengono in contatto tra di loro avviene lo scambio di un identificativo univoco appartenente al proprietario del telefono. L'identificativo verrà salvato insieme ad un indice di contatto dato dalla distanza e dal tempo in cui è avvenuto. Nel caso in cui una persona risultasse essere positiva al virus il medico, grazie all'identificativo fornитоgli dal paziente, potrà marcarlo come positivo. Di conseguenza verranno notificati tutti gli utenti il cui indice di contatto con il paziente in questione risulti essere elevato.

Nasce dunque il progetto **Sync Trace**.



Figura 1.2: Logo Sync Trace

Lo scopo del mio stage è studiare ed implementare le componenti di [backend_G](#) che mettono in sicurezza l'applicazione Sync Trace. In questo modo potrò approfondire le mie conoscenze nell'ambito della sicurezza applicativa, in particolare sul [framework_G Spring Security](#). A livello teorico andrò ad analizzare i principali protocolli di sicurezza esistenti.

1.4 Obiettivi

Notazione

Si farà riferimento ai requisiti secondo le seguenti notazioni:

- O per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- D per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- F per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.

Obiettivi fissati

Si prevede lo svolgimento dei seguenti obiettivi:

- Obbligatori
 - $O01$: Acquisizione competenze sul framework_G Spring Security;
 - $O02$: Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma;
 - $O03$: Portare a termine l'implementazione del codice richiesto con una percentuale di superamento pari al 80%;
- Desiderabili
 - $D01$: Portare a termine l'implementazione del codice richiesti con una percentuale di superamento pari al 100%;
- Facoltativi
 - $F01$: Dare un contributo importante allo sviluppo delle altre parti di backend dell'applicativo collaborando con il gruppo di lavoro;
 - $F02$: Implementare qualche maschera prototipale con il framework_G Angular.

1.5 Way of working

Il way of working dell'azienda in merito ai processi relativi all'esperienza di stage è incentrato soprattutto sullo sviluppo del modello *agile*_G che è un modello altamente dinamico che dà maggiore importanza al software_G e meno importanza alla documentazione, in tal modo questo modello del ciclo di vita del software dà maggiore importanza alle persone e meno importanza ai processi. Infatti, la gran

parte dei modelli agili sviluppano il software in finestre di tempo limitate, chiamate iterazioni. Ogni iterazione rappresenta un piccolo incremento nelle funzionalità del software. Nel susseguirsi di queste iterazioni il **software** deve avvicinarsi sempre di più alle richieste del cliente. Nel mio caso in azienda ogni settimana veniva indetta una riunione per fare il punto sullo sviluppo del software. I metodi agili infatti preferiscono la comunicazione in tempo reale. Il tutto è stato agevolato dal fatto che ogni stagista avesse un proprio compito da svolgere, fattore molto importante nella metodologia **agile** in quanto tutto in team agile dev'essere composto da tutte le persone necessarie per terminare il progetto.

1.6 Strumenti software di comunicazione

Per mantenere un costante tracciamento delle attività svolte durante lo stage sono stati utilizzati i seguenti strumenti **software_G**:

- **Trello:** è un servizio online gratuito che permette di organizzare e gestire i propri progetti tramite la creazione di bacheche condivisibili con altri utenti;

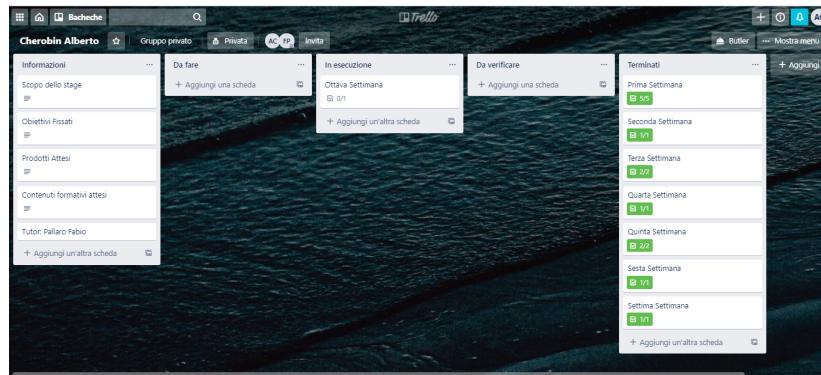


Figura 1.3: Trello

- **Google Sheets:** un foglio elettronico in cui venivano riportati giorno per giorno le attività che sono state svolte;

A	B	C	D	E	F	G	H	I	J	K
1 Data	Attività svolta	Presa visione	Note tutor							
2 29/06/2020	Studio teorico introduttivo ai JSON Web Tokens.	✓	Prima di entrare nel dettaglio di JWT studia comunque anche gli altri protocolli di sicurezza web.							
3	Ripasso teorico linguaggio di programmazione Java.	✓								
4	01/07/2020 Studio teorico a protocolli di sicurezza.	✓								
5	02/07/2020 Studio teorico OAuth 2.0.	✓								
6	02/07/2020 Studio teorico protocollo Kerberos.	✓								
7	03/07/2020 Ripasso Java.	✓								
8	04/07/2020 .	□								
9	05/07/2020 Studio tecnico protocollo SAML 2.0.	□								
10	06/07/2020 Studio funzionamento programmi Stoplight e Codewind.	✓	ok							
11	Studio Spring Core.	✓								
12	Ripasso Dependency Injection.	✓								
13	06/07/2020 Studio introduttivo JWT applicato a Spring Security.	✓								
14	06/07/2020 Studio Spring Security.	✓								
15	10/07/2020 Studio Spring Security.	✓								
16	11/07/2020 .	□								
17	12/07/2020 .	□								
18	13/07/2020 Studio Spring Security.	✓	ok							
19	14/07/2020 Studio Spring Security.	✓	ok							
20	15/07/2020 Studio Spring Security. Primo approccio ad autenticazione per generazione JWT.	✓	ok							
21	16/07/2020 Riunione di aggiornamento + Studio Spring Security.	✓								
22	17/07/2020 Inizio stesura primo progettino con Spring Security.	✓								
23	18/07/2020 Continuazione progettino con Spring Security + Effettuato primo collegamento funzionante tra progetto Spring boot e database MySQL.	✓								
24	19/07/2020 Studio Spring Security.	✓								
	20/07/2020 Inizio stesura primo progettino con Spring Security.	□								
	21/07/2020 .	□								

Figura 1.4: Google Sheets

Capitolo 2

Protocolli di sicurezza

Secondo il piano di lavoro le mie prime 2 settimane sono state dedicate allo studio dei maggiori protocolli di sicurezza nell'ambito web. Facendo riferimento alla documentazione di Spring ho optato per lo studio dei seguenti protocolli: HTTPS, OAuth 2.0, Kerberos e SAML 2.0.

2.1 Protocollo HTTPS

L' HTTP_G è un protocollo a livello applicativo usato come principale sistema per la trasmissione d'informazioni sul web, ovvero in un'architettura tipica client-server_G : il client_G esegue una richiesta e il server restituisce la risposta manda da un altro host. Nell'uso comune il client corrisponde al browser_G ed il server_G la macchina su cui risiede il sito web. Vi sono quindi due tipi di messaggi HTTP_G : messaggi richiesta e messaggi risposta. Nel caso del protocollo HTTP le connessioni vengono generalmente chiuse ogni volta che una particolare richiesta viene soddisfatta. Nello standard HTTP tutte le informazioni sono inviate in chiaro. Questo significa che tutte le informazioni scambiate tra il client e il server, che includono anche tutto ciò che l'utente va a digitare all'interno del sito, sono trasferite attraverso la rete internet pubblica. Essendo non codificate ma inviate in chiaro, tutte queste informazioni sono vulnerabili e possono essere intercettate da malintenzionati.



Figura 2.1: Protocollo HTTP

Questo è un grosso problema. Per questo motivo è stato sviluppato il protocollo HTTPS: *Secure Hypertext Transfer Protocol*. Quest'ultimo critta i dati che stanno per essere recuperati dal protocollo. Vengono usati degli algoritmi di crittografia

per rendere illeggibili i dati che stanno per essere trasferiti. In questo modo se venissero intercettati da qualcuno non sarebbero leggibili. I siti web che fanno uso di questo protocollo sono tipicamente segnalati dal browser con un lucchetto nella barra degli URL_G. La parte di sicurezza dell'HTTP utilizza il protocollo SSL (*Secure Sockets Layer*), esso fa uso di pubbliche chiavi di crittografia per mettere in sicurezza i dati.

Il protocollo SSL sostanzialmente funziona così: quando un client_G si connette al server_G in cui è contenuto il sito web, gli chiede di identificarsi; a quel punto il web server invia una copia del suo certificato SSL. Un certificato SSL è usato per autenticare l'identità di un sito web. In questo modo il client sa che ci si può fidare. Quindi il client verifica che il certificato sia affidabile e, se lo è, invierà un messaggio al web server sul fatto che si può avviare la comunicazione tra le due parti. Il web server risponderà con un riconoscimento e a questo punto la sessione SSL può procedere. Una volta superati tutti questi step può cominciare lo scambio di messaggi crittografati tra client e server.

Un altro protocollo di cui l'HTTP può fare uso è il protocollo TLS (*Transport Layer Security*), il successore dell'SSL. Sostanzialmente svolge le stesse funzionalità del protocollo SSL. Molti siti web stanno facendo uso del protocollo HTTPS indipendentemente se i dati che vengono scambiati sono sensibili o meno. Questo anche perché google_G da precedenza ai siti che fanno uso del certificato SSL e che di conseguenza risultano essere protetti.

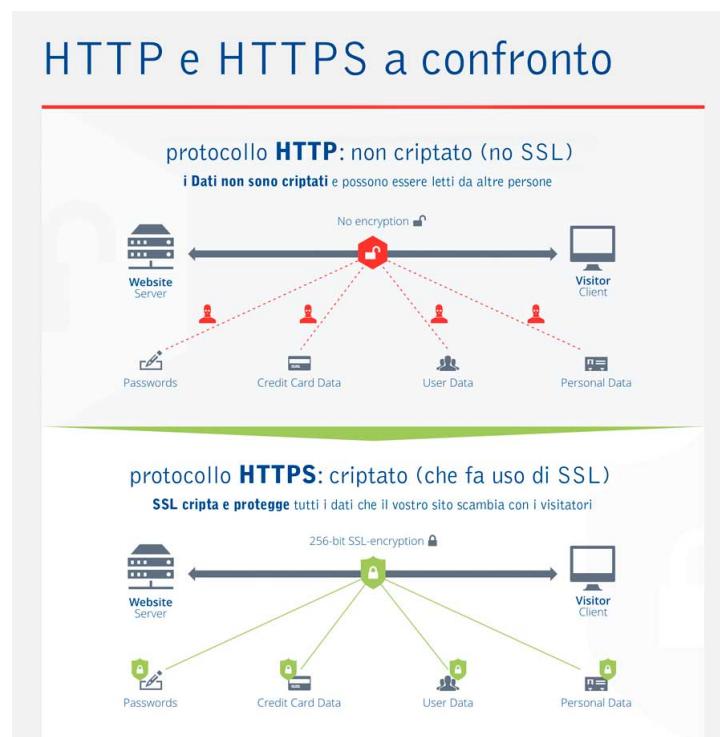


Figura 2.2: Protocolli HTTP e HTTPS a confronto

Riassunto approccio client-server tramite SSL:

1. Un client usa un browser Web per mettersi in contatto con un server Web che ospita un URL sicuro;
2. Il server Web risponde alla richiesta del client e trasmette il suo certificato digitale al browser web;
3. Il client verifica che tale certificato sia valido e corretto. I certificati sono emessi da autorità di certificazione. Questo è un punto importante perché il certificato autentica la società del server Web come legittima;
4. Una volta validato il certificato, il client genera una chiave per la sola sessione corrente, che sarà usata per cifrare tutta la comunicazione con il server Web;
5. Il client cifra la chiave di sessione con la chiave pubblica del server Web che è stata trasmessa con il certificato digitale. Usando la chiave di sessione del server Web si accerta che soltanto tale sever possa decriptare i dati;
6. A questo punto è stabilita una sessione sicura ed entrambe le parti possono comunicare attraverso un canale fidato.

2.2 Protocollo OAuth

2.2.1 OAuth 1.0

Oauth, abbreviazione di “*Open Authorization*”, è un protocollo standard aperto che consente una autorizzazione API_G sicura. Con API ci si riferisce, in questo contesto, ad una interfaccia che funge da trasmettitore di dati tra diverse applicazioni. L'autorizzazione dei trasferimenti, di dati, effettuati tramite delle API è necessaria, altrimenti sussiste il rischio che terzi non autorizzati possano intercettare i dati. Se un'applicazione dovesse pubblicare, per conto di un utente, un post su *Facebook*_G, l'utente deve fornire l'autorizzazione. L'applicazione, a sua volta, ha bisogno del permesso dell'utente per estrarre le informazioni sul suo profilo da un altro servizio. Tramite OAuth, l'utente può concedere tale delega senza dover fornire all'applicazione autorizzata l'*username* e la *password*, mantenendo quindi il controllo sui propri dati.

2.2.2 OAuth 2.0

Pubblicata nell'ottobre 2012 come revisione completa della precedente. In linea di principio, il compito di OAuth2 è lo stesso del suo predecessore: consentire all'utente, tramite l'autorizzazione API, una maggiore flessibilità mantenendo al contempo un alto livello di sicurezza. Mano a mano che i sistemi di *Facebook*_G, *Twitter* e altri operatori API diventavano sempre più complessi sono state superate molte delle carenze del protocollo originale, che rendevano la codifica e l'implementazione più difficili. OAuth 2.0 è stato inoltre integrato con processi di autorizzazione maggiormente differenziati e perfezionato nelle prestazioni. Gli sviluppatori sono

riusciti a fare in modo che OAuth2 non chieda più ad ogni fase di comunicazione i dati di accesso dell'utente (*Resource Owner*), ma solo alla prima autorizzazione della rispettiva applicazione (*client*). Un'altra innovazione degna di nota riguarda i *token*_G di accesso con scadenza più breve, che semplificano al servizio (*Resource Server*) il processo di annullamento delle autorizzazioni concesse. Inoltre, l'utente può decidere autonomamente quali autorizzazioni (*scope*) concedere a un'applicazione.

Tre tipi di flusso

1. Authorization Code Flow:

- (a) L'utente accede al servizio di terza parte e vuole usufruire delle risorse protette contenute nel resource server;
- (b) Il *client*_G (servizio di terza parte) comunica all'authorization server il desiderio da parte dell'utente di accedere alle risorse di sua proprietà all'interno del *server*_G;
- (c) L'*authorization server* comunica direttamente all'utente ora che un determinato client vuole accedere alle sue risorse e se ne consente l'autorizzazione;
- (d) L'utente conferma che è lui stesso ad aver effettuato la richiesta tramite il client;
- (e) Il client contatta di nuovo l'authorization server utilizzando il token di autorizzazione per ottenere un secondo token, chiamato token di accesso;
- (f) Il token di accesso permetterà a quel determinato client di accedere al server contenente le risorse ogni qualvolta ne abbia bisogno. Il token perciò è la prova che quel client ha accesso a quei determinati file dell'utente;
- (g) Il server verifica che il token di accesso sia valido. Una volta avutane la conferma procede all'invio delle risorse desiderate dal client.

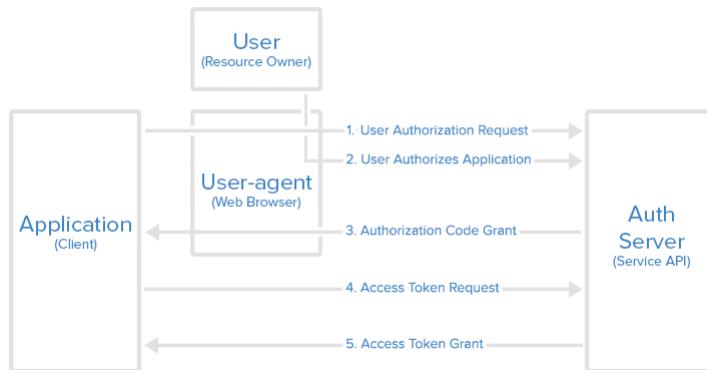


Figura 2.3: Authorization flow

2. *Implicit Flow*: uguale al primo fino alla conferma da parte dell'utente che è stato lui ad effettuare la richiesta tramite il client. A questo punto l'*authorization*

server non invierà il token di autorizzazione bensì direttamente il token di accesso. E' un flusso più semplificato e leggermente meno sicuro. Utilizzabile solitamente con certi tipi di applicazioni, soprattutto quelle basate su [JavaScript](#);

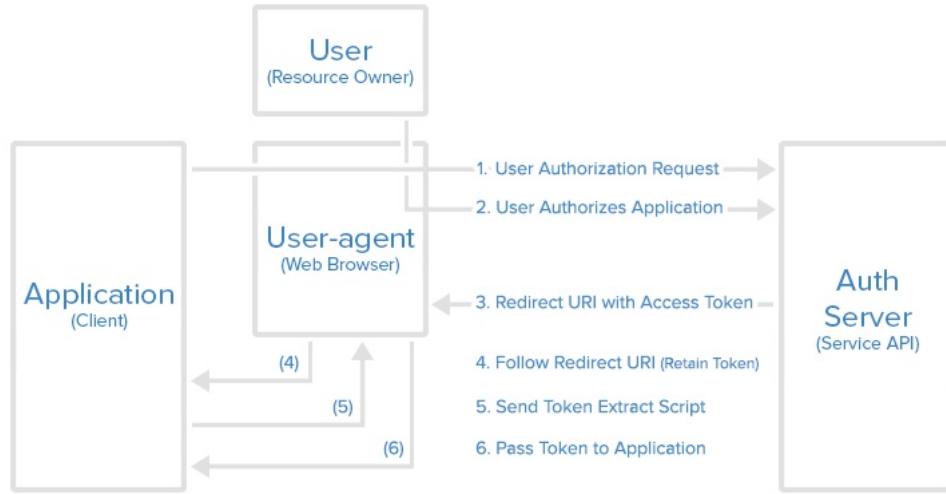


Figura 2.4: Implicit flow

3. *Client Credentials Flow*: viene utilizzato quando si è certi che ci si può fidare del client che si andrà ad utilizzare. Ovvero quando noi stessi abbiamo scritto quel client e questa è proprio la definizione di microservizio. In questo caso oltre ai microservizi esiste sempre l'*authorization server* e ogni qualvolta un microservizio vuole comunicare o richiedere determinate risorse con un altro, contatta il server di autorizzazione e gli invia una chiave speciale grazie al quale si identifica. Una volta identificato, il server di autenticazione gli invia un token di accesso che gli permetterà di effettuare la richiesta all'altro microservizio e di ottenere le risorse che cercava.

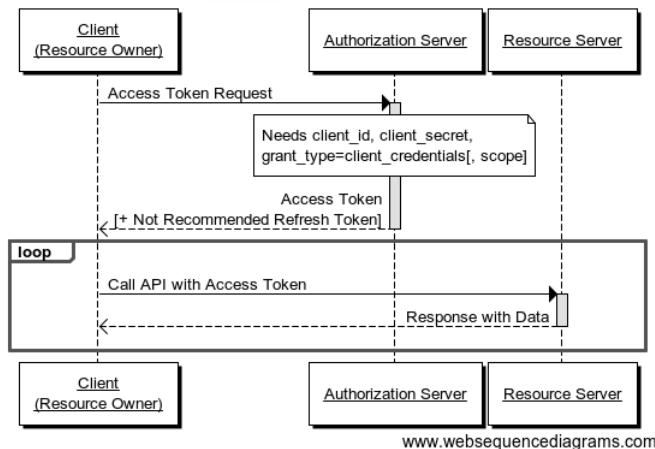


Figura 2.5: Client credential flow

2.3 Protocollo Kerberos

E' un protocollo di autenticazione per applicazioni basate su *client*/*server*. Nella maggior parte dei sistemi la password di un utente viene utilizzata come prova della tua identità, ma chiunque possa ottenere la password potrà essere scambiato per quel determinato utente e ciò non va bene. Pertanto è necessario impedire a chiunque di intercettare la password su una rete non sicura e fornire un mezzo per autenticare gli utenti affinché possano utilizzare qualsiasi servizio in qualsiasi momento. Tutto questo può essere fatto da Kerberos, progettato per due scopi: autenticazione e sicurezza.

Funzionamento del protocollo Kerberos:

1. Supponiamo che un client voglia avere accesso a risorse contenute all'interno di un file server. Con kerberos il client dev'essere prima verificato attraverso una terza componente: la *Key Distribution Center* (KDC). La KDC è composta da un *Authentication Server* (AS) e un *Ticket Granting Server* (TGS).

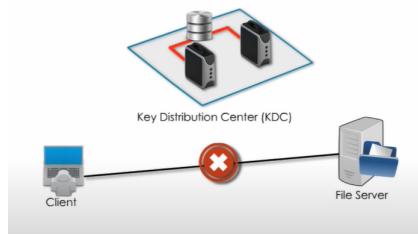


Figura 2.6: Key Distribution Center (KDC)

2. Prima di tutto il client invia una richiesta all'AS per ottenere il ticket di accesso al file server. La sua richiesta è parzialmente criptata da una chiave segreta: la password dell'utente; è importante notare che la password non verrà mai inviata nella rete non sicura ma viene usata come chiave crittografica. Quando l'AS riceve la richiesta va a recuperare la password nel database tramite l'ID dell'utente e la andrà ad usare per decriptare la richiesta appena ricevuta dal client. Quindi la password in questo caso è una chiave segreta condivisa tra il client e l'AS. In questo modo l'utente viene verificato.

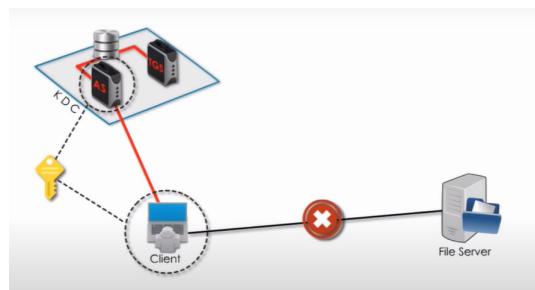


Figura 2.7: Primo step protocollo Kerberos)

3. Dopo di che l'AS invierà un *ticket* al client chiamato *Ticket Granting Ticket* (TGT), criptato con un'altra chiave segreta. Una volta ricevuto il TGT questo verrà inviato dal client al TGS per ultimare la richiesta di accesso al file server. Il TGS andrà a decriptare il ticket tramite la chiave condivisa dall'AS che a suo tempo aveva criptato il messaggio.

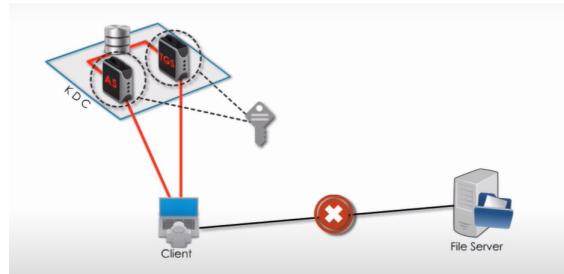


Figura 2.8: Secondo step protocollo Kerberos

4. Una volta accertata l'autenticità del ticket il TGS invierà un *token_G* al *client*, criptato con un'altra chiave segreta, quest'ultima chiave è condivisa con il client server che la userà per decriptare il token che il client gli invierà una volta ottenuto. A questo punto il *file server* permetterà al *client* di utilizzare le risorse da lui richieste per un certo periodo di tempo accordato all'interno del *token*.

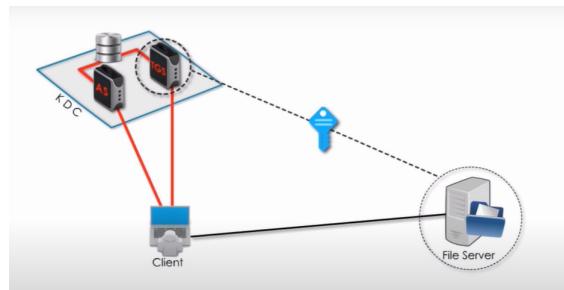


Figura 2.9: Terzo step protocollo Kerberos

Si può notare come tra tutte componenti è condivisa una chiave specifica. Infatti Kerberos è un esempio di utilizzo di chiavi private di criptazione.



Figura 2.10: Logo protocollo Kerberos

2.4 Protocollo SAML 2.0

E' uno standard aperto e viene spesso usato per offrire un *single sign-on* (SSO) alle applicazioni web. Il protocollo può essere usato sia per autenticazione che per autorizzazione. Il protocollo possiede 3 entità:

- *User Agent*: tipicamente il `browser`_G web da cui si interfaccia l'utente;
- *Service Provider* (SP): rappresenta l'applicazione a cui stai cercando di accedere;
- *Identity Provider* (IDP).

Quando avviene la configurazione di SAML viene instaurata “relazione di fiducia” tra il *Service Provider* e l'*Identity Provider*. Un utente che vuole accedere al *Service Provider* si deve prima autenticarsi nell'IDP; se riesce ad autenticarsi con successo e viene autorizzato, l'IDP genera un SAML ASSERTION che verrà poi inviata all'SP. Fin tanto che l'SP ha fiducia nell'IDP all'utente è permesso di effettuare l'accesso. Inoltre finché l'utente è già autenticato nell'IDP esso potrà effettuare l'SSO a tutte le applicazioni che hanno instaurato un collegamento “di fiducia” verso quell'IDP.

Flusso nel dettaglio

L'IDP conosce gli utenti e i suoi attributi. Anche l'SP possiede una propria conoscenza riguardo i dati degli utenti. Quando l'IDP genera un assertion, questo verrà popolato con un identificativo utente e poi inviato all'SP. L'SP a quel punto validerà l'assertion ma quest'ultima non può essere inviata in chiaro o senza la minima protezione. Per questo motivo prima di tutto l'IDP deve firmare l'assertion, in questo modo l'SP può validare l'emittente dell'assertion e in tal modo assicurarsi che sia proprio lui. A questo punto l'SP legge l'identificativo riportato nell'assertion e cercherà di mapparlo all'interno del proprio database contenente i dati utente. In questi casi la ricerca fallirà se l'identificativo utente non risulta essere presente nella sua raccolta dati utente. Affinché il mapping vada buon fine è necessario andare a stabilire delle alcune regole di integrazione, ovvero l'SP potrebbe decidere quale sia l'identificativo dell'utente e quale debba essere il suo formato. A questo punto l'IDP deve accettare e configurarsi in modo tale da poter scrivere in maniera corretta l'identificativo all'interno dell'assertion. A questo punto entrambe le parti possiedono la stessa configurazione e il contenuto delle assertion può essere mappato su un oggetto di tipo utente contenuto nella raccolta dati personale del SP. In tal modo l'SP può permettere l'accesso.

La configurazione o le regole per l'integrazione sono il punto centrale per stabilire con successo una federazione SAML. Queste configurazioni possono essere inserite manualmente all'interno del SP o IDP ma spesso i requisiti e le capacità sono raccolte all'interno di un file `XML`_G, quest'ultimo contiene le impostazioni e il certificato del sistema. A quel punto i file XML vengono scambiati tra i due attori per configurare il tutto. Lo scambio di questi file contenenti metadati è quello che va a stabilire la fiducia tra i due attori.

Flusso di autenticazione utente

Due modi per inizializzare il flusso di autenticazione:

1. *IDP-initiated*: in questo caso il flusso dell'utente comincia accedendo all'IDP. All'utente viene richiesta l'autenticazione e, una volta fatto, l'utente potrà richiedere un servizio. Se l'utente viene autorizzato l'IDP genera una *saml assertion*. Utilizzando l'agente utente (*user agent*) l'*assertion* è inviata all'SP tramite messaggio post. E' lo *user agent* che si comporta come un meccanismo di trasporto per l'*assertion*. L'SP verifica l'*assertion*, lo mappa con il corrispondente utente e così la sessione tra lo *user agent* e l'SP potrà cominciare.
2. *SP-initiated*: in questo caso il flusso comincia dall'utente che contatta l'SP. Se l'utente non è autenticato l'SP reindirizza l'utente all'IDP utilizzando una richiesta di autenticazione. Una volta validato l'utente, l'IDP genera la *saml assertion*. Come prima l'*assertion* viene inviata allo *user agent* e poi all'SP. A questo punto la sessione tra SP e *user agent* può iniziare.



Figura 2.11: Logo protocollo SAML 2.0

Capitolo 3

Progettazione sicurezza per Sync Trace

Nel seguente capitolo andrò a descrivere gli strumenti e la progettazione con cui è stata scritta la sicurezza nella web app Sync Trace.

3.1 Architettura backend dell'app

L'architettura a backend del progetto Sync Trace è a microservizi, ognuno contenente una business logic atta a soddisfare un certo tipo di richieste. Il modo più facile per mettere in comunicazione il lato frontend con quello a backend, costituito da microservizi, è quello di effettuare delle connessioni dirette a ciascuno di essi. Questa soluzione necessita di un gran numero di connessioni a seconda della quantità dei microservizi ed inoltre quest'ultimi dovranno esporre pubblicamente il proprio IP_G ad internet. Tutto ciò pone dei problemi, sia in fatto di sicurezza, dovuta all'esposizione degli indirizzi IP al mondo esterno, sia in fase di latenza, ovvero il tempo che intercorre tra l'invio di una richiesta ed una risposta tenderà ad essere sempre più alto.

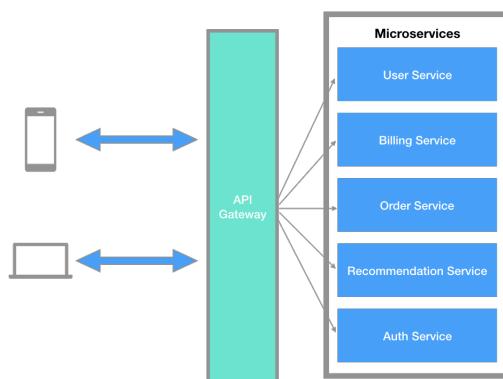


Figura 3.1: API gateway

Per far fronte a queste due problematiche si è deciso di utilizzare un $API\ gateway_G$. Quest'ultimo non è altro che un microservizio interposto tra il frontend e il backend.

In questo modo il frontend dovrà effettuare le richieste all'API *gateway* e quest'ultimo le reindirizzerà al microservizio apposito. Tale soluzione permette di risolvere il problema della sicurezza in quanto i microservizi e il *gateway* possono risiedere all'interno di un network sicuro, in questo modo gli IP dei microservizi saranno resi privati e non più pubblici; l'unico IP esposto ad Internet è quello del API-*gateway*. Per quanto riguarda la latenza il frontend comunicherà attraverso l'API *gateway*, stabilendo così sempre due sole connessioni, ovvero di richiesta e risposta; sarà l'API *gateway* a smistare le richieste al giusto microservizio, ed essendo tutti all'interno dello stesso *network*_G sicuro i microservizi usufruiranno di una connessione internet molto più rapida del sistema precedente. Di conseguenza il frontend non effettuerà più richieste multiple ai microservizi e le informazioni di riposta non saranno più aggregate a livello di frontend ma al livello dell'API gateway che poi reindirizzerà al frontend. Tutto ciò semplifica anche la parte di autenticazione e autorizzazione. Se l'API *gateway* non ci fosse, sarebbe stato necessario implementarle all'interno di ogni singolo microservizio un modulo di sicurezza. In questo modo invece l'autenticazione viene gestita da un solo microservizio, ovvero il **registration-service**, il quale provvederà alla generazione del *token* nel caso di autenticazione avvenuta con successo. Per quanto riguarda l'autorizzazione è stato sufficiente implementare un filtro all'interno dell'API *gateway* per intercettare tutte le richieste in arrivo da frontend. Nel caso nell'header_G della richiesta sia presente un token, questo verrà analizzato per verificarne l'autenticità o meno.

3.2 Il modulo Spring Security di Spring

Provvederò ora alla descrizione del normale funzionamento del **framework**_G Spring Security. Aggiungendo la dipendenza di Spring Security ad un progetto spring boot, comincerà ad intercettare qualsiasi richiesta in input. Il modo con cui la sicurezza viene implementata è attraverso l'immissione di un filtro. Ovvero un costrutto all'interno del server, in cui la web app è contenuta, che permette l'intercetto di una qualsiasi richiesta in arrivo. Un filtro garantisce l'opportunità di processare o manipolare le richieste prima che vengano gestite internamente da un **servlet**, ovvero oggetti scritti in linguaggio **Java**_G che operano all'interno di un *server* web. I filtri possono essere applicati ad una vasta gamma di url, in quanto possono essere decisi arbitrariamente dal programmatore. Vi è infatti un filtro principale chiamato **delegated filter proxy** il cui compito risiede nel delegare il lavoro a filtri più specifici e in grado di soddisfare la richiesta data in input. Uno di questi filtri avvierà il processo di autenticazione dell'utente. In un tipico processo di autenticazione in input vengono fornite le credenziali dell'utente restituendo in output l'esito, ovvero se l'autenticazione è andata a buon fine o meno. In Spring Security viene dato in output quello che viene chiamato **Principal**, ovvero le informazioni sull'utente loggato. Nello specifico, in Spring Security, è necessario fornire un tipo *authenticate*; quest'ultimo non è altro che un'interfaccia interna a Spring che in input contiene le credenziali dell'utente e in output i Principal di quest'ultimo. L'autenticazione effettiva viene affidata all'interfaccia **AuthenticationProvider**, della quale è necessario effettuare l'overload del metodo *authenticate()* che effettuerà le dovute operazioni

di controllo. E' possibile che all'interno di un progetto ci siano più meccanismi di autenticazione e di conseguenza più di un *AuthenticationProvider*. Per la gestione di quest'ultimi è presente un'interfaccia specifica chiamata ***AuthenticationManager*** e anch'essa offre la possibilità di overload della classe *authenticate()*. Affinché questa interfaccia possa capire quale sia il meccanismo di ciascun *Provider*, quest'ultimi sono forniti di un metodo ***support()*** destinato, appunto, a tale funzione. I *Provider* effettueranno l'accesso al database per verificare la correttezza delle credenziali ricevute, ottenendo così, in caso di risponso positivo, un oggetto User. Per fare ciò il *Provider* necessita di una classe chiamata ***UserDetailsService*** la quale contiene il metodo ***loadUserByUsername()*** per l'effettivo confronto delle credenziali. L'oggetto di tipo *Authenticate* con all'interno il Principal dell'utente autenticato verrà poi salvato all'interno di ***thread_G*** locale all'interno del contesto della sicurezza. Il tutto gestito da un altro apposito filtro che ad ogni richiesta di autorizzazione andrà a settare il *thread* locale con i dati contenuti all'interno del *Principal*.

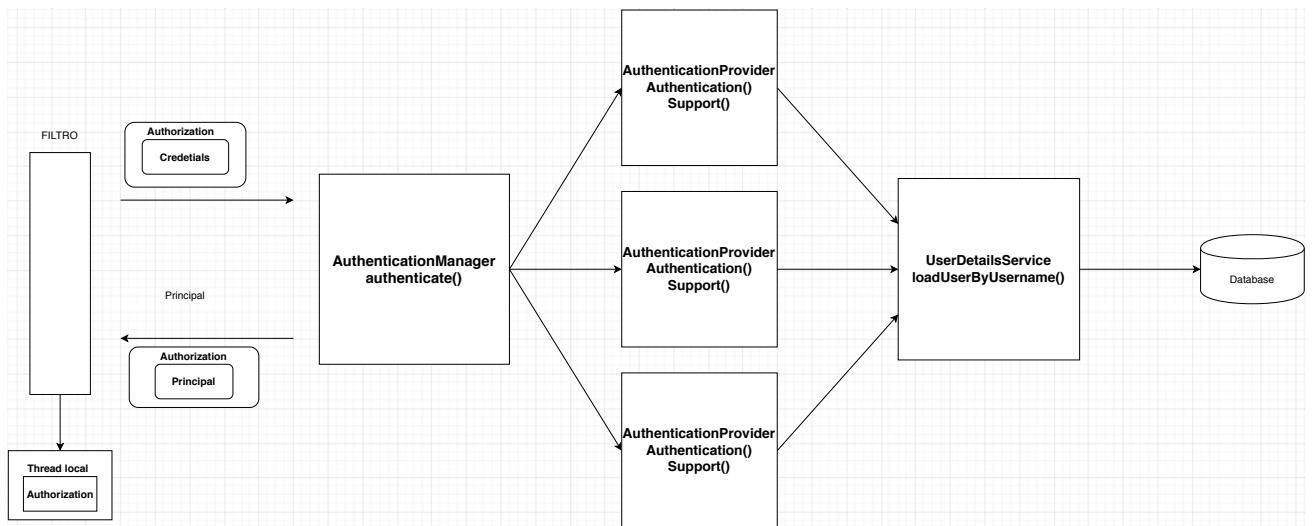


Figura 3.2: Spring Security

3.3 Autenticazione di servizi REST con JWT

3.3.1 Introduzione

Nel *web_G* al giorno d'oggi vi è sempre più l'esigenza di restringere l'accessibilità di una o più pagine di un applicazione web ai soli utenti autenticati o ad una ristretta cerchia di utenti (ad es. agli utenti amministratori). Questa necessità si presenta anche nel mondo delle *API_G*.

3.3.2 Autenticazione Stateless

Un servizio *REST_G* è per definizione *stateless_G*, ovvero le interazioni tra *client_G* e server devono essere senza stato. La gestione di quest'ultimo avviene sul *client*, ottimizzando le prestazioni del server che non ne deve curare lo stato. Una architettura *RESTful_G* che si approccia in maniera stateless rende quindi inappropriato

l'utilizzo dei *cookie_G* di sessione per il meccanismo di autenticazione.

La soluzione stateless piu' semplice è l'autenticazione BASIC: ad ogni chiamata del client viene passato un *token_G* in un *header_G* http contenente le credenziali in modo che il server ad ogni chiamata possa autenticare il *client*. In questo modo si mantiene *stateless_G* il *backend_G* RESTful ma si ottiene un *overhead_G* dovuto al fatto che il server debba autenticare ogni richiesta. Questa soluzione non è quindi adatta a sistemi di produzione. Si immagini infatti di dover ri-eseguire ad ogni richiesta le *query_G* di autenticazione sul database_G di un sistema con migliaia di utenti online nel singolo istante: sarebbe alquanto impegnativo. Una soluzione completamente *Stateless*, scalabile e con nessuno *overhead* è quella *JWT_G* (JSON WEB TOKEN).

3.3.3 JSON WEB TOKEN

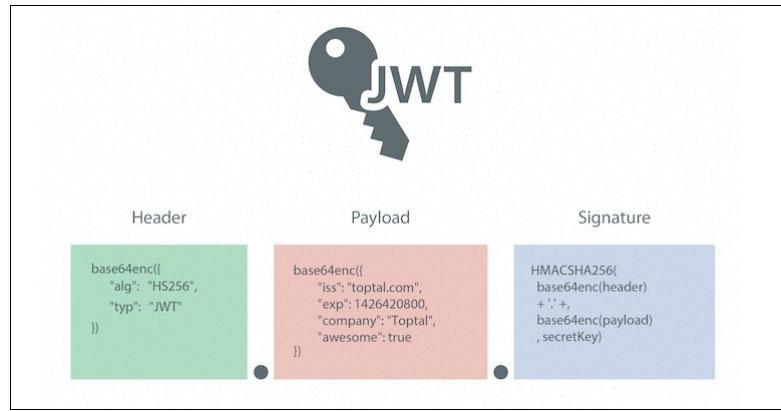
Il JSON WEB TOKEN, abbreviato più comunemente con l'acronimo di JWT, è uno standard che viene usato per gestire le richieste tra due parti. Esso non è altro che un *security token* (una stringa) che funge da contenitore per le *claims* degli utenti. I *claims* sono informazioni di un utente: le generalità dell'utente loggato (ad esempio mail, nome, cognome), la scadenza del token, i privilegi dell'utente (ovvero se è admin o meno) ed altre informazioni personalizzabili a seconda del contesto applicativo. Tali informazioni sono firmate da parte del server secondo la specifica JSON Web Signature (JWS), in questo modo il server è in grado di riconoscere se sono state generate da lui o meno.

Un token è suddiviso in tre parti:

- **Header:** è la prima parte del token che si riferisce ad un *JSON_G* che contiene le due voci "typ" e "alg". Il primo, in questo caso, avrà come valore "JWT"; mentre il campo "alg" contiene il nome dell'algoritmo usato per la codifica del token. Nel mio caso sarà presente l'algoritmo HS512;
- **Payload:** è la seconda parte del *token_G* e consiste nel corpo vero e proprio contenente dei dati, in base al contesto, in formato json e codificati in base64. Il mio payload codificato risulterà essere così:

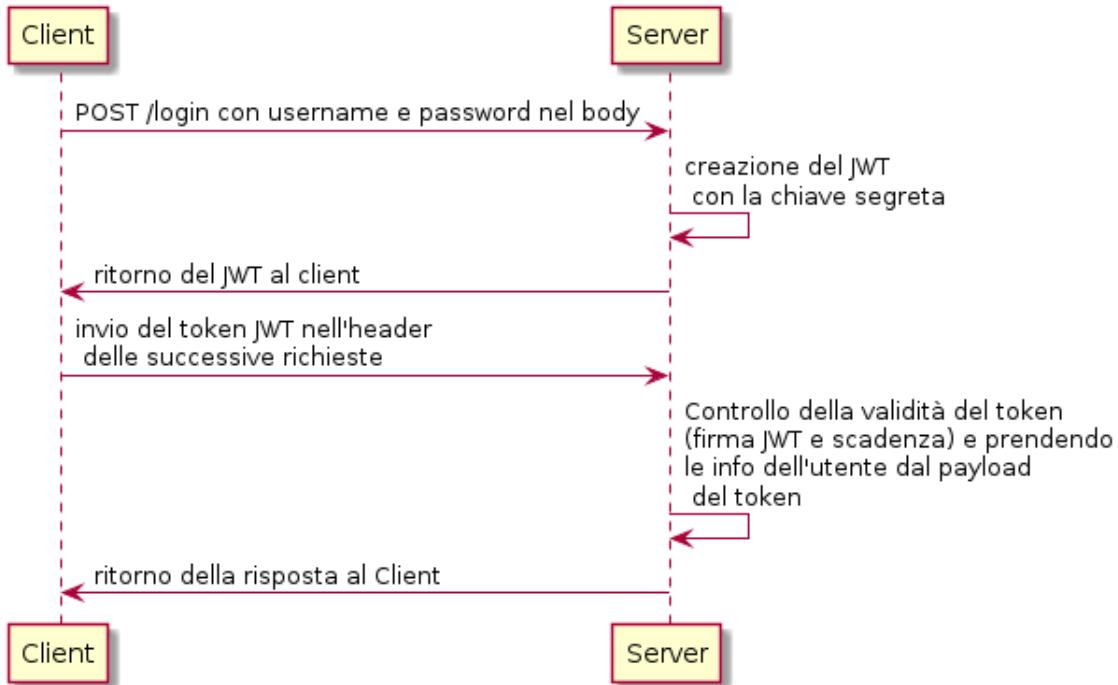
```
{
  "id" : "1",
  "role" : "medico"
}
```

- **Signature:** è la terza parte del token e consiste nella firma. Quest'ultima è il risultato di una funzione hash 512 che prende in input la codifica base64 dell'header concatenandola con un punto alla codifica base64 del payload. Il tutto viene codificato con la "chiave segreta" che solo il server conosce.

**Figura 3.3:** API gateway

Da tutto ciò si può evincere che i JWT possono essere un ottimo modo per trasportare all'interno di ogni richiesta HTTP delle informazioni legate all'autenticazione di un utente in modo sicuro. Il server grazie al campo *signature* è in grado di capire se tali informazioni sono state da lui generate, garantendo così che tale richiesta sia da considerarsi relativa ad un utente autenticato.

3.3.4 Flusso logico di autenticazione JWT

**Figura 3.4:** Flusso autenticazione JWT

- Il **client**, utilizzando l'endpoint di login, invia le proprie credenziali al server per autenticarsi;

2. Il *server* verifica che le credenziali inviate siano corrette. In caso positivo costruisce un *token_G* utilizzando la chiave segreta e inserendo nel *payload* del *token* le informazioni utente e le date di creazione e scadenza del *token* stesso. Quest'ultimo verrà firmato e autenticato con la chiave segreta conosciuta solo dal server;
3. Tale *token* viene inviato nella risposta all'interno di un header *HTTP_G*;
4. Nelle successive richieste il client inoltrerà sempre nell'apposito header il *token* restituito nel punto 3. Tale token funge da “lasciapassare” per garantire i futuri accessi;
5. Ad ogni richiesta il server si assicurerà che il *token* sia stato firmato e autenticato con la sua chiave segreta ed infine che non sia scaduto. Poiché il *token* contiene al suo interno il *payload* con le informazioni necessarie all'autenticazione, il server eviterà di fare *query_G* ogni volta per verificare a quale utente corrisponde;
6. Passati i controlli descritti dal punto 5, il server tratterà l'utente come autenticato e gestirà la richiesta, altrimenti risponderà a quest'ultima con un errore di autenticazione (401: non autorizzato).

3.3.5 Vantaggi e svantaggi dei JWT

- **Vantaggi**
 - Nel caso un *JWT_G* venga rubato, la scadenza di quest'ultimo può limitare i danni e potrà essere facilmente rigenerato;
 - Essendo una stringa di caratteri può essere trasportato facilmente;
 - Il payload è fortemente personalizzabile, quindi può contenere qualsiasi coppia campo-valore si desideri;
- **Svantaggi**
 - Per dare il via alla generazione di un *JWT* è necessario sempre prima effettuare un processo di autenticazione;
 - Essendo il *JWT* generato in base ad una chiave segreta, conosciuta solo dal server, una volta scoperta da un utente con cattive intenzioni questo potrà accedere a dati sensibili. E' bene dunque effettuare dei periodici cambiamenti al valore della chiave segreta;
 - Essendo *stateless_G* non è possibile sapere quali utenti siano attualmente loggati.

3.4 JWT applicato a Spring Security

Come spiegato nel precedente paragrafo, Spring Security, dopo una *login* avvenuta con successo, genera un *cookie*_G che viene passato automaticamente ad ogni richiesta andando a memorizzare nell'elenco delle sessioni attive le informazioni relative a quell'utente. Grazie al cookie ad ogni richiesta, grazie all'utilizzo dei filtri, è in grado di capire se l'utente sia autenticato o meno andando ad analizzare il **Principal**, ovvero le informazioni dell'utente corrente.

Per poter applicare i JWT a Spring Security è necessario:

- non utilizzare i cookie;
- validare il *token*_G prima di processare ogni richiesta e in caso di token valido, impostare nel contesto della request corrente l'autenticazione utente ricostruito dal token. Questa operazione dovrà essere processata prima delle operazioni di verifica di autenticazione eseguite dai filtri standard di Spring Security;
- ricostruire il Principal di Spring a partire dal payload del token e viceversa.

3.4.1 Configurazione Spring Security

```
@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {

    httpSecurity.csrf().disable()
        .cors().and()
        .authorizeRequests().antMatchers( ...antPatterns: "/users", "/login").permitAll() ExpressionUrl
        .anyRequest().authenticated().and() HttpSecurity
        .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint) ExceptionHand
        .and().sessionManagement() SessionManagementConfigurer<HttpSecurity>
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
}
```

Figura 3.5: Configurazione Spring Security

Nella porzione di codice appena postata viene configurato Spring Security. Attraverso l'*override*_G del metodo *configure* vengono impostate le direttive di autenticazione:

- Alcuni *endpoint* potranno essere pubblici, il che significa che non sarà necessario essere autenticati per accedervi. In questo caso sono */login*, che permette l'autenticazione da parte di un utente e */users* che permette la registrazione da parte di un utente. Tutte le altre richieste dovranno essere processate solo se l'utente che vuole accedervi è risultato essere autenticato; ciò sta a significare che tali richieste dovranno avere nell'*header*_G *HTTP*_G un JSON Web Token;

- La session management viene imposta di tipologia *stateless_G*. In questo modo abbiamo comunicato a Spring Security il fatto che non deve creare e gestire le sessioni dell'utente.
- Nell'ultima riga di codice viene impostato il filtro `jwtRequestFilter` affinché scatti prima dei filtri di default di Spring Security, i quali hanno il compito di verificare l'autenticazione.

Andiamo ora nel dettaglio della classe sopra citata, ovvero `jwtRequestFilter`. Questa ha il compito di intercettare qualsiasi richiesta in arrivo dal `frontendG` per verificare la validità del `tokenG` e settare come autenticata la richiesta arrivata, ricostruendo i dettagli dell'utente partendo dal corpo centrale del token.

```

if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
    jwtToken = requestTokenHeader.substring(7);
    try {
        username = jwtTokenUtil.getUsernameFromToken(jwtToken);
    } catch (IllegalArgumentException e) {
        System.out.println("Unable to get JWT Token");
    } catch (ExpiredJwtException e) {
        System.out.println("JWT Token has expired");
    }
} else {
    logger.warn("JWT Token does not begin with Bearer String");
}

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

    UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new
        UsernamePasswordAuthenticationToken(
            username, credentials: null, Collections.emptyList());
    usernamePasswordAuthenticationToken
        .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

    SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
}
chain.doFilter(request, response);

```

Figura 3.6: Metodo doFilterInternal

Come si evince dal codice sopra riportato, viene verificato che all'interno dell'header della richiesta sia presente un token. Quest'ultimo inoltre dovrà essere preceduto dalla stringa "Bearer ", se così non fosse verrà eseguito il corpo dell'`else` per segnalare la mancanza. In caso contrario verrà analizzato il token per controllarne la veridicità e la scadenza. Nel momento in cui il token ha superato tutti gli appositi controlli verrà eseguito il corpo del secondo `if`, andando a settare nel contesto dell'autenticazione il token.

I metodi di verifica e di costruzione di un token sono affidati alla classe *jwtTokenUtil*, la quale contiene i seguenti metodi:

- `getUsernameFromToken`: estrae lo username dal token;
- `isTokenExpired`: verifica se il token è scaduto o meno;
- `getClaimFromToken`: estrae i claims dell'utente dal token;
- `generateToken`: avvia la costruzione del token.

I codici riportati sono stati tutti inseriti all'interno dell'*API_G gateway_G* in modo tale da poter mantenere la massima sicurezza in tutti i microservizi ad esso collegati.

L'autenticazione effettiva dell'utente viene svolta all'interno del microservizio *registration-service*. Ovvero una volta che l'utente effettua una chiamata all'endpoint `/login`, l'API gateway reindirizzerà la chiamata al microservizio *registration-service*.

```
public String authenticate(Login login) throws Exception {

    Authentication auth = null;

    try {
        auth = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(
            login.getEmail(), login.getPassword()));
    } catch (DisabledException e) {
        throw new Exception("USER_DISABLED", e);
    } catch (BadCredentialsException e) {
        throw new Exception("INVALID_CREDENTIALS", e);
    }

    final String token = jwtTokenUtil.generateToken(((UtenteUser) auth.getPrincipal()));
    System.out.println(token);
    return token;
}

@Override
public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {

    Optional<UtenteUser> user = registrationRepo.findByEmail(email);
    if (user.isEmpty()) {
        throw new UsernameNotFoundException("User not found with email: " + email);
    }
    return user.get();
}
```

Figura 3.7: Autenticazione

Come si evince dall'immagine sopra riportata, Spring fornisce una classe chiamata *AuthenticationManager* la quale, internamente, andrà a richiamare il metodo

`loadUserByUsername`, di cui è stato effettuato l'*override* subito dopo, la quale andrà ad effettuare il confronto tra le credenziali fornite dall'utente e quelle presenti nel database. In questo caso però non viene controllato l'username dell'utente ma bensì l'email, la quale dev'essere univoca all'interno sistema. E' possibile che in seguito al controllo possano essere lanciate due tipi di eccezioni nel caso l'esito del confronto risultasse essere negativo:

- Utente disabilitato;
- Credenziali non valide;

Nel caso in cui il confronto effettuato risultasse essere positivo viene avviata la costruzione del token e di conseguenza ritornato all'API gateway che lo invierà direttamente all'utente a `frontendG`.

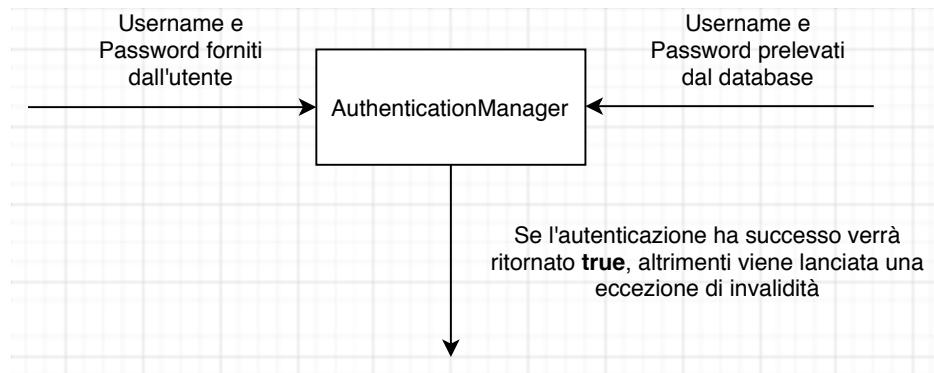


Figura 3.8: Flusso authentication manager

3.4.2 Diagrammi di sequenza di generazione e validazione del token

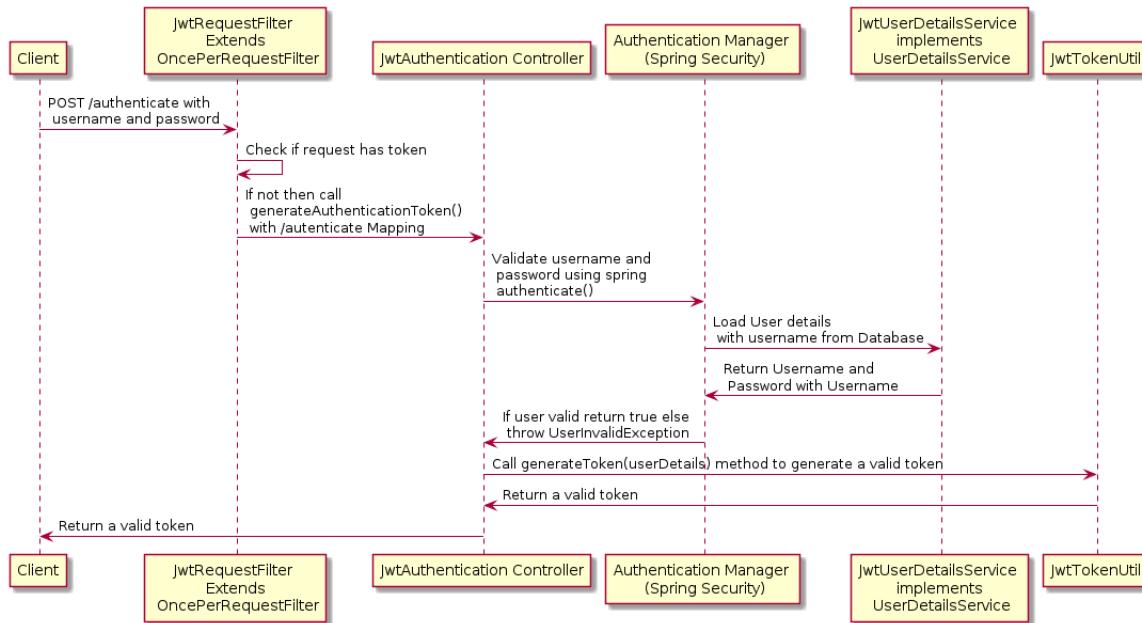


Figura 3.9: Flusso generazione JWT

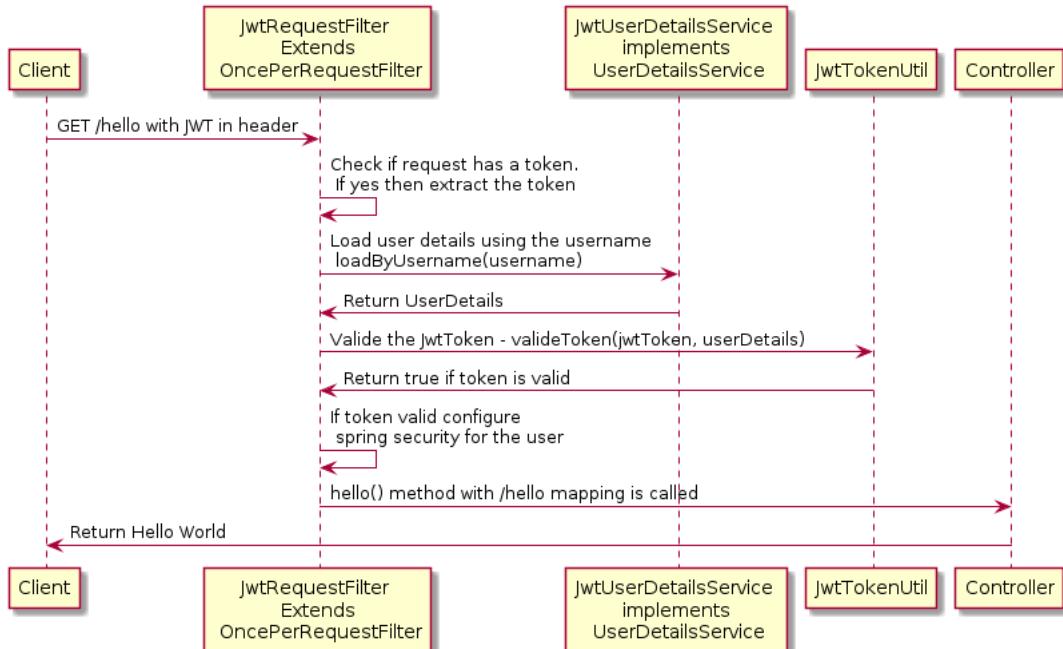


Figura 3.10: Flusso validazione JWT

3.4.3 Perché utilizzare i JWT?

Quando viene effettuata una login da parte di un utente, la maggior parte dei linguaggi di programmazione lato server, come ad esempio PHP, e framework_G, come lo stesso Spring Security, viene creata una sessione tra il server_G e il client_G per poter memorizzare informazioni dell'utente al suo interno. In questo modo sarà possibile recuperare le informazioni di autenticazione nelle successive chiamate. Qui sorge un problema: cosa succederebbe se utilizzassimo più server per bilanciare il carico del sito? Se la sessione viene creata all'interno del server A e successivamente il bilanciatore di risorse spostasse la sessione sul server B, viene persa la sessione che nel primo server conteneva tutti i dati sensibili dell'utente. (In realtà per avere sessioni condivise su più server esse non si memorizzeranno più localmente ma all'interno di uno spazio condiviso, come un database no-sql). Non tutte le soluzioni però sarebbero compatibili con questa tipologia di architettura. Se, per esempio, venisse progettato un sistema API_G in cui si vuole utilizzare un token_G per far riferimento ad una particolare sessione di autenticazione, ad ogni chiamata all'api login verrebbe generato il token e memorizzato nel database associato così all'utente. Il client passerà questo token nelle successive chiamate e il server dovrà accedere al database per verificare a quale utente è associato. In questo modo però stiamo creando un collo di bottiglia tra i vari server e il database poiché ad ogni richiesta api partirà una query_G al database per capire a quale utente è associato quel token, oltre che a creare un problema di sicurezza (il database potrebbe essere attaccato da qualche utente malintenzionato e recuperare tutti i nostri token di sessione).

A questo punto entrano in gioco i JSON Web Token. L'idea che c'è alla base è che dopo l'autenticazione, il server prepara un token all'interno del quale racchiude un payload in cui viene dichiarato in maniera esplicita chi è l'utente loggato. Dentro il token, oltre il payload viene inserita la firma dal server (costituito dal payload stesso criptato con la sua chiave segreta in codifica hash 256). Il client riceve il token e se vuole sarà libero di leggere il payload contenuto ma non potrà modificarlo poiché se lo facesse il token sarà invalidato dal server. Il client dovrà comunicare al server il token ricevuto per tutte le successive chiamate in cui è richiesta l'autenticazione. Il server riceverà il token ed estrapolerà il payload ma prima si assicurerà che il token sia stato firmato e autentificato con la sua chiave privata. Poiché il token contiene il payload con tutte le informazioni necessarie all'autenticazione (es. iduser), il server potrà evitare di passare ogni volta dal database per verificare a quale utente corrisponde quel token e ciò è ottimo per la scalabilità dell'applicazione.

3.5 Alternativa ai JWT: Paseto Token

Vado ora ad introdurre una delle ultime tecnologie nell'ambito della sicurezza informatica sviluppato negli ultimi anni: i Paseto [token](#)_G. Paseto è un protocollo relativamente nuovo, progettato nel 2018, che si sta facendo rapidamente conoscere all'interno dei contesti della sicurezza informatica. Sebbene siano ancora una tecnologia giovane sono risultati essere incredibilmente utili in quanto vanno a risolvere alcuni problemi di sicurezza generati dai JSON Web Token.

3.5.1 Cos'è un Paseto Token

Paseto è una nuova specifica che consente di creare token sicuri e senza stato che possono essere condivisi in sicurezza sul web. In sostanza, Paseto consente di prendere i dati [JSON](#)_G e condensarli in un unico token che può essere facilmente condiviso su Internet in modo sicuro e a prova di manomissione. Infatti, Paseto è stato sviluppato come un'alternativa più semplice e sicura ai [JWT](#)_G.

3.5.2 Struttura Paseto Token

Un Paseto token è costituito da 3 o, talvolta, anche 4 parti.

- La prima sezione è la **versione** del protocollo, andando a comunicare quale versione dello standard Paseto è in uso;
- La seconda sezione del Paseto è lo **scopo**. Paseto definisce solo due scopi per i token: locale o pubblico.
- La terza sezione del Paseto è il **payload**. Esso non è altro che la versione crittografata dell'agglomerato di dati JSON che si desidera inviare tramite la rete;
- La quarta sezione, non obbligatoria, è il **footer**. Può essere utilizzato per memorizzare metadati aggiuntivi, non crittografati. Ciò è utile per gli scenari in cui potrebbe essere necessario gestire eventuali rotazioni di chiavi di lettura.

Tutte le varie sezione saranno poi separate da un punto, come segue:

versione.scopo.payload.footer_opzionale

3.5.3 Come funzionano i Paseto token

Come spiegato nel paragrafo precedente, i Paseto token hanno due tipologie di scopo possibili: locale oppure pubblico.

- **Locale:** I Paseto locali vengono sempre creati e crittografati utilizzando una chiave segreta. Una libreria per sviluppatori Paseto prenderà i dati JSON che si desiderano trasmettere in modo sicuro e li crittograferà utilizzando la chiave segreta. Il Paseto locale può quindi essere de-crittografato in seguito

utilizzando la stessa chiave segreta che è stata utilizzata per crearlo. Nel caso qualcuno ottenessse un Paseto locale, non potrà estrarre nulla di utile da esso senza avere la chiave. Finché quest'ultima è al sicuro, il Paseto è al sicuro anche se condiviso pubblicamente.

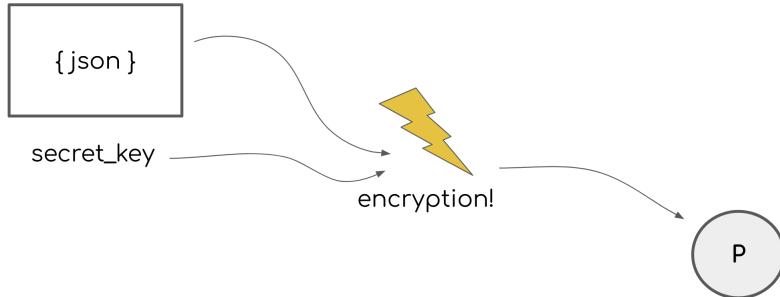


Figura 3.11: Generazione Paseto locale

Il modo con cui un Paseto locale viene creato è relativamente semplice:

- Una funzione casuale sicura genera una stringa di byte casuale;
- Un algoritmo di hashing crittografico utilizza la stringa di byte casuale come input per creare un **nonce**;
- L'header del Paseto (versione.scopo) è combinato con il footer (se presente) per creare una stringa di pre-autenticazione;
- Il payload del token viene criptato usando la chiave segreta insieme alla stringa di pre-autenticazione e finalmente viene generato il token.
- **Pubblico:** I Paseto pubblici sono perfetti per gli ambienti in cui non è possibile condividere in sicurezza una chiave segreta con tutte le parti coinvolte in una transazione. Per questo motivo essi non sono crittografati ma vengono solo firmati digitalmente. Ciò significa che se un utente malintenzionato si impossessa di un Paseto pubblico, sarà in grado di vedere tutti i dati che contiene ma non sarà in grado di modificarli senza che si venga a sapere grazie alla firma utilizzata nella creazione.

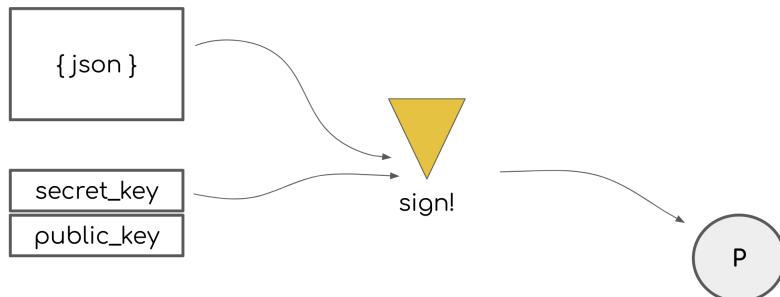


Figura 3.12: Generazione Paseto pubblico

Un Paseto token pubblico viene creato nel seguente modo:

- Viene creata una coppia di chiavi: pubblica e privata;
- L'header del Paseto (versione.scopo) viene combinato con il contenuto del payload e il footer per creare una stringa di pre-autenticazione;
- La stringa di pre-autenticazione viene digitalmente firmata dalla chiave privata;
- La stringa firmata viene inserita nella posizione del payload all'interno del token.

3.5.4 Che problemi risolvono?

I Paseto token sono stati progettati come token monouso e pensati per essere utilizzati per trasmettere in modo sicuro dati JSON sul web.

Paseto locali

Come accennato in precedenza è necessario utilizzare i Paseto locali in scenari in cui è possibile archiviare in modo sicuro una chiave segreta condivisa. Ad esempio, supponiamo che si stia creando una applicazione web che consente agli utenti di acquistare e scaricare file video. L'app potrebbe essere costituita da due servizi separati: un sito web che gestisce i flussi degli utenti e degli acquisti e un servizio di download che consente agli utenti di scaricare i file per i quali hanno pagato. In questo scenario, sia il sito web che il servizio di download vengono eseguiti su un **backend_G** lato **server_G** che può archiviare in modo sicuro una chiave segreta condivisa. Quando un utente acquista un file video, il sito web genererà un token Paseto locale che include i dettagli sull'acquisto dell'utente. Il sito web reindirizzerà quindi il **browser_G** dell'utente al servizio di download con il Paseto inserito come parametro all'interno dell'URL. Il servizio di download riceverà quindi questa richiesta, analizzerà il parametro token dall'**URL_G** e decriptograferà il Paseto locale. Il servizio di download sarà quindi in grado di convalidare l'ID di acquisto e consentire all'utente di scaricare il file specificato nei parametri della stringa di query, supponendo che Paseto lo consenta.

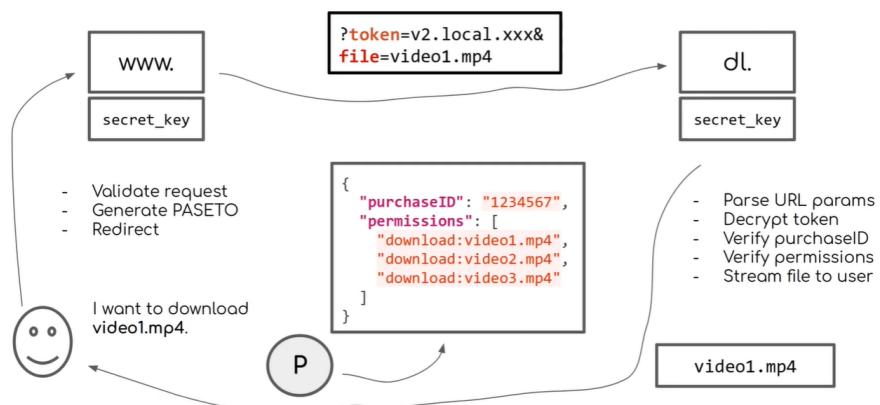


Figura 3.13: Flusso Paseto locale

Paseto Pubblici

Il caso d'uso ideale per i Paseto pubblici è trasmettere dati JSON in modo tale che i destinatari possano essere sicuri che provengono dal mittente originale. E questo è un problema particolarmente rilevante nel mondo dell'autenticazione web.

Supponiamo che si stia progettando un sito web utilizzando un protocollo di sicurezza. In questo scenario si ha:

- Un utente;
- Un sito web a cui l'utente vuole accedere;
- Un server di autorizzazione che controlla gli accessi e i permessi degli utente.

Il modo con cui l'autenticazione funziona tramite l'utilizzo dei Paseto token è il seguente:

1. L'utente clicca sul pulsante di login all'interno del sito;
2. L'utente viene reindirizzato al server di autenticazione per eseguire l'accesso;
3. L'utente inserisce in un **form_G** le sue credenziali;
4. Il server di autorizzazione convalida le credenziali dell'utente e crea un Paseto pubblico utilizzando una chiave privata a cui ha accesso solo il server di autorizzazione;
5. Il server di autorizzazione reindirizza quindi l'utente al sito Web con il Paseto pubblico come parametro URL;
6. Il sito web riceve la richiesta dell'utente e analizza il Paseto dal parametro URL del token;
7. Il sito Web convalida il Paseto e quindi crea una sessione di lunga durata per l'utente utilizzando cookie protetti;
8. L'utente è ora autenticato nel sito Web.

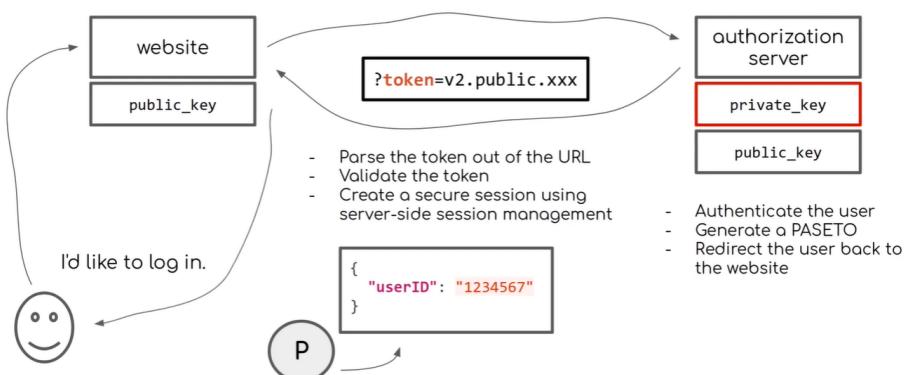


Figura 3.14: Flusso Paseto pubblico

In questo esempio, si noti che il Paseto viene utilizzato solo una volta: per informare il sito Web che l'utente si è autenticato con successo tramite il server di autorizzazione. L'altra cosa da notare è che questo flusso funziona solo perché il sito web avrà accesso alla chiave pubblica del server di autorizzazione. Questa chiave pubblica è necessaria affinché il sito web possa verificare la validità del Paseto inviatogli dal server di autorizzazione.

3.5.5 Paseto Token VS JSON Web Token

A questo punto, una domanda che sorge spontanea è: se i Paseto token sono così simili ai JWT, perché esistono? I Paseto token, difatti, sono stati progettati per aggirare vari problemi che i JWT presentano. In particolare le comunità di crittografia e sicurezza hanno criticato i JWT per:

- Essere ampiamente utilizzati in modo improprio, andando così ad influire sulla sicurezza web;
- Forzare le implementazioni ad aderire all'inserimento dell'algoritmo utilizzato all'interno dell'header (alg). Ciò consente agli aggressori di modificare il valore alg e cambiare il metodo di verifica della firma in qualcosa di diverso da quello originariamente previsto, rendendo così la falsificazione del token una possibilità reale;
- Consentire scelte di crittografia scadenti che consentono agli aggressori di attaccare i token crittografati in vari modi.

Per riassumere ad un livello elevato, le specifiche JOSE (che includono JWT, JWE, JSON Web Signatures, ecc.) sono estremamente flessibili che costringono gli sviluppatori a fare molte scelte di sicurezza e crittografia di basso livello che possono causare problemi di sicurezza catastrofici.

La specifica e le implementazioni Paseto sono state progettate per rispondere a ciascuna di queste critiche. Paseto adotta un approccio ai token di sicurezza consolidando le scelte degli sviluppatori a due scopi: si necessita di un modello di sicurezza simmetrico (token locale) o asimmetrico (token pubblico)? In base alla decisione, Paseto effettua le migliori scelte possibili per la crittografia autenticata e le firme digitali per garantire che i token rimangano protetti e non siano soggetti a vulnerabilità crittografiche.

La specifica Paseto definisce anche chiaramente come i suoi token dovrebbero e non dovrebbero essere usati nel tentativo di ridurre l'uso improprio dei token nei modi in cui le persone attualmente abusano dei JWT.

	Public-key Crypto	Encrypted by Default	Standardized	Modern Crypto	Standard Claim Checks
JWT	✓	✓	✓	✓	✓
Paseto	✓	✓ (locale)	X	✓	✓

Capitolo 4

Test effettuati

Una volta terminata la fase di codifica sono stati effettuati i test di unità che consistono nell'isolare una parte del codice e verificarne il corretto funzionamento. Gli strumenti utilizzati per fare ciò sono Postman e Mockito.

Di seguito vengono riportati in tabella tutti i test effettuati per verificare il corretto funzionamento del token_G . Ogni test può avere esito "Positivo" nel caso venga superato con successo, "Negativo" altrimenti.

Tabella 4.1: Elenco test di unità effettuati

Descrizione	Esito
Test per verificare il libero accesso alla registrazione da parte di un utente non registrato	Positivo
Test per verificare il libero accesso all'autenticazione da parte di un utente non autenticato	Positivo
Test per verificare se in seguito all'autenticazione il sistema restituisce il token	Positivo
Test per verificare se la presenza di un token non valido restituisca un'eccezione di non autorizzazione	Positivo
Test per verificare se la durata un token valido consente il normale accesso	Positivo
Test per verificare se un token valido ma scaduto nega l'accesso	Positivo
Test per verificare se le credenziali di login errate lanciano l'eccezione di credenziali non valide	Positivo
Test per verifica se la presenza di un token valido consente di visualizzare tutti gli utenti del sistema	Positivo
Test per verificare se la presenza di un token valido consente di visualizzare i dettagli di un utente specifico	Positivo
Test per verificare se la presenza di un token valido consente di eliminare un utente specifico	Positivo

Tabella 4.1: Elenco test di unità effettuati

Descrizione	Esito
Test per verificare se la presenza di un token valido consente di modificare i dati di uno specifico profilo utente	Positivo
Test per verificare se la presenza di un token valido consente di visualizzare tutti gli utenti infetti	Positivo
Test per verificare se la presenza di un token valido consente di visualizzare uno specifico utente infetto	Positivo
Test per verificare se la presenza di un token valido consente di conoscere il numero di utente infettati	Positivo
Test per verificare se la presenza di un token valido consente di aggiungere un nuovo utente infetto	Positivo
Test per verificare se la presenza di un token valido consente di modificare i dati di un utente infetto	Positivo
Test per verificare se la presenza di un token valido consente di eliminare un utente infetto	Positivo
Test per verificare se la presenza di un token valido consente di visualizzare tutti gli utenti ad alto rischio	Positivo
Test per verificare se la presenza di un token valido consente di visualizzare tutti gli utenti a medio rischio	Positivo
Test per verificare se la presenza di un token valido consente di visualizzare tutti gli utenti a basso rischio	Positivo
Test per verificare se la presenza di un token valido consente di aggiungere un utente a rischio	Positivo
Test per verificare se la presenza di un token valido consente di modificare un utente a rischio	Positivo
Test per verificare se la presenza di un token valido consente di eliminare un utente a rischio	Positivo

```
@Test
public void testGetAllUsers() throws Exception {

    RestTemplate restTemplate = new RestTemplate();
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth("eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxIiwiaWFjIjoxNTk3NDE4MDE0fQ.NA6gT3RX4gEc72JFeUB1a5Bm5jM5RP" +
        "NTs5rCCUuwLU_~V3RDi8Cdfy9pbeBRWEL~lVNFC1l9Q8_-5TT6KWQ");
    HttpEntity<String> entity = new HttpEntity<>(<body> "", headers);

    final String baseUrl = "http://localhost:" + randomServerPort + "/users";
    URI uri = new URI(baseUrl);

    ResponseEntity<String> result = restTemplate.exchange(uri, HttpMethod.GET, entity, String.class);

    UtenteUser fake = new UtenteUser(id: 0, fiscal_code: "vuoto", email: "vuoto", name: "vuoto", surname: "vuoto", password: "vuoto",
        piva: "vuoto", r_social: "vuoto", address: "vuoto", role: "vuoto", firebase: "vuoto", userid: "vuoto");
    //ResponseEntity<String> result = restTemplate.exchange(uri, HttpMethod.GET, entity, String.class);

    Assertions.assertEquals(expected: 200, result.getStatusCodeValue());
}
```

Figura 4.1: Test con JUnit

Capitolo 5

Tecnologie e strumenti

IntelliJ IDEA: è un ambiente di sviluppo integrato (IDE) per il linguaggio di programmazione Java_G . Oltre ad essere ottimizzato per quest'ultimo, è in grado di fornire un'ampia gamma di plugin_G che consentono la semplificazione del lavoro;

<https://www.jetbrains.com/idea/>

GitLab: affinché il codice fosse a disposizione di tutti gli sviluppatori, in accordo con l'azienda, è stata creata una *repository* su GitLab.

<https://about.gitlab.com/>

Git: software_G per il controllo del versionamento del codice all'interno delle *repository* G di gitLab G . Utilizzato da me tramite terminale senza l'ausilio di programmi come GitHub desktop_G o Git Kraken_G per gusto personale.

<https://git-scm.com/>

Postman: è un'applicazione che consente di costruire, testare e documentare API più velocemente. E' possibile effettuare delle chiamate API senza dover mettere mano al codice dell'applicazione, consentendo di effettuare le chiamate tramite questo plugin che fornisce un'utile interfaccia grafica. Le richieste possono essere effettuate sia verso un server locale che verso un server online impostando tutti i dati di una tipica chiamata API, dagli *headers* al *body*.

<https://www.postman.com/>

Mockito: è un *framework* G di test open source per Java rilasciato sotto licenza MIT. Il framework permette la creazione di oggetti finti (*mock object*) all'interno dei test di unità automatici al fine di testare il corretto comportamento della API.

<https://site.mockito.org/>

JUnit5: è un framework di test di unità per il linguaggio di programmazione Java.

<https://junit.org/junit5/>

Capitolo 6

Conclusioni

In questo capitolo sono riportate le considerazioni conclusive riguardanti l'attività di stage.

6.1 Raggiungimento degli obiettivi

Lo stage si è svolto rispettando i tempi prefissati e tutti gli obiettivi obbligatori e desiderabili sono stati portati a termine seguendo il piano di lavoro redatto prima dell'inizio dello stage.

- Obbligatori:
 - O01: Acquisizione competenze sulle tematiche della sicurezza informatica applicata a Spring Security;
 - O02: Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma;
 - O03: Portare a termine l'implementazione del codice richiesto con una percentuale di superamento pari al 80%;
- Desiderabili:
 - D01: Portare a termine l'implementazione del codice richiesti con una percentuale di superamento pari al 100%;

6.2 Conoscenze acquisite

Per quanto riguarda le conoscenze tecniche acquisite, sono molto soddisfatto di essermi potuto avvicinare all'ambito della sicurezza informatica, la quale nel corso di laurea è trattata in maniera marginale. Il progetto mi ha permesso di approfondire il framework_G Spring del quale conoscevo solamente il modulo Spring boot_G, utilizzato nel progetto del corso di ingegneria del software. Ma soprattutto mi ha consentito di entrare a far parte di un team numeroso e fare la conoscenza di alcune dinamiche aziendali di lavoro di gruppo fino ad allora a me sconosciute. Il riuscire a mediare le linee guida teoriche fornite dagli studi, con le esigenze reali di un'azienda, è stata, credo, un'esperienza fondamentale nella mia vita lavorativa.

6.3 Valutazione personale

Lo stage è stata un'attività molto positiva, che ha accresciuto il mio bagaglio culturale in termini di conoscenze acquisite, ma soprattutto di esperienza pratica. Ritengo che l'esperienza del tirocinio sia fondamentale anche dal punto di vista personale e non solo professionale. Questa esperienza mi ha messo alla prova in prima persona, facendo emergere i punti di forza e di debolezza della mia personalità, sui quali sicuramente andrò a lavorare per affrontare al meglio situazioni lavorative future. Mi ha dato modo di apprezzare le conoscenze acquisite durante gli studi universitari, e di evidenziarne le mancanze. Come per esempio il dover affrontare un argomento sconosciuto studiando da documentazioni e articoli invece che da un libro di testo o appunti presi a lezione. Ciò permette di accrescere la capacità di autonomia di uno studente che si approccia per la prima volta al mondo del lavoro. Osservando questa esperienza ora che essa è giunta al termine, non posso che ritenermi soddisfatto della scelta effettuata. Ritengo, quindi, l'esperienza del tirocinio positiva e molto importante per uno studente che al termine del percorso di studi, inizia ad intraprendere e a costruire quello che sarà poi il suo progetto professionale.

Glossario

Agile Nell'ingegneria del software la metodologia agile propone un approccio meno strutturato e più focalizzato sull'obiettivo di consegnare al cliente, in tempi brevi e frequentemente, software funzionante e di qualità. [3](#), [4](#)

API In informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. [7](#), [15](#), [17](#), [23](#), [26](#)

App Abbreviazione di applicazione. [2](#)

Backend All'interno di un software è la parte che consente le effettive interazioni che un utente effettua con esso. [2](#), [18](#), [29](#)

Bluetooth Standard tecnico-industriale di trasmissione dati per reti personali senza fili. [2](#)

Browser E' un'applicazione per l'acquisizione, la presentazione e la navigazione di risorse sul web. [12](#), [29](#)

Client In informatica, nell'ambito delle reti informatiche e dell'architettura logica di rete detta client-server, indica genericamente un qualunque componente software, presente tipicamente su una macchina host, che accede ai servizi o alle risorse di un'altra componente detta server, attraverso l'uso di determinati protocolli di comunicazione. [5](#), [6](#), [8](#), [10](#), [17](#), [19](#), [26](#)

Client-server In informatica il termine indica un'architettura di rete nella quale genericamente un computer client o terminale si connette ad un server per la fruizione di un certo servizio, quale ad esempio la condivisione di una certa risorsa hardware/software con altri client, appoggiandosi alla sottostante architettura protocollare. [5](#)

Cookie Vengono utilizzati dalle applicazioni web lato server per archiviare e recuperare informazioni a lungo termine sul lato client. [21](#)

Facebook E' una rete sociale, lanciato a scopo commerciale il 4 febbraio 2004, posseduto e gestito dalla società Facebook Inc. [7](#)

Form Indica la parte di interfaccia utente di un'applicazione web che consente all'utente client di inserire e inviare al web server/application server uno o più dati liberamente digitati dallo stesso sulla tastiera attraverso l'uso di componenti grafici detti widget.. [30](#)

Framework in informatica e specificamente nello sviluppo software, è un'architettura logica di supporto (spesso un'implementazione logica di un particolare design pattern) sul quale un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. [2](#), [3](#), [16](#), [26](#), [35](#), [37](#)

Frontend Nel campo della progettazione software e sviluppo software il front end è la parte di un sistema software che gestisce l'interazione con l'utente o con sistemi esterni che producono dati di ingresso. [22](#), [24](#)

Gateway E' un dispositivo di rete che collega due reti informatiche di tipo diverso operando sia al livello di rete che ai livelli superiori, del modello ISO/OSI. Il suo scopo principale è quello di veicolare i pacchetti di rete all'esterno di una rete locale (LAN). [15](#), [23](#)

Git Kraken Software dotato di interfaccia utente per l'utilizzo della tecnologia git. [35](#)

GitHub desktop Software dotato di interfaccia utente per l'utilizzo della tecnologia git. [35](#)

Google E' un motore di ricerca per Internet il cui dominio è stato registrato il 15 settembre 1997. [6](#)

Header In informatica e nella commutazione di pacchetto, indica quella parte di un pacchetto che contiene informazioni di controllo necessarie al funzionamento della rete, cioè le informazioni di protocollo aggiunte di strato in strato, in opposizione al carico pagante, ovvero ai dati utili che vengono trasportati tra gli utilizzatori della rete. [18](#), [21](#)

HTTP HyperText Transfer Protocol. [5](#), [20](#), [21](#)

IP In informatica e nelle telecomunicazioni l'indirizzo IP è colui che identifica univocamente un dispositivo detto host collegato a una rete informatica che utilizza l'Internet Protocol come protocollo di rete per l'instradamento/indirizzamento, inserito dunque nell'intestazione (header) del datagramma IP per l'indirizzamento tramite appunto il protocollo IP. Esso equivale all'indirizzo stradale o al numero telefonico, infatti sono informazioni complete ed univoche a livello mondiale, similmente all'indirizzo IP. [15](#)

Java In informatica è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione, specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione. [16](#), [35](#)

JavaScript In informatica è un linguaggio di programmazione orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client (esteso poi anche al lato server) per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso. [9](#)

JSON Acronimo di JavaScript Object Notation, è un formato adatto all'interscambio di dati fra applicazioni client/server. [18](#), [27](#)

JWT JSON Web Token. [18](#), [20](#), [27](#)

Network In informatica e telecomunicazioni, è una tipologia di rete di telecomunicazioni a commutazione di pacchetto caratterizzata da un insieme di dispositivi hardware con opportuni software di commutazione, ossia nodi di commutazione collegati l'uno con l'altro da appositi canali di comunicazione (link), tali da fornire un servizio di comunicazione che permette lo scambio e la condivisione di dati e la comunicazione tra più utenti o dispositivi distribuiti o terminali (host): i dati vengono trasmessi e trasferiti sotto forma di pacchetti dati (PDU, Packet Data Unit), composte da un header (che contiene i dati per il recapito del messaggio) e un body (che contiene il corpo del messaggio), il tutto regolato da precisi protocolli di rete. [16](#)

Nonce In crittografia il termine nonce indica un numero, generalmente casuale o pseudo-casuale, che ha un utilizzo unico. Nonce deriva infatti dall'espressione inglese for the nonce, che significa appunto "per l'occasione". Un nonce viene utilizzato spesso nei protocolli di autenticazione per assicurare che i dati scambiati nelle vecchie comunicazioni non possano essere riutilizzati in attacchi di tipo replay attack.. [28](#)

Overhead In informatica, è utilizzato per definire le risorse accessorie, richieste in sovrappiù rispetto a quelle strettamente necessarie per ottenere un determinato scopo in seguito all'introduzione di un metodo o di un processo più evoluto o più generale. [18](#)

Override Nella programmazione ad oggetti è l'operazione di riscrittura di un metodo ereditato. [21](#)

Plugin In campo informatico è un programma non autonomo che interagisce con un altro programma per ampliarne o estenderne le funzionalità originarie. [35](#)

Query In informatica il termine query viene utilizzato per indicare l'interrogazione da parte di un utente di un database, strutturato tipicamente secondo il modello relazionale, per compiere determinate operazioni sui dati. [18](#), [20](#)

Repository In informatica, è un ambiente di un sistema informativo in cui vengono gestiti i metadati. [35](#)

REST rappresenta un sistema di trasmissione di dati su HTTP senza ulteriori livelli, quali ad esempio SOAP. I sistemi REST non prevedono il concetto di sessione, ovvero sono stateless, come approfondito successivamente.. [17](#)

RESTful I servizi web che sono conformi ad uno stile di architettura rest vengono chiamati servizi RESTful. [17](#)

Server in informatica e telecomunicazioni è un componente o sottosistema informatico di elaborazione e gestione del traffico di informazioni che fornisce, a livello logico e fisico, un qualunque tipo di servizio ad altre componenti (tipicamente chiamate clients, cioè clienti) che ne fanno richiesta attraverso una rete di computer, all'interno di un sistema informatico o anche direttamente in locale su un computer. [5](#), [6](#), [8](#), [10](#), [20](#), [26](#), [29](#)

Software In informatica si intendono tali il semplice dato o informazione oppure più propriamente le istruzioni di un programma codificate in linguaggio macchina o in linguaggio di programmazione (codice sorgente), memorizzate su uno o più supporti fisici, sotto forma di codice eseguibile. [1](#), [3](#), [4](#), [35](#)

Spring è un *framework open-source* per lo sviluppo di applicazioni su piattaforma Java. [2](#)

Spring boot Modulo di Spring che permette la creazione di applicazioni Web con web server integrato. [37](#)

Stateless Si tratta del principio che stabilisce la necessità di avere una comunicazione stateless, senza stato, cioè una comunicazione tra client e server in cui ciascuna richiesta è indipendente dalle altre. [18](#), [20](#), [22](#)

Token Stringa di caratteri criptata e utilizzata dagli utenti come "lascia passare" per accedere a risorse di tipologia sesibile. [8](#), [11](#), [18](#), [20–22](#), [26](#), [27](#), [33](#)

Twitter E' un servizio di notizie e microblogging fornito dalla società Twitter, Inc. [7](#)

URL Uniform Resource Locator. [6](#), [29](#)

Web World Wide Web, abbreviato con Web, è uno dei principali servizi di Internet, che permette di navigare e usufruire di un insieme molto vasto di contenuti amatoriali e professionali (multimediali e non) collegati tra loro attraverso legami (link), e di ulteriori servizi accessibili a tutti o ad una parte selezionata degli utenti di Internet. [17](#)

XML E' un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. [12](#)

Bibliografia

Riferimenti bibliografici

Andrew S. Tanenbaum, David J. Wetherall. *Reti di calcolatori, quinta edizione.*
Pearson, Settembre 2011.

Siti web consultati

A Thorough Introduction to PASETO. URL: <https://master--okta-blog.netlify.app/blog/2019/10/17/a-thorough-introduction-to-paseto>.

Alternatives to JSON Web Tokens. URL: <https://www.scottbrady91.com/JOSE/Alternatives-to-JWTs>.

API Gateway. URL: <https://www.youtube.com/watch?v=8WuVBbXsHzg>.

Italian coders spring jwt. URL: <https://github.com/dario-frongillo/spring-jwt-auth-italiancoders/>.

Kerberos: Technical Overview. URL: https://www.youtube.com/results?search_query=kerberos+protocol.

Manifesto Agile. URL: <http://agilemanifesto.org/iso/it/>.

OAuth 2.0. URL: <https://www.ionos.it/digitalguide/server/sicurezza/protocollo-oauth/>.

PASETO token. URL: <https://paseto.io/>.

Protocollo Kerberos. URL: <http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rh-it-4/ch-kerberos.html>.

SAML 2.0: Technical Overview. URL: https://www.youtube.com/watch?v=SvppXbpv-5k&ab_channel=VMwareEnd-UserComputing.

SAML 2.0. URL: <https://spring.io/projects/spring-security-saml>.

Spring Security. URL: <https://www.youtube.com/watch?v=I0poT4UxFxE&t=151s>.

Spring Security JWT. URL: <https://github.com/koushikkothagal/spring-security-jwt>.