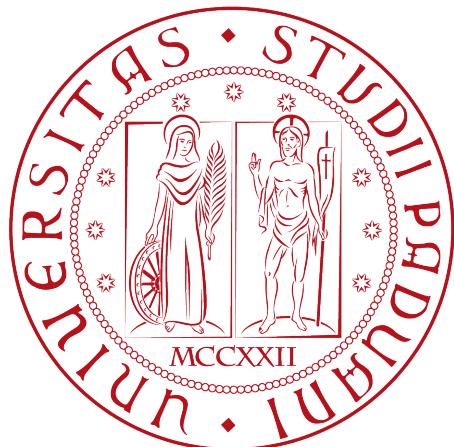


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA



**Gestione del catalogo applicativo per un
sistema di Identity and Access Management
cloud-based**

Tesi di laurea triennale

Relatore

Prof. Mauro Conti

Laureando

Matteo Di Pirro

ANNO ACCADEMICO 2015-2016

Matteo Di Pirro:

*Gestione del catalogo applicativo per un sistema di Identity and Access Management
cloud-based,*
Tesi di laurea triennale, © Settembre 2016

Dedicato alla mia famiglia

Abstract

Il presente documento riassume il lavoro svolto durante il periodo di stage, della durata di 320 ore, presso l'azienda iVoxIT S.r.l. di Padova.

Lo scopo principale del prodotto sviluppato è quello di semplificare ed automatizzare la gestione del catalogo applicativo di Monokee, un sistema di Identity as a Service (IDaaS). Nel corso del documento verranno anche esposte le basi teoriche del prodotto, come la gestione delle identità e il Single Sign-On (SSO).

In una prima fase mi sono concentrato sull'apprendimento degli strumenti e delle tecnologie da utilizzare e preso confidenza con le tematiche della gestione delle identità, dell'autenticazione e dell'autorizzazione. Dopo aver identificato le principali funzionalità richieste e compreso il funzionamento di Monokee e dell'Identity and Access Management (IAM) ho iniziato la progettazione dell'applicazione. La definizione di un'architettura che fosse estendibile, manutenibile e integrabile con quella esistente ha richiesto molti sforzi, ma il risultato è stato all'altezza della aspettativa dell'azienda e ha soddisfatto tutte i requisiti individuati.

“The modern world needs people with a complex identity who are intellectually autonomous and prepared to cope with uncertainty; who are able to tolerate ambiguity and not be driven by fear into a rigid, single-solution approach to problems, who are rational, foresighted and who look for facts; who can draw inferences and can control their behavior in the light of foreseen consequences, who are altruistic and enjoy doing for others, and who understand social forces and trends.”

— Robert Havighurst

Ringraziamenti

Innanzitutto voglio ringraziare i miei genitori, Daniela e Paolo. Grazie per essermi stati vicini in questi lunghi anni di studio e per avermi lasciato intraprendere la mia strada, anche nelle occasioni in cui questa non ha seguito i vostri pensieri e desideri. Grazie per i vostri consigli, per le vostre critiche, per i vostri rimproveri, per aver condiviso con me sia i momenti di gioia che quelli di difficoltà e per avermi insegnato a rialzarmi quando le cose non vanno come vorrei. Questo primo, grande, traguardo è dedicato a voi.

Grazie ad Elena, la mia fidanzata, per avermi sopportato tutti questi anni e per aver condiviso con me gioie, dolori, stress e felicità. Grazie per avermi motivato ed incitato tanto nei momenti felici quanto in quelli più difficili. Grazie per aver sempre creduto in me e nelle mie capacità e per avermi sempre dato il tuo totale ed incondizionato supporto. Se oggi sono come sono, è anche merito tuo.

Grazie a Roberto, il mio tutor aziendale, per avermi permesso di partecipare a Monokee, un prodotto che ora sento un po' anche mio. Grazie ad Enrico, Maria Giovanna, Sara, Silvia e Valentina, per avermi accolto nel loro gruppo e per avermi fatto lavorare in un ambiente fantastico. Mi avete insegnato che c'è solo un modo per raggiungere grandi traguardi: è farlo insieme. Senza di voi sarebbe stato sicuramente tutto più difficile. Lavorare nel vostro team è stato un onore, prima che un piacere. Grazie a Fabio, Federica, Silvia, Simone e a tutto lo staff di Athesys S.r.l. per avermi permesso di vivere questa bellissima esperienza e per avermi coinvolto dal primo fino all'ultimo giorno, e oltre.

Grazie a tutti i miei amici e a tutte le persone che mi sono state vicine in questi anni.

Grazie, infine, al Prof. Mauro Conti, relatore della mia tesi, per le sue indicazioni e l'aiuto che mi ha fornito durante lo stage e la stesura del documento.

Padova, Settembre 2016

Matteo Di Pirro

Indice

1 Introduzione	2
1.1 Organizzazione dell'elaborato	2
1.2 Convenzioni adottate	2
2 Contesto aziendale	4
2.1 Profilo dell'azienda	4
2.1.1 I servizi	4
2.2 Metodo di lavoro	5
2.2.1 Scrum	5
2.2.2 Ciclo Scrum	6
3 L'ambito: l'identità	8
3.1 Identity and Access Management - IAM	8
3.1.1 Identity Management - IdM	8
3.1.2 Access Management - AM	9
3.1.3 IdM e AM: IAM	11
3.2 Single Sign-On - SSO	12
3.2.1 Caratteristiche	12
3.2.2 Federazione delle identità	14
3.2.2.1 Security Assertion Markup Language	16
3.3 On-premises o IDaaS?	17
3.3.1 Caratteristiche intrinseche	19
3.3.2 Maturità funzionale	20
3.3.3 Cosa scegliere?	21
3.4 Un esempio: Sistema Pubblico di Identità Digitale	22
3.4.1 Attori e ruoli	22
3.4.2 Come funziona	23
3.4.3 Livelli di sicurezza	23
3.4.4 Come ottenere l'identità digitale	24
3.4.5 Vantaggi	24
4 Il progetto	26
4.1 Monokee	26
4.1.1 I moduli	28
4.2 Monokee Catalogue Manager	28
4.2.1 Funzionalità principali	28
4.3 Il futuro ed i competitors	34
5 Tecnologie, strumenti e linguaggi utilizzati	37

5.1	Tecnologie	37
5.1.1	Node.js	38
5.1.2	Express.js	40
5.1.3	MongoDB	41
5.2	Strumenti	42
5.2.1	Git	42
5.2.2	Bitbucket	43
5.2.3	JSHint	43
5.2.4	Mocha.js	43
5.2.5	DHC	44
5.2.6	Robomongo	44
5.2.7	APIDoc e JSDoc	45
5.3	Linguaggi	45
5.3.1	JavaScript	45
6	Analisi dei requisiti	47
6.1	Attori coinvolti	47
6.2	Casi d'uso ad alto livello	47
6.2.1	Operazioni permesse ad un Utente Non Autenticato	48
6.2.1.1	UCU1 - Aggiunta di Catalogue Manager ad un Application Broker	49
6.2.1.2	UCU2 - Modifica della configurazione specifica di un dominio di Catalogue Manager	50
6.2.1.3	UCU3 - Accesso a Catalogue Manager	51
6.2.1.4	UCU4 - Visualizzazione messaggio di errore per utente non autorizzato	51
6.2.2	Operazioni generali per l'Utente Autenticato	53
6.2.2.1	UCA1 - Visualizzazione Applicazioni	54
6.2.2.2	UCA2 - Aggiunta Applicazione Pubblica	55
6.2.2.3	UCA3 - Modifica Applicazione Pubblica	56
6.2.2.4	UCA4 - Rimozione Applicazione Pubblica	57
6.2.2.5	UCA5 - Gestione Cataloghi di Dominio	57
6.2.2.6	UCA6 - Ricerca Applicazione	58
6.2.2.7	UCA10 - Gestione Gruppi di Applicazioni	58
6.2.2.8	UCA13 - Visualizzazione di un messaggio di errore per applicazione pubblica già presente	59
6.2.2.9	UCA14 - Visualizzazione Statistiche	60
6.2.2.10	UCA15 - Visualizzazione Log	61
6.2.2.11	UCA16 - Visualizzazione Log sulle operazioni che hanno avuto successo	61
6.2.2.12	UCA17 - Visualizzazione Log delle operazioni che hanno generato errori	62
6.2.2.13	UCA18 - Ricerca Domini Aziendali di Monokee	62
6.2.2.14	UCA19 - Visualizzazione di un messaggio di errore per catalogo già presente	63
6.3	Requisiti	63
6.3.1	Principali requisiti funzionali	64
6.3.2	Requisiti di vincolo	68
7	Progettazione	69

7.1	Interfaccia REST-like	69
7.2	Architettura	70
7.2.1	Moduli esterni	70
7.2.2	Models di Monokee	72
7.2.3	Modules di Catalogue Manager	73
7.2.4	Routes di Catalogue Manager	74
7.3	Modalità di autenticazione	75
7.3.1	JSON Web Token	75
7.3.2	SSO con SAML	79
7.4	Principali componenti	81
7.4.1	Modelli	81
7.4.2	Moduli	88
7.5	Design Pattern	94
7.5.1	Module Pattern	94
7.5.2	Template Method	96
7.5.3	Middleware	97
7.5.4	Chain of Responsibility	98
8	Implementazione	100
8.1	Autenticazione a Catalogue Manager	100
8.1.1	Implementazione dei JWT	100
8.1.2	Modalità SAML IdP Initiated	101
8.2	Elenco dei servizi implementati	103
9	Verifica e validazione	107
9.1	Verifica	107
9.1.1	Analisi statica	107
9.1.2	Analisi dinamica	108
9.2	Validazione	109
10	Valutazione retrospettiva	110
10.1	Obiettivi fissati	110
10.2	Obiettivi raggiunti	110
10.3	Conclusioni	111
	Glossario	114
	Acronimi	124
	Bibliografia	127

Elenco delle figure

2.1	Logo di iVoxIT	4
2.2	Ruoli del framework Scrum	6
2.3	Framework <i>Scrum</i>	7
3.1	Ciclo di vita dell'identità	9
3.2	Gestione di identità, privilegi e accessi	11
3.3	Architettura di autenticazione tradizionale	14
3.4	Architettura di autenticazione federata	15
3.5	Principali componenti di SAML v2.0	16
3.6	Confronto tra sistemi IDaaS e on-premises nell'ambito IAM	18
3.7	Scelta tra sistemi IDaaS e on-premises nell'ambito IAM	22
3.8	Logo di SPID	22
3.9	Livelli di sicurezza SPID	23
4.1	Domain broker di Monokee	26
4.2	Application broker di Monokee	27
4.3	Catalogo applicativo di Monokee	27
4.4	Confronto tra Amazon Italia e Spagna	30
4.5	Gartner Magic Quadrants	36
5.1	Funzionamento dello stack MEAN	37
5.2	Confronto tra server Node.js e server Java	38
5.3	Funzionamento di un server Node.js	39
5.4	Confronto di prestazioni tra Java e Node.js	40
5.5	Logo di Express.js	40
5.6	Logo di MongoDB	41
5.7	Logo di Git	43
5.8	Logo di Bitbucket	43
5.9	Logo di Mocha.js	44
5.10	Logo di DHC	44
5.11	Logo di Robomongo	44
5.12	Logo di JavaScript	45
6.1	Operazioni Generali per l'Utente Non Autenticato	48
6.2	Operazioni Generali per l'Utente Autenticato	53
7.1	Architettura generale	70
7.2	Package models	72
7.3	Package modules	73
7.4	Package routes	74

7.5	Logo di JWT	75
7.6	Esempio di token di Catalogue Manager	76
7.7	Autenticazione con JWT	78
7.8	Confronto tra SSO SP e IdP Initiated	80
7.9	Modello Catalogue	82
7.10	Attributi di Category	82
7.11	Modello CatalogueDomain	83
7.12	Modello CatalogueForm	84
7.13	Modello CatalogueGroup	84
7.14	Modello CatalogueLog	85
7.15	Attributi di Log	85
7.16	Modello CatalogueSAML	86
7.17	Attributi di Instruction e Step	86
7.18	Modello CatalogueThirdType	87
7.19	Attributi di Property	87
7.20	Attributi di Header	88
7.21	Modello Domain	88
7.22	Gerarchia per la gestione delle immagini	89
7.23	Modulo per la gestione degli errori	90
7.24	Proprietà dei parametri per il salvataggio dei log	91
7.25	Modulo per il salvataggio dei log su database	92
7.26	Design pattern Template Method	96
7.27	Design pattern ChainOfResponsibility	98
8.1	Single Sign On IdP Initiated in Catalogue Manager	102

Elenco delle tabelle

5.1	Confronto di terminologia tra un database relazionale e MongoDB . .	42
6.21	Principali requisiti funzionali	68
6.22	Requisiti di vincolo	68
7.1	Corrispondenza tra CRUD e HTTP	69
7.2	Esempio di salvataggio di un log	93
8.1	Servizi REST esposti dal back end di Catalogue Manager	106

Elenco dei frammenti di codice

7.1	Esempio di header JWT	76
7.2	Esempio di payload JWT	77
7.3	Esempio di signature JWT	77
7.4	Esempio di header HTTP per l'invio di un JWT	78
7.5	Struttura del JSON di errore	90
7.6	Esempio di oggetto contenente i messaggi di errore	90
7.7	Esempio di oggetto contenente i codici dei log	92
7.8	Document di CatalogueLog	93
7.9	Log riformattato	93
7.10	Oggetto JavaScript in notazione classica	94
7.11	Module pattern	95
8.1	Header di un JWT di Catalogue Manager	100
8.2	Payload di un JWT di Catalogue Manager	101

Capitolo 1

Introduzione

In questo primo capitolo verrà esposta l'organizzazione dell'elaborato e verranno spiegate le convenzioni tipografiche adottate.

1.1 Organizzazione dell'elaborato

Il secondo capitolo esporrà il contesto aziendale nel quale si è svolta l'attività di stage.

Il terzo capitolo illustrerà la realtà attuale dei sistemi di *Identity and Access Management (IAM)_G*, le possibili evoluzioni future e gli ambiti di utilizzo.

Il quarto capitolo esporrà il prodotto per il quale si è svolta l'attività di stage, Monokee. Verrà inoltre esposto il ruolo del progetto svolto.

Il quinto capitolo esporrà le tecnologie, gli strumenti e i linguaggi utilizzati nel corso dell'attività di stage.

Il sesto capitolo presenterà le attività di analisi dei requisiti e l'individuazione delle principali funzionalità da implementare.

Il settimo capitolo presenterà ad alto livello la progettazione effettuata e le principali scelte progettuali implementate.

L'ottavo capitolo illustrerà i risultati dell'attività di implementazione di quanto progettato.

Il nono capitolo esporrà le attività di verifica e validazione.

Il decimo capitolo riassumerà alcune considerazioni finali, relativamente al prodotto implementato, all'attività di stage nel suo complesso ed all'intero percorso di studi dello studente.

1.2 Convenzioni adottate

Nella stesura del presente documento sono state adottate le seguenti convenzioni tipografiche:

- ogni occorrenza di termini tecnici, ambigui o di acronimi verrà marcata in corsivo e con una G a pedice. Questo indica che quel termine è presente nel glossario in fondo al documento;
- la prima occorrenza di un acronimo viene riportata con la dicitura estesa, in corsivo, e le lettere che compongono l'acronimo vengono riportate in grassetto. Ogni occorrenza successiva presenterà solo la forma ridotta;
- le parole chiave presenti in ciascun paragrafo saranno marcate in grassetto, in modo da consentirne una facile individuazione;
- ogni termine corrispondente a nomi di file, componenti dell'architettura o codice verrà marcato utilizzando un *font* non proporzionale (a larghezza fissa);
- i termini in lingua inglese non presenti nel glossario, e non marcati in grassetto, sono evidenziati semplicemente in corsivo, senza la G a pedice.

Inoltre, per ciascun diagramma *Unified Modeling Language (UML)_G* è utilizzato lo standard 2.0.

Capitolo 2

Contesto aziendale

2.1 Profilo dell'azienda

L'attività di stage descritta nel presente documento è stata svolta presso l'azienda **iVoxIT S.r.l** (logo in Figura 2.1) di Padova.



Figura 2.1: Logo di iVoxIT

iVoxIT S.r.l. fa parte del gruppo **Athesys S.r.l.**, fondato nel 2010 dall'unione di professionisti dell'**Information Technology (IT)_G** con l'obiettivo di fornire consulenza ad alto livello tecnologico e progettuale. Tra le altre cose, *Athesys S.r.l* fornisce supporto nell'istanziazione del processo di **Data Lifetime Management (DLM)_G**, con particolare attenzione alla sicurezza nella conservazione e nell'esposizione dei dati gestiti.

L'azienda opera in tutto il territorio nazionale, prevalentemente nel Nord Italia, e vanta esperienze a livello europeo in paesi quali Olanda, Regno Unito e Svizzera.

Grazie all'adozione delle *best practices_G* definite dalle linee guida **Information Technology Infrastructure Library (ITIL)_G** e alla certificazione **ISO 9001** il gruppo è in grado di assicurare un'alta qualità professionale.

2.1.1 I servizi

I principali ambiti di consulenza del personale di *Athesys S.r.l* sono i seguenti:

- **Database Management:** nelle aziende è sempre più necessario memorizzare grosse quantità di dati in modo da renderli facilmente reperibili. I **DataBase Administrator (DBA)_G** di *Athesys S.r.l* guidano il cliente nella scelta della

tecnologia più appropriata, occupandosi anche dell'integrazione con il sistema informativo preesistente. In particolare:

- identificazione e progettazione dell'infrastruttura più adatta in base alle esigenze di affidabilità e scalabilità;
 - progettazione, realizzazione e manutenzione di basi di dati efficienti;
 - interazione e migrazione fra diverse piattaforme;
 - protezione dei dati nei confronti di accessi o manipolazioni non autorizzate;
 - *backup* e ripristino di dati.
- **Identity and Access Management:** come sarà maggiormente descritto in seguito, gli strumenti di IAM_G consentono alle aziende di effettuare il controllo dell' $identità_G$ e degli accessi ai propri servizi attraverso sistemi di *Single Sign On (SSO)*_G. Fra i servizi offerti ci sono:
 - studio del sistema informativo aziendale;
 - individuazione del sistema IAM_G più adatto;
 - implementazione di motori di $workflow_G$ coinvolti nei processi di *provisioning_G*;
 - implementazione di moduli di *autenticazione_G* e *autorizzazione_G*.
 - **System Integration:** integrazione dei sistemi informativi e delle infrastrutture aziendali in modo da garantire $efficienza_G$, $scalabilità_G$ e $affidabilità_G$.
 - **Business Intelligence:** un'azienda deve essere in grado di reperire, elaborare e analizzare i dati. Questi dati, però, provengono da sorgenti sempre più eterogenee. *Athesys S.r.l* si occupa di raccogliere questi dati e di renderli disponibili sotto forma di *report*.

2.2 Metodo di lavoro

Athesys S.r.l, e di conseguenza *iVoxIT S.r.l*, utilizza un ciclo di sviluppo $agile_G$ nel quale il *team* è gestito secondo una metodologia *Scrum*. *Scrum* è una metodologia di sviluppo semplice da comprendere, ma difficile da padroneggiare: non è un processo o una tecnica per costruire i prodotti, al contrario è un *framework_G* del quale si possono impiegare i vari processi e tecniche. In *Scrum* sono presenti uno o più *team*; i cui membri hanno ruoli, regole e capacità diverse, e tutti competono al successo finale.

2.2.1 Scrum

Gli aspetti fondamentali di Scrum sono:

- **Trasparenza:** gli aspetti significativi del processo devono essere visibili ai responsabili;
- **Ispezione:** gli utenti devono ispezionare ciò che è stato fatto, in modo da controllare se le cose stanno andando nel verso giusto. Le ispezioni non dovrebbero essere tanto frequenti da bloccare il flusso di lavoro, e sono utili quando condotte da ispettori qualificati.

- **Adattamento:** se un'ispezione determina che uno o più aspetti deviano dai limiti accettabili, e che, di conseguenza, il prodotto finale non sarà accettabile, il processo o il materiale ispezionato deve essere corretto. La correzione deve essere fatta il prima possibile, per evitare ulteriori deviazioni.

In *Scrum* esistono solamente tre ruoli principali (mostrati in Figura 2.2):

- **Product Owner**, mantiene la visione complessiva del prodotto;
- **Scrum Master**, aiuta il *team* a rimanere concentrato e focalizzato sull'obiettivo finale;
- **Development Team**, realizza il prodotto.

I membri del *team* si auto organizzano il lavoro al posto di ricevere indicazioni esterne, e dispongono di competenze che non necessitano dipendenze da altri membri dello stesso *team*. In questo modo si ottimizzano flessibilità, creatività e produttività.

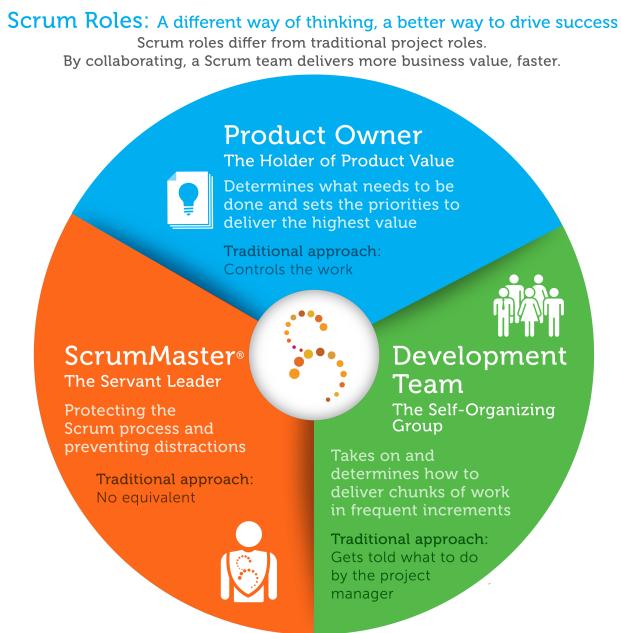


Figura 2.2: Ruoli del framework Scrum¹

2.2.2 Ciclo Scrum

Scrum è un metodo iterativo che divide il progetto in blocchi rapidi di lavoro chiamati **Sprint**, della durata massima di quattro settimane. Alla fine di ogni *Sprint* si ottiene un incremento del prodotto.

In Figura 2.3 è mostrato un tipico ciclo *Scrum*. Il *Product Owner* crea una "lista di desideri" con delle priorità associate, chiamata **Product Backlog**. Questa lista contiene tutto ciò di cui potrebbe aver bisogno il prodotto, ed è l'unica fonte di requisiti.

¹Fonte: <https://www.scrumalliance.org/employer-resources/scrum-roles-demystified>

L'unico responsabile della *Product Backlog* è il *Product Owner*, che ne mantiene il contenuto e l'ordinamento.

Ad ogni *Sprint* il *Development Team* prende in considerazione un pezzo della lista e decide come implementare le funzionalità richieste (**Sprint Planning**). La durata massima dello *Sprint Planning* è otto ore al giorno per un mese. È compito dello *Scrum Master* assicurarsi del rispetto dei tempi: quotidianamente effettua una breve riunione, della durata di circa quindici minuti, per valutare i progressi fatti (**Daily Scrum**). Queste consentono al *team* di pianificare le attività per le successive 24 ore, valutando il lavoro svolto dall'ultimo *Daily Scrum*. Lo *Scrum Master* si assicura che il *Daily Scrum* venga svolto, ma di fatto questo viene condotto dal *Development Team*. Così facendo si promuove la comunicazione.



Figura 2.3: Framework Scrum²

Lo *Scrum* termina con una **Sprint Review** e con una **Sprint Retrospective**. La prima ispeziona i risultati ottenuti e li corregge se necessario. È una riunione informale, nella quale lo *Scrum Master* collabora con gli *stakeholders*_G per valutare ciò che è stato fatto. Il *Product Owner* espone cosa è stato fatto e cosa no, analizza le priorità della *Product Backlog* e decide, insieme agli altri, cosa sarà fatto nel prossimo *Sprint*. La *Sprint Retrospective*, invece, analizza i miglioramenti che possono essere fatti al *team*, e si svolge dopo la *Sprint Review*. Vengono identificati gli effetti dello *Sprint* appena concluso sulle persone e sugli strumenti, e, se necessario, vengono pianificati dei miglioramenti. È compito dello *Scrum Master* incoraggiare il *team* a migliorarsi.

Alla fine di ogni *Sprint* il prodotto è potenzialmente consegnabile al cliente.

²Fonente: <https://www.scrumalliance.org/why-scrum>

Capitolo 3

L'ambito: l'identità

3.1 Identity and Access Management - IAM

La capacità di fornire il giusto accesso ai giusti utenti alle giuste risorse nei tempi e nelle modalità giuste è la chiave per aumentare la produttività e per mitigare rischi di qualsiasi tipo. Per questo la gestione dell' $identità_G$ è una parte fondamentale nella sicurezza del mondo IT_G .

Storicamente l'approccio a questa disciplina è stato molto semplice: tutto si basava su **qualcosa che l'utente conosceva**, tipicamente una *password*. Per applicazioni a rischio più alto sono stati poi introdotti ulteriori controlli, come l'*autenticazione_G* a più fattori (**qualcosa che l'utente possiede**, come *smartphone*, carte speciali, chiavette, eccetera) o controlli biometrici (**qualcosa che l'utente è**, come scansione della retina o rilevamento delle impronte digitali).

Questi sistemi sono ottimi se si deve gestire solamente l' $identità_G$ di una persona, ma l'avvento del *mobile* ha cambiato tutto. Inoltre, in seguito alla nascita dell'*Internet of Things (IoT)_G*, anche qualsiasi dispositivo può connettersi ad Internet: non più solo utenti o *smartphone*, ma anche sensori, elettrodomestici, e altro.

In un contesto come questo è necessario essere in grado di gestire agevolmente le $identità_G$ e gli accessi alle risorse. Ecco perché nascono sistemi di *Identity and Access Management (IAM)_G*. Nelle successive sezioni saranno approfondite le due componenti fondamentali di un sistema di IAM_G : la gestione dell' $identità_G$ (**Identity Management (IdM)_G**) e la gestione degli accessi (**Access Management (AM)_G**).

3.1.1 Identity Management - IdM

La funzionalità principale dell'*Identity Management_G* è fornire $identità_G$ creando *account* ed eventualmente disabilitando o cancellando quelli creati quando non sono più necessari (Figura 3.1). Tipicamente questo è un compito complicato, perché le $identità_G$ richieste sono molto eterogenee: impiegati, clienti, fornitori, *partner*, eccetera. È pertanto necessario un fornitore di $identità_G$ unico per ogni tipologia di *account* che automatizzi l'intero processo. Ogni tipologia deve, inoltre, avere degli attributi ben definiti (ad esempio il periodo di validità).

Solitamente ad ogni *account* è associato un identificatore univoco (ID), in modo da associare un' $identità_G$ all'*account* stesso e tracciare le operazioni effettuate a scopo di *log* o *report*. L'ID può essere rappresentato da un indirizzo email, oppure essere generato *ad hoc*, ma è fondamentale che la sua unicità sia mantenuta: questo comporta la

memorizzazione dell'ID stesso in caso di cancellazione, in modo da evitare la possibilità di inconsistenze.

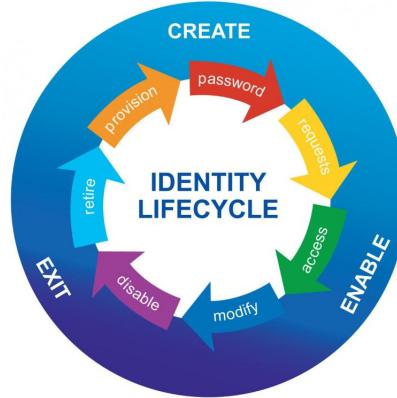


Figura 3.1: Ciclo di vita dell'identità¹

Scegliere tra disabilitazione e cancellazione di un *account* è fondamentale: la cancellazione è, ovviamente, più rischiosa, soprattutto se fatta in modo errato, ma consente il riutilizzo dell'*identità_G* cancellata. La disabilitazione, invece, consente il *rollback_G*, ma comporta un maggior numero di dati memorizzati. Generalmente gli *account* vengono disabilitati quando l'utente associato non è presente (ad esempio per periodi di malattia) e riabilitati al rientro, oppure cancellati dopo un certo numero di giorni di inattività (**use it or lose it**).

Le politiche di creazione, disabilitazione e cancellazione, comunque, variano da un'organizzazione all'altra e, in generale, si integrano con i sistemi di gestione del personale.

3.1.2 Access Management - AM

La semplice associazione *identità_G*/*account* è poco utile se non ci sono autorizzazioni o politiche di accesso associate a quell'*account*. L'*am_G* si occupa di creare e far rispettare queste politiche: applica regole diverse in base al ruolo e controlla le autorizzazioni degli *account* generati e gestiti dai sistemi di *IdM_G*.

Le politiche di *autenticazione_G* e *autorizzazione_G* possono essere rafforzate da un sistema di controllo degli accessi, usato per imporre dei permessi sui sistemi e sulle risorse. Principalmente ne esistono di quattro tipi:

- **Mandatory Access Control (MAC)_G**;
- **Discretionary Access Control (DAC)_G**;
- **Role-Based Access Control (RBAC)_G**;
- **Attribute Based Access Control (ABAC)_G**.

Di seguito verranno descritti vantaggi e svantaggi di ciascuno.

¹Fonte: <https://www.linkedin.com/pulse/identity-management-internet-things-george-moraetes>

Mandatory Access Control

MAC_G è fondato su un modello gerarchico basato sui livelli di sicurezza. Ogni utente, o oggetto, è associato ad un livello di sicurezza. Gli utenti possono accedere alle risorse che corrispondono al loro livello, o che hanno un livello inferiore nella gerarchia.

Gli accessi sono controllati solamente dagli amministratori, che impostano ogni permesso degli utenti. I sistemi MAC_G sono considerati molto sicuri proprio grazie a questa amministrazione centralizzata e sono generalmente utilizzati in ambiti governativi.

Il principale pregio è però anche un difetto, perché gli amministratori devono obbligatoriamente assegnare ogni permesso senza poter delegare niente: per sistemi di grandi dimensioni questo può portare ad un "sovraffuso" dello staff amministrativo. Proprio per questo motivo i sistemi MAC_G sono poco usati in ambito web.

Discretionary Access Control

DAC_G si basa su un modello governato da una lista che indica chi può accedere a cosa e cosa può fare con quella risorsa. Nella lista vanno elencati tutti gli utenti e i permessi, per ogni risorsa. Sistemi di questo tipo utilizzano un'architettura più distribuita e sono generalmente più facili da gestire dei MAC_G , perché gli amministratori non hanno l'onere dell'impostazione dei permessi. Tuttavia sono anche meno sicuri, perché i permessi possono essere impostati in modo errato.

La maggior parte dei sistemi operativi usano modelli DAC_G .

Role-Based Access Control

$RBAC_G$ si basa su un modello di ruoli e responsabilità. Agli utenti non è permesso l'accesso al sistema: un insieme di ruoli è assegnato ad un utente, e un insieme di livelli di accesso è assegnato a ciascun ruolo. I ruoli sono gestiti dagli amministratori, che determinano quanti e quali devono essere e, successivamente, assegnano questi ruoli a funzioni e compiti.

In questo modello è necessario conoscere in modo più specifico il sistema, perché ad ogni ruolo devono essere assegnate delle politiche di accesso e delle funzioni. È possibile delegare il ruolo di amministratore semplicemente creando un nuovo ruolo e assegnandolo a delle persone.

$RBAC_G$ è difficile da implementare; per grandi organizzazioni, inoltre, è complicato identificare con precisione i ruoli. Nonostante questo, però, è molto utilizzato nell'ambito web, perché consente di automatizzare la creazione e la gestione degli utenti.

Attribute Based Access Control

$ABAC_G$ è un meccanismo di controllo degli accessi alle risorse che fornisce maggiore flessibilità rispetto a $RBAC_G$. L'autorizzazione_G è basata su una serie di attributi associati con utenti, risorse, transazioni e attività; le decisioni sono rappresentate da regole condizionali sui valori degli attributi. In questo modo le politiche di sicurezza sono completamente indipendenti da utenti e oggetti.

Gli attributi utilizzabili in un sistema di questo tipo sono moltissimi, e variano da quelli più "stabili" (come ruoli, etichette di sicurezza, eccetera) a quelli più "dinamici" (come posizione geografica, tipologia di autenticazione_G effettuata, ambiente operativo dal quale viene effettuato l'accesso, eccetera). Considerato che le decisioni vengono prese sulla base delle condizioni nel momento dell'accesso, $ABAC_G$ è un meccanismo di controllo degli accessi **dipendente dal contesto**.

3.1.3 IdM e AM: IAM

IdM_G e AM_G sono componenti fondamentali per qualsiasi sistema di sicurezza e dipendono fortemente l'uno dall'altro: non si possono associare ruoli e politiche se non si dispone di *account*, e un *account* senza permessi e politiche di accesso non serve a niente. La loro unione e combinazione porta alla definizione di sistemi di IAM_G . La gestione di accessi e $identità_G$ non è più un male non necessario, ma è una componente fondamentale della sicurezza di qualsiasi organizzazione: non si tratta più di avere a che fare solamente con persone confinate nel loro ufficio, ma di gestire interazioni tra impiegati, fornitori, soci, clienti e oggetti (Figura 3.2).



Figura 3.2: Gestione di identità, privilegi e accessi²

L' IoT_G , in particolare, è fonte di un elevatissimo numero di opportunità per le aziende e di servizi per i consumatori: un esempio è la possibilità di connettere le auto ad Internet, per aprirle, chiuderle, monitorare i consumi, accendere o spegnere il motore, eccetera. D'altro canto ha portato ad un grande incremento dei rischi legati alla sicurezza: ogni dispositivo connesso ad Internet rischia di essere compromesso e di fornire pericolose informazioni circa il suo utilizzo. Il dispositivo può poi essere controllato in modo da agire sull'ambiente circostante: sarebbe possibile aumentare o diminuire la temperatura di una stanza, spegnere, disabilitare o manomettere delle funzionalità e molto altro. È necessario gestire anche l' $identità_G$ delle cose ($IDentity of Things (IDoT)_G$) per assegnare ruoli e permessi anche agli oggetti, oltre che alle persone.

La marcata eterogeneità delle entità coinvolte rende i sistemi di IAM_G molto complessi: è necessario gestire e proteggere enormi moli di dati (**sicurezza**), derivanti da interazioni tra persone e oggetti (**privacy**) effettuate con modalità in continua espansione (**resilienza_G** e **scalabilità_G**). Oltre a questo, le $identità_G$ da gestire sono sempre di più: non ci sono più solo quelle fornite dai governi, ma anche quelle digitali create direttamente dagli utenti (ad esempio quelle sui *social networks*).

Tra i compiti principali dell' IAM_G , dunque, ci sono:

²Immagine tratta da [11]

- mantenimento della consistenza dell' $identità_G$ di un utente tra le varie applicazioni;
- supporto del SSO_G ;
- controllo degli accesi alle risorse basato sui ruoli o sugli attributi ($RBAC_G$ e $ABAC_G$);
- gestione di diverse $identità_G$, ruoli e permessi: la stessa persona può essere un amministratore di un sistema e un semplice utente in un altro;
- fornitura di processi e procedure per gestire il ciclo di vita dell' $identità_G$ (di utenti e di oggetti);
- gestione dell' IoT_G ;
- fornitura di meccanismi di accesso sicuri e senza interruzioni ad applicazioni interne, esterne e $cloud_G$;
- identificazione di *account* orfani (ancora attivi sebbene non più associati ad un utente);
- incremento della produttività degli utenti, grazie alla possibilità di accedere alle applicazioni in meno tempo e in maggiore sicurezza;
- incremento della sicurezza;
- gestione delle relazioni tra oggetti e persone;
- memorizzazione delle operazioni svolte da ciascun utente (o oggetto) nel sistema e generazione di *log* e *report*;
- fornitura di servizi *self service* per l'utente: cambio *password*, gestione profilo, eccetera;
- pianificazione e giustificazione dell'allocazione delle risorse.

3.2 Single Sign-On - SSO

L'avvento dell'era del $cloud_G$ ha portato alla proliferazione di applicazioni web. Se questo da un lato rappresenta un grande passo in avanti, dall'altro porta molti problemi: le applicazioni $on-premises_G$ possono essere configurate per supportare un unico insieme di credenziali, ma molto spesso le applicazioni web salvano le informazioni degli utenti nei loro *databases* senza condividerle con gli altri. Esiste quindi un notevole numero di credenziali diverse, in applicazioni diverse, per lo stesso utente.

Per risolvere questo problema serve un sistema in grado di fornire accesso a tutte le applicazioni con un unico *login* e in grado di scalare per supportare un insieme sempre maggiore di utenti e credenziali.

La soluzione si chiama SSO_G . Il SSO_G è un servizio normalmente fornito dai sistemi di AM_G : all'utente vengono chieste le credenziali solamente una volta, e, dopo il primo accesso, il sistema memorizza la sessione e trasmette i suoi dati (e la sua $identità_G$) a tutte le applicazioni.

3.2.1 Caratteristiche

I benefici principali sono i seguenti:

- **esperienza utente:** l'utente non viene continuamente interrotto dalle richieste delle sue credenziali. Di fatto il SSO_G rimuove i confini tra le applicazioni e consente all'utente di spostarsi dall'una all'altra senza interruzioni;
- **sicurezza:** le credenziali sono controllate e gestite da un server centrale e non dal servizio al quale l'utente sta accedendo, che di conseguenza non può memorizzarle. Questo limita le possibilità di *phishing_G*;
- **risparmio di risorse:** gli amministratori possono risparmiare tempo e risorse utilizzando un punto di accesso centralizzato.

D'altro canto, il SSO_G presenta anche degli aspetti negativi:

- **Single point of failure:** se il *provider SSO_G* non funziona gli utenti non possono autenticarsi a niente;
- **one key to kingdom:** se il *provider SSO_G* viene violato il danno è enorme.

Il problema principale, però, è che un sistema di questo tipo necessita, prima di tutto, di ricevere le informazioni sull'*identità_G* dell'utente da qualche parte. Attualmente, tuttavia, non esiste un unico *database* di utenti.

Esistono pertanto tre principali approcci al SSO_G :

- **centralizzato:** il principio è di disporre di un *database* globale e centralizzato di tutti gli utenti e di centralizzare allo stesso modo la politica di sicurezza. Questo approccio è destinato principalmente ai servizi dipendenti tutti dalla stessa entità, per esempio all'interno di una azienda, in quanto è molto difficile, se non impossibile, disporre di un *database* con le *identità_G* di tutti gli utenti nel mondo. Inoltre, la centralizzazione della politica di sicurezza è una limitazione troppo forte per un uso non specificatamente aziendale;
- **federato:** differenti gestori ("federati" tra loro) gestiscono dati di uno stesso utente. L'accesso ad uno dei sistemi federati permette automaticamente l'accesso a tutti gli altri sistemi. Un viaggiatore, ad esempio, potrebbe essere sia passeggero di un aereo che ospite di un albergo. Se la compagnia aerea e l'albergo usassero un approccio federato, potrebbero stipulare un accordo reciproco sull'*autenticazione_G* dell'utente finale. Il viaggiatore potrebbe quindi autenticarsi per prenotare il volo ed essere autorizzato, in forza di quella sola *autenticazione_G*, ad effettuare la prenotazione della camera d'albergo. Questo approccio è stato sviluppato per rispondere ad un bisogno di gestione decentralizzata degli utenti: ogni gestore federato mantiene il controllo della propria politica di sicurezza.
- **cooperativo:** il principio è che ciascun utente dipenda, per ciascun servizio, da uno solo dei gestori cooperanti. In questo modo se si cerca, ad esempio, di accedere ad una rete locale, l'*autenticazione_G* viene effettuata dal gestore che ha in carico l'utente per l'accesso alla rete. Come per l'approccio federato, in questa maniera ciascun gestore gestisce in modo indipendente la propria politica di sicurezza. L'approccio cooperativo risponde ai bisogni di strutture istituzionali nelle quali gli utenti sono dipendenti da una entità, come ad esempio in università, laboratori di ricerca, amministrazioni, eccetera.

Di seguito verrà descritto in dettaglio l'ambito della federazione delle *identità_G*, con un esempio di protocollo utilizzato per lo scambio dei dati: *Security Assertion Markup Langage (SAML)_G*.

3.2.2 Federazione delle identità

In un modello di federazione delle $identità_G$ ci sono due componenti principali, ***Identity Provider (IdP)*** e ***Service Provider (SP)***, che condividono un rapporto di fiducia: l' IdP_G deve fidarsi dell'applicazione per l'invio delle informazioni dell'utente e l'applicazione deve fidarsi dell' IdP_G per accettare l' $identità_G$ ricevuta.

L' IdP_G memorizza le informazioni sull' $identità_G$ degli utenti. Il suo compito è molto più importante della semplice *autenticazione_G* dell'utente, perché le informazioni sulla sua $identità_G$ saranno poi inviate alle applicazioni della federazione. Le principali modalità di memorizzazione delle credenziali sono *database* e *servizi di directory_G*, come *Active Directory (AD)_G*.

Il SP_G , invece, fornisce servizi agli utenti.

Una delle principali confusioni sulla federazione dell' $identità_G$ è la differenza tra ***identità_G federata*** e ***autenticazione_G federata***. La seconda può essere considerata come un sottoinsieme della prima: l'*autenticazione_G* è semplicemente un modo per verificare che chi sta tentando l'accesso sia chi dice di essere. L'*autenticazione_G* federata è quindi un mezzo per ottenere l' $identità_G$ federata: si può avere la prima senza la seconda. La federazione dell' $identità_G$ è invece un modo sicuro per trasmettere l' $identità_G$ attraverso sistemi diversi. La chiave dell'intero sistema è la fiducia: chi memorizza le informazioni e chi le chiede devono fidarsi l'un l'altro. L'applicazione non fa nessuno sforzo per verificare l' $identità_G$ dell'utente: semplicemente si fida dell' IdP_G .

Uno dei punti chiave della federazione delle $identità_G$ è che l'*autenticazione_G* è divisa dall'*autorizzazione_G*. La maggior parte delle applicazioni, infatti, inizialmente autentica l'utente e successivamente decide cosa l'utente può fare in base all'esito dell'*autenticazione_G* (Figura 3.3).

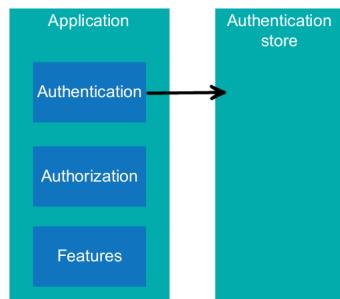


Figura 3.3: Architettura di autenticazione tradizionale³

Questo, comunque, non significa che l'applicazione esegue il processo di *autenticazione_G*: potrebbe infatti appoggiarsi ad un sistema esterno (ad esempio *AD_G*). Il punto chiave è che è l'applicazione stessa ad **inviare** la richiesta di *autenticazione_G*.

Con la federazione dell' $identità_G$ quest'ultima richiesta non deve essere inviata dall'applicazione. I due processi (*autenticazione_G* e *autorizzazione_G*) possono essere fatti separatamente e da due sistemi diversi. In questo caso, l' IdP_G invia le informazioni sull'utente all'applicazione, che le utilizza per decidere se autorizzare o meno determinate operazioni (Figura 3.4). È l' IdP_G a "parlare" con il sistema di memorizzazione delle credenziali.

³Immagine ispirata da [4]

⁴Immagine ispirata da [4]

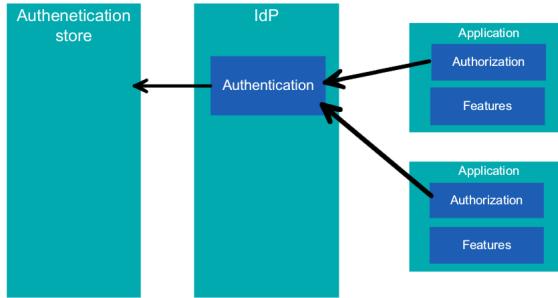


Figura 3.4: Architettura di autenticazione federata⁴

Vantaggi

La federazione dell'*identità_G* offre molti vantaggi, sia lato amministratore che lato utente. Uno dei punti forti è proprio questo: molti sistemi di *SSO_G* tradizionali portano benefici solo ad uno dei due lati.

I principali aspetti positivi sono:

- **sicurezza delle credenziali degli utenti:** molti utenti sono preoccupati dal fatto che le applicazioni possano memorizzare le loro credenziali. Se le prime vengono compromesse, le seconde sono accessibili da chiunque. Con la federazione dell'*identità_G* questo non succede, perché le credenziali sono da tutt'altra parte;
- **esperienza utente senza interruzioni:** così come detto per il *SSO_G*, l'utente può accedere a molte applicazioni senza quasi rendersi conto di quello che sta succedendo;
- **solo decisioni di autorizzazione_G:** le applicazioni devono decidere solo cosa può, o non può, fare un utente, senza preoccuparsi dell'*autenticazione_G*;
- **riduzione del numero di username e password:** essendo un sistema di *SSO_G*, è sufficiente una coppia di *username/password* per accedere ovunque. In aggiunta, il *SSO_G* federato è molto sicuro;
- **facilità di integrazione:** al posto di unire due *IdP_G* separati (processo che può essere molto lungo e complesso), è sufficiente istruire un *IdP_G* a fidarsi dell'altro e viceversa;
- **altamente estensibile:** può essere usato qualsiasi nuovo tipo di *autenticazione_G* senza modificare niente nelle applicazioni.

Svantaggi

I principali svantaggi, invece, sono:

- **one key to kingdom:** come già discusso per il *SSO_G*, se l'*IdP_G* viene violato i malintenzionati avranno accesso a tutte le applicazioni di tutti gli utenti;
- **infrastruttura specializzata:** implementare una soluzione di questo tipo è molto difficile e costoso: bisogna sviluppare un *IdP_G*, configuralo e manutenerlo;

- **conformità agli standard:** per poter comunicare con altri IdP_G serve "parlare la stessa lingua", che tradotto significa essere conformi allo stesso standard. Il problema è che lo standard non è unico, e riuscire a raggiungere un buon livello di interoperabilità è difficile.

3.2.2.1 Security Assertion Markup Language

$SAML_G$ è un standard basato su *eXtensible Markup Language (XML)*_G per inviare informazioni sull' $autenticazione_G$ e sull' $identità_G$, in particolare:

- gli attributi dell'utente;
- l' $autorizzazione_G$ che l'utente possiede sulla risorsa richiesta;
- l' $autenticazione_G$, avvenuta o meno, dell'utente.

L'uso di $SAML_G$ presenta due principali vantaggi:

1. **indipendenza dalla piattaforma:** permette ai sistemi di sicurezza ed alle applicazioni *software* di essere sviluppate in modo indipendente. È progettato per interagire con qualsiasi sistema;
2. **riduzione del rischio nel trasferimento delle informazioni:** tramite meccanismi di sicurezza come firme digitali (XML_G *Signature*), cifrature (XML_G *Encryption*), crittografia, codifica, eccetera, si riduce il rischio di attacchi esterni durante lo scambio di asserzioni.

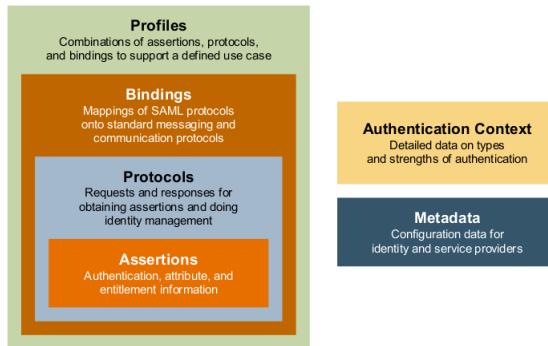


Figura 3.5: Principali componenti di SAML v2.0⁵

$SAML_G$ è una collezione di standard con quattro componenti principali: **Assertions**, **Protocols**, **Bindings** e **Profiles** (Figura 3.5). Di seguito verrà spiegato, brevemente, il loro scopo.

Assertions

$SAML_G$ permette di assicurare l' $identità_G$ di un utente nella forma di **dichiarazioni** su un **soggetto** (persona fisica o macchina). Un'asserzione è un insieme di queste informazioni, scritte in formato XML_G . Ad esempio, un'asserzione potrebbe dire che il

⁵ Immagine tratta da [15]

soggetto si chiama "Mario Rossi", possiede l'indirizzo email "mariorossi@email.com" ed è membro del gruppo "programmatori".

Solitamente un'asserzione contiene un insieme di informazioni richieste e informazioni opzionali comuni a tutte le asserzioni: è presente un soggetto, delle **condizioni** usate per validare l'asserzione stessa e altre dichiarazioni. Queste ultime possono essere di tre tipi:

- **di autenticazione_G**: create dall'entità che autentica il soggetto;
- **di attributo**: attributi sull'*identità_G* del soggetto (nome, indirizzo email, eccetera);
- **di autorizzazione_G**: ciò che l'utente può fare con la risorsa richiesta.

Protocols

Definiscono come si richiedono e si ricevono le asserzioni. I protocolli definiscono come devono essere formulate le richieste (**SAMLRequest**) e le risposte (**SAMLResponse**) e forniscono regole per interpretare questi elementi. Le richieste sono chiamate *queries*: il SP_G invia delle *queries* all' IdP_G . Le risposte arrivano sotto forma di asserzioni. Di conseguenza ci sono tre tipi di *query*, corrispondenti ai tre tipi di asserzione: **di autenticazione_G**, **di attributo** e **di autorizzazione_G**.

Bindings

Indicano esattamente come i messaggi $SAML_G$ debbano essere inviati attraverso i protocolli di trasmissione delle informazioni (ad esempio **HyperText Transfer Protocol (HTTP)_G**). Il *binding* è un'operazione di mappatura delle *SAMLRequests* e *SAMLResponses* sui protocolli di comunicazione standard.

Profiles

Definiscono alcuni scenari d'uso comune su come le asserzioni, i protocolli e i *binding* debbano essere combinati tra loro in procedure definite per soddisfare gli scopi per i quali tali scenari d'uso sono stati pensati.

3.3 On-premises o IDaaS?

Un numero sempre crescente di compagnie sta "migrando" da soluzioni IAM_G *on-premises_G* a soluzioni $cloud_G$ (**IDentity as a Service (IDaaS)_G**). Questa migrazione è in gran parte dovuta alla possibilità di utilizzare il SSO_G tra applicazioni $cloud_G$ (**Software as a Service (SaaS)_G**), ma le differenze tra le due modalità sono molte altre. Di seguito verrà presentato un confronto basato sui seguenti punti:

- **caratteristiche intrinseche**: in particolare:
 - **piattaforma software e facilità di installazione**: il costo iniziale per configurare ed installare il prodotto;
 - **manutenzione e aggiornamento**: risorse richieste per manutenere ed aggiornare il prodotto;
 - **sicurezza e protezione dei dati**: difficoltà per rendere il sistema sicuro;
 - **privacy**: difficoltà per mantenere i dati privati;
 - **agilità**: capacità di adattarsi ai cambiamenti;

- **distribuzione dell'architettura:** vantaggi sul posizionamento dell'architettura (*on-premises_G* o *cloud_G*).
- **maturità funzionale:** in particolare:
 - **modernità dell'architettura:** consistenza, accesso facile a tutte le funzionalità, eccetera;
 - **modernità dell'interfaccia grafica:** facilità con la quale l'utente trova quello che cerca;
 - **integrazione con *SaaS_G*:** facilità di integrazione del *SSO_G* con applicazioni di tipo *SaaS_G*;
 - **integrazione con applicazioni *on-premises_G*:** facilità di integrazione del *SSO_G* con applicazioni di tipo *on-premises_G*;
 - **personalizzazione:** facilità di modifica per adattarsi a necessità specifiche;
 - **aderenza agli standard:** grado di supporto degli standard;
 - **maturità generale delle funzionalità *IAM_G*:**
 - **flessibilità dei termini di licenza:** supporto per varie opzioni appropriate per differenti casi d'uso.

In Figura 3.6 viene mostrato un confronto qualitativo sulle caratteristiche appena elencate.

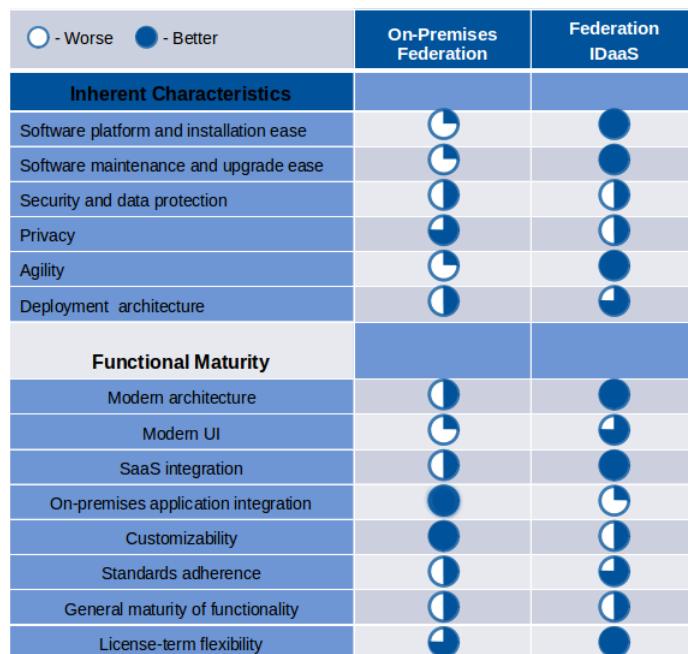


Figura 3.6: Confronto tra sistemi IDaaS e on-premises nell'ambito IAM⁶

⁶Immagine tratta da [2]

3.3.1 Caratteristiche intrinseche

Piattaforma software e facilità di installazione

Per definizione l' $IDaaS_G$ è quello in cui il servizio mette a disposizione la piattaforma e il *software*, quindi non richiede nessuno sforzo da parte degli utilizzatori, che risparmiano moltissimo tempo e denaro pagando solo le funzionalità di cui hanno bisogno. Inoltre:

- se l'organizzazione non dispone di uno *staff* adatto a mantenere un sistema *on-premises_G* o non può procurarselo la scelta $IDaaS_G$ è la migliore;
- se l'organizzazione non ha nessuna esperienza di *SaaS_G* allora conviene l'alternativa *on-premises_G*;
- se il tempo a disposizione è poco conviene $IDaaS_G$, in quanto non necessita della configurazione iniziale;
- se il costo iniziale per la creazione dell'infrastruttura è elevato conviene, ancora una volta, $IDaaS_G$.

Manutenzione e aggiornamento

In questo ambito $IDaaS_G$ vince di misura. Infatti, non essendoci la necessità di installazione, non c'è neanche quella di manutenzione o aggiornamento. Al contrario, un prodotto *on-premises_G* necessita di essere seguito e aggiornato.

Sicurezza e protezione dei dati

Nell'ambito della sicurezza la decisione è più difficile, e in generale dipende dalla situazione. Si potrebbe pensare che sia più sicuro mantenere i dati *on-premises_G* in quanto l'organizzazione ne ha il completo controllo. Questo è vero, ma molto spesso non si dispone di esperti di sicurezza che possano, a tempo pieno, dedicarsi alla protezione dei dati. Sebbene le organizzazioni di grandi dimensioni siano in grado di far fronte a questo problema, quelle di dimensioni medio/piccole non possono, e questo rende, per loro, più conveniente la scelta $IDaaS_G$.

Privacy

Il tema della *privacy* è molto controverso. Se i dati sono interamente mantenuti dall'organizzazione si ha l'assoluto controllo e si è sicuri di dove risiedono. D'altro canto, se il sistema è *cloud_G* bisogna chiedersi dove sono memorizzati, come vengono trasferiti e in che giurisdizione ricadono. Se i dati sono molto sensibili è più sicuro usare sistemi *on-premises_G*. Al contrario, se l'organizzazione ha particolari requisiti di riservatezza o protezione (come un'agenzia governativa) può essere utile cercare un sistema $IDaaS_G$ che possieda questi requisiti: alcuni recenti sistemi *cloud_G*, ad esempio, soddisfano le necessità delle agenzie federali statunitensi. Infine $IDaaS_G$ consente di memorizzare i dati sotto una specifica giurisdizione: è sufficiente cercare un produttore che supporta quella voluta.

Agilità

L'agilità è il maggior punto di vantaggio dei sistemi $IDaaS_G$. Sistemi di questo tipo possono scalare con facilità. Ogni cambiamento viene fatto in gran velocità e i produttori non devono richiedere ai clienti di aggiornare per ottenere il supporto di altre funzionalità (il ciclo di rilascio si misura in settimane nel primo caso e mesi nell'altro). Sebbene molte aree siano già mature, in altre dei cicli così veloci sono un grande

vantaggio. Per questi motivi, $IDaaS_G$ è la scelta giusta se si necessita l'integrazione con applicazioni in rapida evoluzione o un rapido rilascio di nuove funzionalità.

Distribuzione dell'architettura

Dipendentemente dalle situazioni può essere più vantaggioso avere un sistema eseguito su $cloud_G$ o no. Ad esempio, se l'organizzazione non dispone di una connessione veloce, l' $IDaaS_G$ può comportare latenze significative; d'altra parte, se l'integrazione con $SaaS_G$ è una priorità l' $on-premises_G$ deve essere scartato. Oggigiorno, comunque, molte organizzazioni fanno uso di *servizi di directory_G* $on-premises_G$, quindi, se si è su $cloud_G$, sono necessari dei connettori per accedere a questi sistemi.

Per scegliere bisogna chiedersi chi sono gli utenti e da dove accedono. Se sono persone esterne all'organizzazione e devono accedere ad applicazioni su $cloud_G$ la scelta $IDaaS_G$ è assolutamente la più conveniente.

3.3.2 Maturità funzionale

Modernità dell'architettura

La maggior parte dei nuovi sistemi IAM_G è di tipo $IDaaS_G$, quindi è naturale che questi presentino delle architetture più vicine alle *best practices_G* correnti. Anche i sistemi $on-premises_G$ sono stati riprogettati per adattarsi, ma il cambiamento dell'architettura richiede pesanti cambiamenti e aggiornamenti.

Modernità dell'interfaccia

La maggior parte delle innovazioni grafiche vengono introdotte prima in sistemi $IDaaS_G$. Il mondo web è molto più dinamico, e un'interfaccia sempre aggiornata è un obbligo. Al contrario, i sistemi $on-premises_G$ continuano a rimanere ancorati alle loro scelte senza cambiare.

Integrazione con SaaS

L'integrazione con $SaaS_G$ è molto più facile e veloce per sistemi $IDaaS_G$, anche perché l'organizzazione non deve fare niente per integrare i nuovi *software*. Le alternative $on-premises_G$, invece, richiedono costi e tempo aggiuntivi per integrare nuove applicazioni.

Integrazione con applicazioni on-premises

In generale l'integrazione con applicazioni $on-premises_G$ è migliore per le soluzioni $on-premises_G$. Sebbene anche i sistemi $IDaaS_G$ stiano cercando di supportarle, se si necessita l'integrazione con un gran numero di applicazioni di questo tipo, allora la scelta migliore è ancora la prima.

Personalizzazione

La personalizzazione è il principale vantaggio dei sistemi $on-premises_G$. In teoria $IDaaS_G$ offre un approccio adatto a quasi tutte le organizzazioni grazie alla possibilità di configurare il prodotto in molti modi diversi. La realtà, però, è più confusa.

La personalizzazione comporta un cambiamento nel codice del prodotto, perché richiede l'aggiunta di funzionalità specifiche e necessarie solamente ad un ristretto numero di interessati. Sebbene i prodotti $on-premises_G$ possano essere altamente personalizzati, la personalizzazione in sé dovrebbe essere evitata e nel futuro sarà sempre meno necessaria: più è matura un'area di interesse meno personalizzazioni sono richieste e più è possibile prevedere diverse configurazioni per incontrare i bisogni di

tutti i clienti. D'altro canto, la maturità di aree dinamiche come il *cloud_G* o il *mobile* è ancora molto lontana.

Se si ritiene che un prodotto configurabile non si adatti alle necessità dell'azienda, allora l'*on-premises_G* è la scelta giusta.

Aderenza agli standard

Vista la grande diffusione di applicazioni *SaaS_G*, molti produttori di sistemi *IAM_G* tendono ad aggiungere supporto agli standard più velocemente nelle versioni *IDaaS_G* rispetto a quelle *on-premises_G*. Questa tendenza è destinata a crescere, in quanto permette di ricevere *feedbacks* più velocemente e di reagire con maggiore rapidità alla nascita di nuovi standard.

Maturità generale delle funzionalità

In media i sistemi *on-premises_G* tendono ad essere più vecchi. Di conseguenza dispongono di tecnologie superate e di funzionalità più legate ad applicazioni *on-premises_G*. *IDaaS_G*, invece, offre un'ottima integrazione con *SaaS_G* e un buon supporto di applicazioni *on-premises_G*. Nel lungo periodo ci si aspetta che nel *cloud_G* le innovazioni arrivino prima⁷, quindi *IDaaS_G* è la scelta migliore.

Flessibilità dei termini della licenza

Storicamente, i sistemi *on-premises_G* sono venduti con licenze pagate una sola volta, mentre quelli *IDaaS_G* normalmente prevedono un pagamento ogni mese e si adattano meglio alle esigenze degli utenti perché consentono di pagare solo le funzionalità volute.

Nonostante questo, anche le licenze dei *software on-premises_G* si stanno adattando e stanno diventando più flessibili, quindi la differenza tra le due alternative è poca e non c'è un vincitore netto.

3.3.3 Cosa scegliere?

La scelta tra *cloud_G* e *on-premises_G* dipende fortemente dalle necessità dell'organizzazione e dal *Total Cost of Ownership (TCO)_G*. Soluzioni *on-premises_G* sono adatte se si hanno vincoli stretti sulla residenza e sulla protezione dei dati, se si vogliono implementazioni *IAM_G* mature e personalizzabili e un forte controllo sull'amministrazione delle *identità_G* (*Identity Governance and Administration (IGA)_G*), attualmente carenti nei prodotti basati su *cloud_G*. Al contrario, sistemi *IDaaS_G* sono adatti se si cercano implementazioni *IAM_G* dinamiche e una maggiore integrazione con *SaaS_G* (Figura 3.7).

⁷Vedi [2]

⁸Immagine tratta da [7]

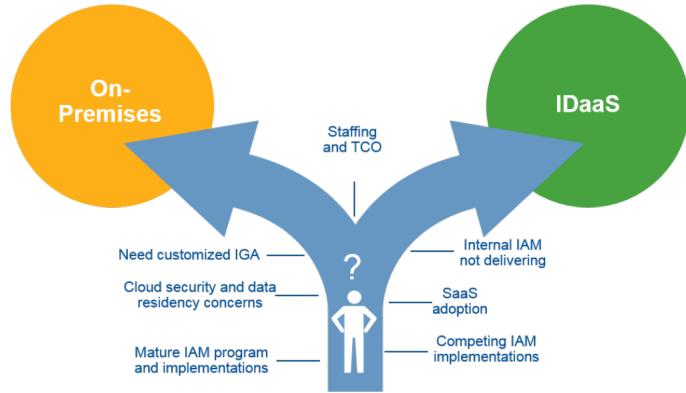


Figura 3.7: Scelta tra sistemi IDaaS e on-premises nell'ambito IAM⁸

3.4 Un esempio: Sistema Pubblico di Identità Digitale

Il *Sistema Pubblico Identità Digitale (SPID)_G* (logo in Figura 3.8) permette a cittadini e imprese di accedere con un unico *login* a tutti i servizi *online* di pubbliche amministrazioni e imprese aderenti. *SPID_G* nasce con lo scopo di favorire la diffusione di servizi *online* e di agevolarne l'utilizzo da parte di cittadini e imprese.

Il modello proposto da *SPID_G* segue un approccio federato di aziende private e accreditate per la fornitura dei servizi di *identità_G* digitale. Cittadini e imprese possono scegliere il loro fornitore di *identità_G* preferito.



Figura 3.8: Logo di SPID⁹

3.4.1 Attori e ruoli

Lo *SPID_G* identifica differenti ruoli:

- **Fornitori di *identità_G* digitale (o *Identity Provider_G*):** sono imprese accreditate dall'*Agenzia per l'Italia Digitale (AgID)_G* con il compito di identificare l'utente in modo certo, di creare le *identità_G* digitali, di assegnare le credenziali e di gestire gli attributi dell'utente. Devono garantire la correttezza dell'*identità_G* digitale e la riservatezza delle informazioni.

⁹Fonte: <http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/spid/percorso-attuazione>

- **Fornitori di servizi** (o *Service Provider_G*): privati o pubbliche amministrazioni che erogano servizi *online*.
- **Utente**: titolare di un'*identità_G* digitale *SPID_G* che utilizza i servizi erogati dai fornitori di servizi.
- **Gestore di attributi qualificati** (o **Attribute Provider**): ha il potere di attestare gli attributi qualificati su richiesta dei fornitori di servizi.
- **Agenzia**: organismo di vigilanza che si occupa di gestire l'accreditamento e di monitorare i gestori dell'*identità_G* digitale e i gestori di attributi qualificati.

3.4.2 Come funziona

Un utente si registra al servizio tramite un *IdP_G* che crea un'*identità_G* digitale e gli assegna le credenziali per il riconoscimento. L'utente può utilizzare la sua *identità_G* digitale per l'accesso ai servizi *online* offerti dai *SP_G*, che sono collegati a tutti gli *IdP_G*. Sarà compito dell'*IdP_G* verificare la correttezza dei dati di *login* immessi dall'utente e fornire al *SP_G* solo gli attributi dell'utente strettamente necessari alla fornitura del servizio.

SPID_G infatti introduce il concetto di informazioni necessarie e sufficienti per il servizio. I fornitori del servizio potranno richiedere solamente le informazioni minime necessarie all'erogazione del servizio stesso, come si legge in [14]:

I fornitori di servizi, per verificare le policies di sicurezza relativi all'accesso ai servizi da essi erogati potrebbero avere necessità di informazioni relative ad attributi riferibili ai soggetti richiedenti. Tali policies dovranno essere concepite in modo da richiedere per la verifica il set minimo di attributi pertinenti e non eccedenti le necessità effettive del servizio offerto e mantenuti per il tempo strettamente necessario alla verifica stessa, come previsto dall'articolo 11 del decreto legislativo n. 196 del 2003.

3.4.3 Livelli di sicurezza

In Figura 3.9 sono mostrati i tre livelli di sicurezza di *SPID_G*.



Figura 3.9: Livelli di sicurezza SPID¹⁰

¹⁰Fonte: <http://www.spid.gov.it/>

- **Livello 1:** garantisce con un buon grado di affidabilità dell' $identità_G$ accertata nel corso dell'attività di $autenticazione_G$. A tale livello è associato un rischio moderato e compatibile con l'impiego di un sistema $autenticazione_G$ a singolo fattore, ad es. la *password*; questo livello può essere considerato applicabile nei casi in cui il danno causato da un utilizzo indebito dell' $identità_G$ digitale ha un basso impatto per le attività del cittadino/impresa/amministrazione. Per il livello 1 la credenziale sarà dunque una *password* di almeno 8 caratteri, da rinnovarsi ogni 180 giorni, formulata secondo i consueti criteri di sicurezza.
- **Livello 2:** garantisce con un alto grado di affidabilità dell' $identità_G$ accertata nel corso dell'attività di $autenticazione_G$. A tale livello è associato un rischio ragguardevole e compatibile con l'impiego di un sistema di $autenticazione_G$ informatica a due fattori non necessariamente basato su certificati digitali; questo livello è adeguato per tutti i servizi per i quali un indebito utilizzo dell' $identità_G$ digitale può provocare un danno consistente. Per il livello 2, oltre alla *password* sarà necessario inserire il codice proveniente da un dispositivo a chiave variabile (ad esempio una *One Time Password*) che potrebbe essere anche un'applicazione sul cellulare.
- **Livello 3:** garantisce con un altissimo grado di affidabilità dell' $identità_G$ accertata nel corso dell'attività di $autenticazione_G$. A tale livello è associato un rischio altissimo e compatibile con l'impiego di un sistema di $autenticazione_G$ informatica a due fattori basato su certificati digitali e criteri di custodia delle chiavi private su altri dispositivi; questo è il livello di garanzia più elevato e da associare a quei servizi che possono subire un serio e grave danno per cause imputabili ad abusi di $identità_G$. È anche interessante notare come la definizione di "dispositivo" includa sia sistemi di tipo *hardware* sia di tipo *software* (ad esempio sono tali i generatori di *password* attraverso applicazioni per *smartphone*).

3.4.4 Come ottenere l'identità digitale

Gli attori che possono assumere il ruolo di IdP_G sono molteplici (banche, operatori di telefonia mobile, *certification authority*, fornitori di soluzioni IT_G) e giocano un ruolo fondamentale nel decretare il successo del sistema perché portano in "dote" allo $SPID_G$ il proprio bacino di utenti potenziali. La condizione ottimale è che il numero di IdP_G sia sufficientemente elevato per raggiungere il maggior numero di utenti, e contemporaneamente limitato per minimizzare il numero di relazioni tra SP_G e IdP_G .

Il processo di richiesta dell' $identità_G$ e di $autenticazione_G$ è fondamentale perché un'esperienza utente eccessivamente complicata potrebbe scoraggiare gli utenti che vorrebbero aderire al servizio.

3.4.5 Vantaggi

I SP_G che aderiscono allo $SPID_G$, sia pubbliche amministrazioni sia imprese private, possono disporre di un parco utenti senza censirli, non hanno gli oneri derivanti dalla conservazione dei dati personali e non devono preoccuparsi di evitare attacchi volti al furto delle credenziali. Inoltre, i SP_G possono avere profili con un' $identità_G$ certa, eliminando i cosiddetti "falsi profili", ed univoca, eliminando i duplicati.

Per gli utenti, $SPID_G$ consente di semplificare la vita di cittadini e imprese nell'interazione con la pubblica amministrazione tramite servizi *online* grazie ad un unico *login*. Il sistema garantisce la massima sicurezza e *privacy*. Il SP_G non può conservare

i dati dell'utente che riceve dall' IdP_G ed è assolutamente vietata la tracciatura delle attività di un individuo.

Capitolo 4

Il progetto

4.1 Monokee

Monokee è un sistema $IDaaS_G$ nato con lo scopo di realizzare una corretta gestione degli utenti e delle loro autorizzazioni all'interno di sistemi informativi eterogenei. Utilizzandolo, ciascun utente sarà in grado di gestire in modo centralizzato l'accesso in SSO_G ad applicazioni differenti. In parole povere, Monokee assicura che le persone giuste accedano alle risorse di propria competenza.

Ogni utente di Monokee ha a disposizione un **domain broker** (Figura 4.1) in cui sono elencati i diversi domini ad esso associati. Un dominio è uno spazio di un web server che identifica in maniera precisa il nome di un privato, un ente o un'organizzazione su Internet. Il dominio **personale** è obbligatorio: qui sono presenti le applicazioni personali; ci possono poi essere uno o più domini di tipo **company**, ovvero relativi ad aziende registrate al servizio.

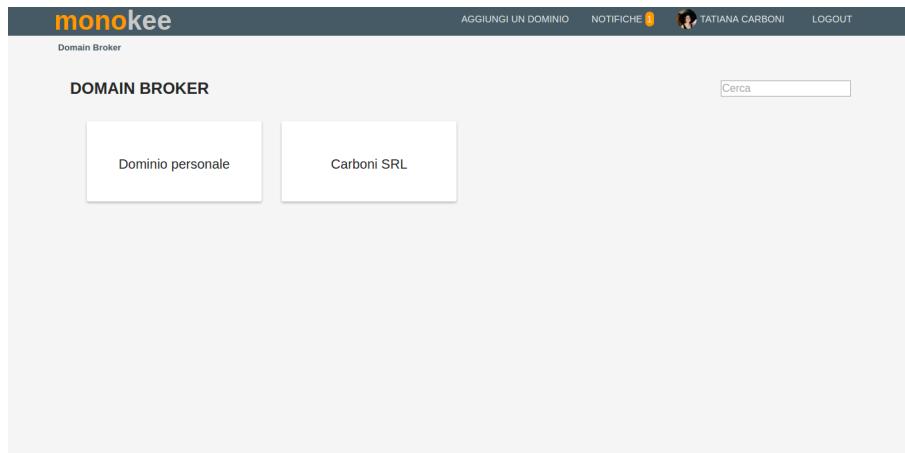


Figura 4.1: Domain broker di Monokee

Le utenze sono quindi di due tipi differenti:

- **consumer:** con il solo dominio personale (il rapporto utente:domini è 1:1);

- **company**: oltre al dominio personale hanno anche la possibilità di avere domini aziendali (il rapporto utente:domini è 1:N).

Attualmente i concetti di dominio e *domain broker* sono propri di Monokee: nessun *competitor* e nessun altro sistema di IAM_G supporta una gestione di questo tipo. In generale, infatti, il problema viene risolto imponendo all'utente di registrarsi più volte al sistema, creando un evidente controsenso per un sistema di gestione delle *identità_G*.

Accedendo ad uno dei domini a sua disposizione, l'utente può visualizzare l'**application broker**, ovvero un elenco delle applicazioni associate a quel dominio (Figura 4.2). Da qui è possibile, selezionando un'applicazione, effettuare l'accesso in SSO_G .

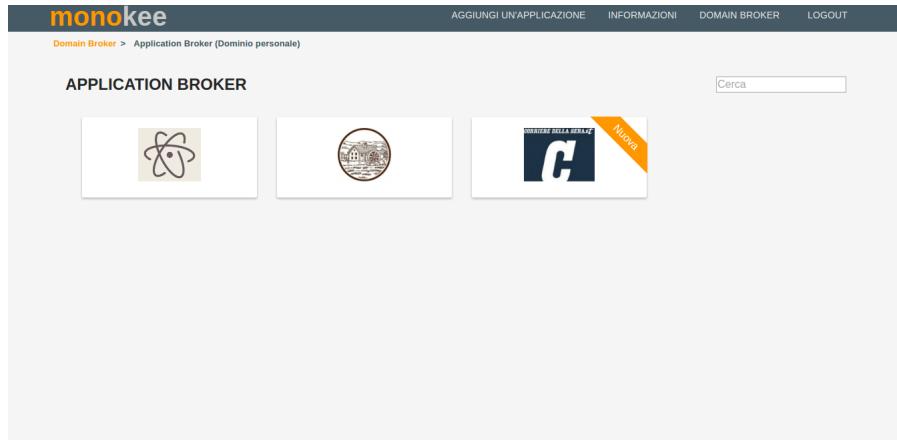


Figura 4.2: Application broker di Monokee

Se i permessi associati all'*account* lo consentono si possono inoltre gestire le applicazioni dell'*application broker* rimuovendo quelle presenti o aggiungendone altre dal **catalogo** (illustrato in Figura 4.3).

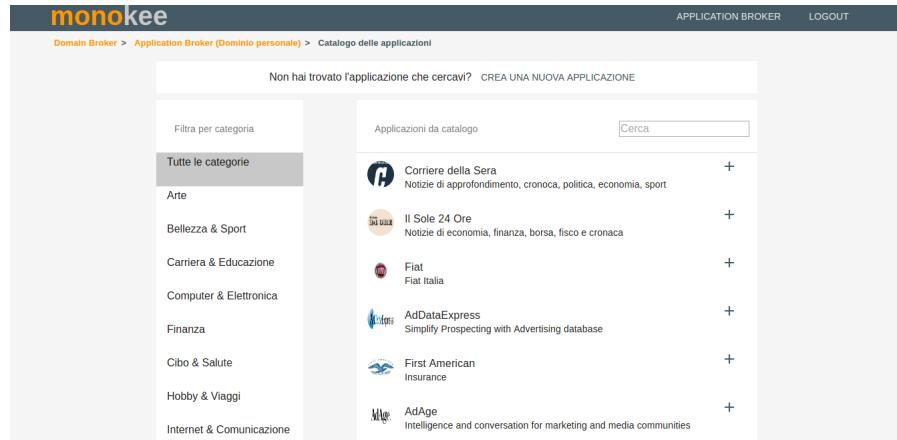


Figura 4.3: Catalogo applicativo di Monokee

Infine, se il dominio è di tipo *company*, un utente amministratore può gestire gli utenti, le applicazioni ed eventuali gruppi appartenenti al dominio.

4.1.1 I moduli

L'architettura base di Monokee si compone di sette moduli che interagiscono tra di loro per il corretto funzionamento del servizio:

- **Front end:** responsabile delle interazioni con l'utente e scritto utilizzando Angular.js;
- **Plug in:** ha una duplice funzione: da un lato permette di inserire nuove applicazioni non presenti del catalogo attraverso una fase di **learning** della struttura della pagina web, e dall'altro consente l'accesso in SSO_G alle applicazioni tramite il **form fulfillment**;
- **Identity Provider_G:** si occupa dell'accesso di tipo federato $SAML_G$ elaborando le richieste del *Service Provider_G* (**SAMLRequest**) e producendo l'appropriata **SAMLResponse** consentendo o meno l'accesso all'applicazione. È scritto in Java;
- **AD_G Integration:** consente l'integrazione del sistema con uno o più *servizi di directory_G* di tipo AD_G e si occupa di effettuare periodicamente delle *query* volte a replicare l'intera struttura delle utenze nel *database* del sistema inviando una copia di tutti i dati al modulo *Core*; questo strumento garantisce a Monokee di operare sempre sull'ultima versione aggiornata delle utenze;
- **Mobile:** applicazione *mobile* per *smartphone* e *tablet*;
- **Service Provider_G:** si occupa della generazione di *SAMLRequest* per erogare dei servizi. Il SP_G di Monokee è utilizzato per l'accesso all'applicazione di gestione del catalogo, obiettivo dell'attività di stage. Così come l' IdP_G , è scritto in Java;
- **Core:** insieme di servizi **R***Epresentational State Transfer (REST)_G su protocollo $HTTP_G$ che ricevono, ed elaborano, i dati provenienti dagli altri moduli, si occupano della loro gestione e memorizzazione nel *database*, producono ed inviano le risposte in base alle situazioni e salvano i *log* di quanto è stato fatto.*

Un ulteriore modulo, obiettivo dell'attività di stage, verrà descritto nella successiva sezione.

4.2 Monokee Catalogue Manager

Lo scopo dell'attività di stage è quello di realizzare la parte di back end del gestore del catalogo applicativo dell'applicazione Monokee.

La fonte principale delle applicazioni accedibili in SSO_G è il catalogo: attraverso di esso è possibile cercare e aggiungere applicazioni al proprio dominio. Il prodotto sviluppato permetterà la gestione, lato amministratore, del catalogo stesso, con le principali funzionalità di aggiunta, rimozione e configurazione.

4.2.1 Funzionalità principali

Attraverso il catalogo applicativo sarà possibile gestire le applicazioni utilizzabili in modo "predefinito" in Monokee. In particolare, le funzioni principali saranno:

- gestione di un'applicazione:

- aggiunta e configurazione;
- rimozione;
- modifica della configurazione;
- categorizzazione delle applicazioni;
- visualizzazione e ricerca delle applicazioni;
- gestione dei cataloghi specifici dei domini:
 - aggiunta di un catalogo ad un dominio *company*;
 - rimozione di un catalogo da un dominio *company*;
 - aggiunta di un'applicazione al catalogo;
 - rimozione di un'applicazione dal catalogo;
- gestione dei gruppi di applicazioni:
 - aggiunta di un gruppo;
 - rimozione di un gruppo;
 - modifica di un gruppo;
 - aggiunta di un'applicazione al gruppo;
 - rimozione di un'applicazione dal gruppo;
- visualizzazione di statistiche sul numero di applicazioni, gruppi e utenti;
- visualizzazione dei *log* delle operazioni svolte dagli utenti e degli errori riscontrati durante l'esecuzione di queste operazioni.

Principalmente, quindi, le entità coinvolte sono tre:

1. applicazione;
2. gruppo;
3. categoria.

Applicazione

Le applicazioni possono essere di due tipi:

- pubbliche;
- private, ovvero specifiche di un dominio aziendale.

È presente, di base, un catalogo pubblico, contenente tutte le applicazioni pubbliche. Accanto a questo possono esistere anche dei cataloghi di dominio (un catalogo per ogni dominio aziendale esistente): questi raccolgono le applicazioni private. Così facendo è possibile inserire applicazioni specifiche di un'azienda (anche raggiungibili solamente attraverso la rete *ethernet* aziendale) e permettere agli utenti di quell'azienda di usufruirne.

Indipendentemente dal fatto che siano pubbliche o private, gli utenti di Monokee potranno accedere alle applicazioni in SSO_G . Attualmente, l'autenticazione_G può avvenire in tre modi:

- **form-based**, attraverso due differenti modalità:
 - **form fulfillment**: una fase di *learning* specifica per applicazione e *browser* effettuata grazie a dei *plug in* già sviluppati consente di "istruire" Monokee sulla struttura della pagina dell'applicazione web per poter eseguire il SSO_G "riempiendo" il *form* di accesso;
 - **richieste POST realizzate tramite *Asynchronous JavaScript and XML (AJAX)*_G**.
- **federata**, con l'utilizzo dello standard $SAML_G$ 2.0;
- **accesso di terzo tipo**, ad applicazioni di terze parti che necessitano una richiesta POST $AJAX_G$ completa di *headers* $HTTP_G$.

Esiste, in realtà, anche un quarto tipo di applicazione supportato da Monokee, ma non è rilevante per la discussione in quanto le applicazioni di questo tipo non possono essere gestite da Catalogue Manager.

Le regole di accesso specifiche delle applicazioni dovranno quindi tenere conto della modalità di *autenticazione_G*. Inoltre, l'accesso tramite SSO_G comporta il dover conoscere l'**Uniform Resource Locator (URL)_G** della pagina di *login* dell'applicazione. Questa pagina può variare da *browser* a *browser*, come già citato, ma anche da Paese a Paese (**localizzazione**): per una stessa applicazione di base possono essere quindi memorizzate più pagine di *login*, dipendentemente dal Paese "bersaglio". Un esempio è dato da Amazon (Figura 4.4):

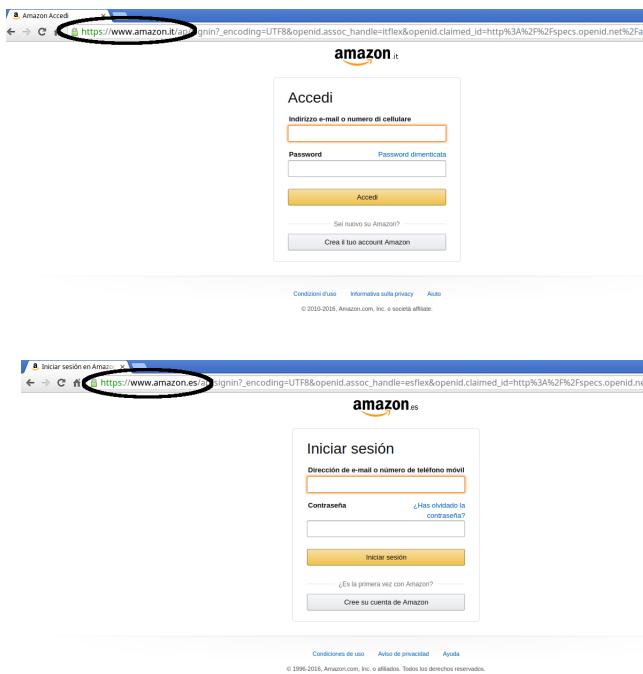


Figura 4.4: Confronto fra le pagine di login di Amazon Italia e Amazon Spagna

Come si può notare, nell' URL_G i due *host* sono diversi (www.amazon.it in un caso, www.amazon.es nell'altro). Questo porta a dover memorizzare, per ogni applicazione localizzata, una lista delle diverse localizzazioni e dei corrispondenti URL_G per il *login*. Dal punto di vista dell'utente, ogni localizzazione è considerata un'applicazione a sé stante. Nel caso appena citato, ad esempio, sarebbero presenti due diverse "versioni" di Amazon, Amazon Italia e Amazon Spagna. Ovviamente l'utente potrà inserire nel suo dominio entrambe le versioni, qualora voglia autenticarsi sia al sito italiano che a quello spagnolo.

Appena creata, l'applicazione non è **pubblicata**: questo significa che gli utenti di Monokee non possono vederla, in quanto i dati necessari al SSO_G non sono ancora stati impostati. Una volta pubblicata, l'applicazione è visibile nel catalogo. Può accadere, però, che sia necessario il cambiamento di alcuni dati in seguito ad una modifica dell'applicazione. Quest'ultima può quindi essere messa, momentaneamente, in **manutenzione** in modo da poter modificare i dati necessari e avvisare gli utenti che l'autenticazione potrebbe non funzionare correttamente.

Gruppo

Applicazioni con caratteristiche simili (ad esempio nel caso della localizzazione sopra citata) possono essere raggruppate per consentire una migliore gerarchizzazione e per generare meno confusione nell'utente. I gruppi possono essere creati per applicazioni sia pubbliche che private, e aiutano gli amministratori del catalogo a mantenere ordine tra le applicazioni presenti. Non c'è limite al numero di applicazioni che possono essere inserite in un gruppo, ma un'applicazione può essere inserita in un solo gruppo per volta.

Così come per le applicazioni, anche i gruppi possono essere pubblici o privati, ma in un gruppo privato non possono essere inserite applicazioni pubbliche e viceversa.

Categorie

Ogni applicazione deve appartenere ad una o più categorie, in modo da rendere il più semplice possibile la ricerca nel catalogo. Il sistema di categorizzazione è più vicino ad un sistema di "etichette": la categorizzazione non è, infatti, mutuamente esclusiva e può accadere (e in generale accade) che ad un'applicazione siano associate più categorie. Questo è un esempio del cosiddetto "**schema ambiguo per argomento**". Schemi di questo genere sono più difficilmente manutenibili da parte degli sviluppatori, perché richiedono un costante lavoro di controllo e aggiornamento, ma risultano molto più utilizzabili da parte degli utenti, in quanto non è necessario sapere esattamente cosa si sta cercando.

Considerato che Monokee può essere usato sia per uso personale che per uso aziendale si è resa necessaria una profonda ad approfondita categorizzazione delle applicazioni web. Tale categorizzazione tiene conto di entrambi gli utilizzi previsti di Monokee e questo ha portato all'ottenimento di un alto numero di "etichette". Tuttavia, un utente di Monokee rischierebbe di essere confuso e disorientato da un numero elevato di categorie: è stato pertanto fatto un ulteriore lavoro di raggruppamento, in modo da definire le seguenti 13 "sovra categorie":

- Arts;
- Beauty & Sport;
- Career;
- Computer & Electronics;

- Education;
- Finance;
- Food & Health;
- Hobby & Travel;
- Internet & Communication;
- News;
- People & Pets;
- Productivity;
- Vehicles.

Queste ultime vengono mostrate nella pagina di aggiunta di un'applicazione all'*application broker*, come mostrato in Figura 4.3. Le "etichette", invece, vengono mostrate all'utente del gestore del catalogo che, essendo un amministratore di Monokee, ha una maggiore conoscenza del mondo web e non rischia di essere sopraffatto dal gran numero di categorie. Di seguito vengono indicate le sovra categorie, complete delle categorie mostrate agli utenti del gestore del catalogo di Monokee:

- **Arts:**
 - Animation & Comics;
 - Arts & Entertainment;
 - Movies;
 - Music & Audio;
 - Photography.
- **Beauty & Sport:**
 - Beauty;
 - Sports.
- **Career:**
 - Jobs & Employment.
- **Computer & Electronics:**
 - Computer Hardware;
 - Computer Security;
 - Electronics;
 - Multimedia;
 - Networking;
 - Programming;
 - Software.
- **Education:**

- Book Retailers;
- Dictionaries;
- Ebooks;
- Education;
- Science;
- Universities.

- **Finance:**

- Banking;
- Financial Investing;
- Financial Management;
- Insurance.

- **Food & Health:**

- Food Delivery;
- Health;
- Nutrition;
- Restaurants.

- **Hobby & Travel:**

- Games;
- Hotels;
- Maps;
- Online Games;
- Recreations & Hobbies;
- TV & Video;
- Video Games.

- **Internet & Communication:**

- Chat;
- ECommerce;
- Email;
- File Sharing;
- Forum & Blog;
- Search Engine;
- Social Networks;
- Web Hosting.

- **News:**

- News;
- Newspapers;

- Weather.
- **People & Pets:**
 - Pets;
 - Relationship & Dating.
- **Productivity:**
 - Collaboration;
 - *Customer Relationship Management (CRM)_G*;
 - Data Analysis;
 - *Enterprise Resource Planning (ERP)_G*;
 - Human Resources;
 - Supply Chain Management;
 - Transportation.
- **Vehicles:**
 - Aviation;
 - Boating;
 - Car Buying & Rentals;
 - Cars;
 - Motorcycles;
 - Railways.

4.3 Il futuro ed i competitors

Quello delle soluzioni *IDaaS_G* è un mercato recente ed in forte espansione, introdotto dagli analisti del gruppo Gartner come categoria chiave dell'*IT_G* solamente nel 2014. Confrontando i due *report* (corredati dai celebri *Magic Quadrant*, figure 4.5a e 4.5b) prodotti da Gartner nel giugno 2015 e giugno 2016, infatti, si può notare come, nell'arco di un solo anno, molte aziende abbiano deciso di investire in tale ambito sviluppando e facendo crescere i propri prodotti. Particolarmente significativa è la presenza e la crescita di molte compagnie *leader* nel settore *IT_G*, quali, ad esempio, Microsoft, Salesforce e IBM.

Nonostante ciò, e in particolare nonostante la crescita di Microsoft e Centrify, Gartner ha identificato, anche nel 2016, Okta come principale *leader* del mercato *IDaaS_G*. Okta, Inc. è stata fra le prime aziende a capire il potenziale dell'emergente ambito dell'*IDaaS_G*. Ciò che ha determinato il suo successo e l'ha eletta *leader* anche per il 2016 è l'ampia gamma di funzionalità offerte dal suo prodotto, la semplicità nell'utilizzo e l'impegno nel supporto e nell'integrazione del suo sistema con *servizi di directory_G* aziendali, come *AD_G*.

Gartner stabilisce dei "requisiti minimi" per poter partecipare alla ricerca, e divide i partecipanti nelle quattro aree che formano il *Magic Quadrant*:

²Fonse: <http://get.onelogin.com/GartnerMQ-2015.html>

²Immagine tratta da [10]

- **leaders:** i molti clienti sono soddisfatti e le funzionalità offerte sono al passo con le richieste del mercato. I servizi offerti sono molti e il supporto è di alto livello;
- **challengers:** sono sulla buona strada per diventare *leaders*, ma la loro visione complessiva dell'*IDaaS_G* è troppo limitata o focalizzata solo su un'area specifica. I clienti sono generalmente soddisfatti, ma devono fare richieste specifiche per nuove funzionalità;
- **visionaries:** i loro prodotti incontrano i bisogni della maggior parte dei clienti, ma questi ultimi non sono sufficienti a renderli dei *leaders*. Sono molto innovativi, e spesso presentano funzionalità che gli altri non hanno;
- **niche players:** i loro prodotti sono adatti a specifici casi d'uso, come particolari settori industriali. Hanno di solito pochi clienti, e i prezzi sono troppo alti per permettere di essere competitivi sul mercato. Ad ogni modo, nel loro ambito possono essere molto conosciuti ed apprezzati.

Posizionamento di Monokee

Nello sviluppo del suo prodotto, *iVoxIT S.r.l* ha analizzato a fondo le funzionalità offerte dalle aziende *competitor* ed ha dedicato particolare attenzione ad Okta quale riferimento del mercato. Lo studio di tali realtà ha portato all'individuazione di alcuni punti critici attorno ai quali l'azienda si è concentrata al fine di trovare delle soluzioni efficaci ed innovative che le permettano di distinguersi nell'ambito *IDaaS_G*. Un esempio è la gestione dei domini già citata. In particolare, l'azienda punta ad affermarsi come realtà importante in ambito europeo, collocandosi in uno spazio che le permetta ampie possibilità di sviluppo e crescita (attualmente, fra le aziende indicate nel *Magic Quadrant* di Gartner, solo iWelcome opera nativamente in Europa).

(a) Giugno 2015¹(b) Giugno 2016²**Figura 4.5:** Gartner Magic Quadrants

Capitolo 5

Tecnologie, strumenti e linguaggi utilizzati

5.1 Tecnologie

Di seguito saranno presentate le tecnologie principali utilizzate durante l'attività di stage. Lo *stack* tecnologico usato è denominato **MEAN** (Figura 5.1):

- MongoDB, *database* non relazionale;
- Express.js, *framework_G* per applicazioni web eseguite su Node.js;
- Angular.js, *framework_G* JavaScript eseguito su *browser*;
- Node.js, ambiente di esecuzione lato server per applicazioni ad eventi e relative comunicazioni.

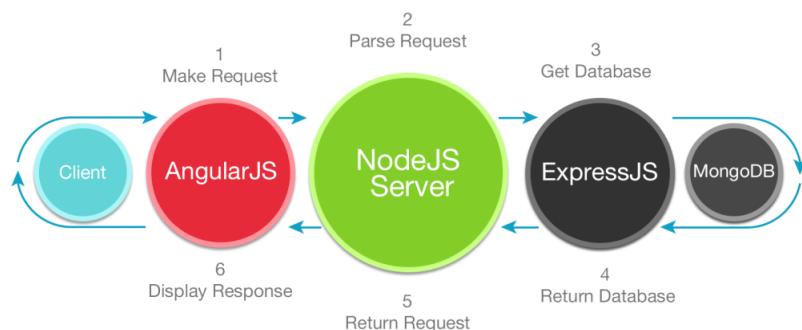


Figura 5.1: Funzionamento dello stack MEAN¹

In particolare, durante l'attività di stage sono state utilizzate le tecnologie lato server, quindi MongoDB, Express.js e Node.js, che saranno descritte in seguito.

¹Fonente: <http://trinathswebapps.blogspot.it/2016/01/history-of-angularjs.html>

5.1.1 Node.js

Asincronismo

In Figura 5.2 viene mostrato un confronto tra un server Java e uno Node.js. La differenza principale è che Node.js permette la realizzazione di **server asincroni non bloccanti**. Lo stile architetturale orientato agli eventi lo rende particolarmente adatto a catturare la reattività di un'applicazione web.

Quando un server viene invocato in modo sincrono, l'applicazione client che l'ha invocato deve attendere la risposta prima di poter continuare la sua esecuzione. Se la risposta è immediata va tutto bene, ma è ragionevole supporre che la maggior parte delle applicazioni eseguite da un server impieghino del tempo per essere eseguite, con conseguente rallentamento dell'intero sistema.

Se il server viene invocato in modo asincrono, però, il client non deve aspettare la risposta del server e può proseguire con l'esecuzione.

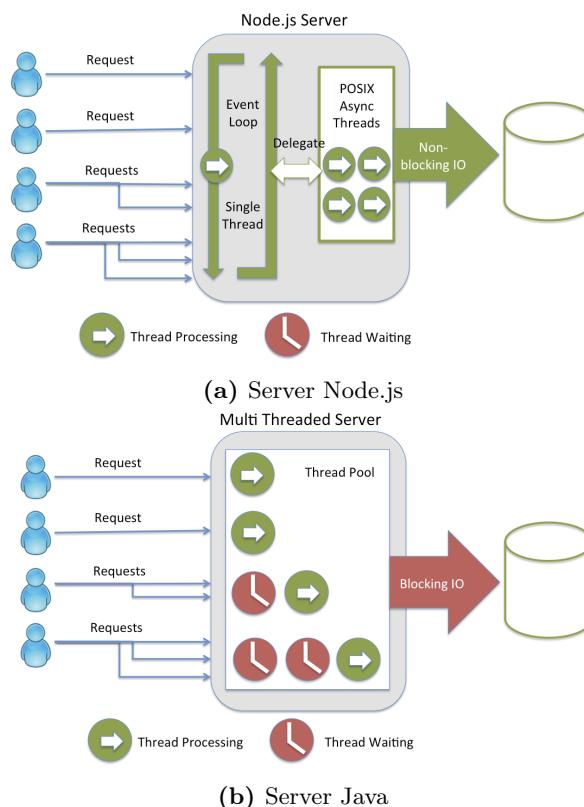


Figura 5.2: Confronto tra server Node.js e server Java²

In Figura 5.3 viene mostrato il funzionamento di un server Node.js nel contesto di una richiesta di **I/O_G** . Le funzionalità di I/O_G sono molto importanti nell'architettura di Node.js. Un unico $thread_G$ (il **main $thread_G$**) rimane in ascolto di connessioni. Quando si riceve una richiesta il $thread_G$ esegue l'evento richiesto, qualunque esso sia. Tuttavia, se l'operazione richiede del tempo, come una lettura da

²Fonete: <https://strongloop.com/strongblog/node-js-is-faster-than-java/>

file system_G o da *database*, viene aperto un altro *thread_G* (il **worker thread_G**) che si occupa di eseguire quanto richiesto. Questo meccanismo di delega avviene attraverso l'uso di una funzione di *callback*, ovvero una funzione invocata al termine dell'esecuzione di un'altra. Grazie a questo meccanismo di "chiamata all'indietro" il *thread_G* delegato può notificare al *thread_G* principale il completamento dell'operazione richiesta. Nel frattempo quest'ultimo è libero di continuare con l'esecuzione del programma principale.

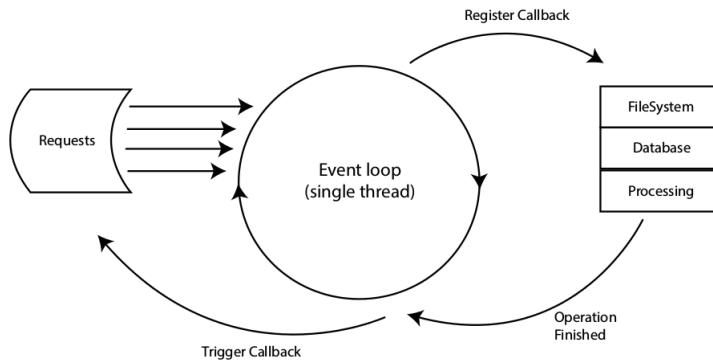


Figura 5.3: Funzionamento di un server Node.js³

Questo meccanismo asincrono migliora notevolmente le *performance* e l'utilizzo delle risorse, ma presenta anche degli aspetti negativi. Il controllo del flusso di esecuzione risulta più complesso, e l'eccessivo uso di *callback* può portare ad un rallentamento.

Comunque sia, per applicazioni web molto orientate all'*I/O_G* l'uso di Node.js, rispetto ad un tradizionale server Java, consente un miglioramento complessivo delle *performance* e ad una riduzione dei tempi di attesa.

Performance

In Figura 5.4 si riporta un confronto di prestazioni tra un server Node.js e uno Java effettuato da Paypal ([28]) e basato sulla stessa applicazione scritta nei due modi diversi.

La prima differenza evidente riguarda il **numero di richieste al secondo**: Node.js riesce a gestirne il doppio rispetto a Java, e soprattutto lo fa dimezzando il **tempo di risposta medio** (questa è la seconda notevole differenza).

³Immagine tratta da [13]

⁴Immagine tratta da [28]

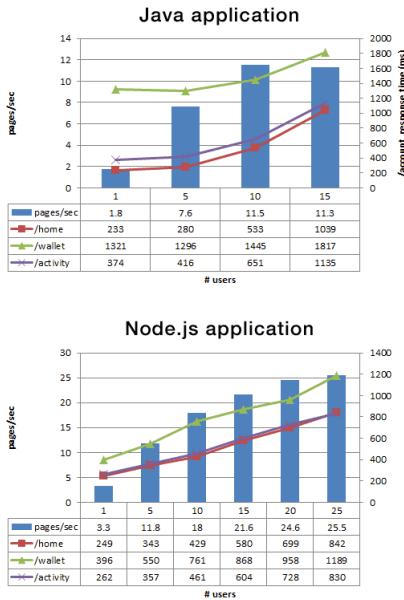


Figura 5.4: Confronto di prestazioni tra Java e Node.js⁴

5.1.2 Express.js

Per lo sviluppo del back end si è fatto uso di **Express.js** (logo in Figura 5.5), un *framework*_G che costituisce lo standard *de facto* per la creazione di servizi *REST*_G in ambiente Node.js. Le sue caratteristiche principali sono il **routing** e le funzioni **middleware**.



Figura 5.5: Logo di Express.js⁵

Routing

Express.js fornisce un metodo semplice per gestire gli *endpoint* a cui l'applicazione risponde. Per creare una *route* è sufficiente utilizzare l'oggetto **Router**, specificando a quale metodo *HTTP*_G (**GET**, **POST**, **PUT** o **DELETE**) e *Uniform Resource Identifier (URI)*_G rispondere. In risposta all'invocazione è possibile inviare qualsiasi tipo di dato, dai più semplici ai più complessi, utilizzando i metodi che l'oggetto **Response** di Express.js fornisce.

⁵Fonte: <http://expressjs.com/>

Middleware

Per *middleware* si intendono delle strutture che permettono di astrarre funzionalità locali in funzionalità distribuite. In Express.js, le funzioni *middleware* permettono di eseguire delle operazioni prima, durante o dopo l'esecuzione del codice associato all'*endpoint* invocato, modificando gli oggetti associati alla richiesta, terminando l'invocazione o chiamando un altro *middleware*.

Possono essere di vario tipo:

- a livello di **applicazione**: definiti per ogni possibile *endpoint*;
- a livello di **router**: definiti quindi per gli *endpoints* associati a quel *router*;
- per **gestione degli errori**;
- di **terze parti**: possono essere importati per aggiungere delle funzionalità all'applicazione.

5.1.3 MongoDB

Il *DataBase Management System (DBMS)*_G utilizzato è **MongoDB** (logo in Figura 5.6).



Figura 5.6: Logo di MongoDB⁶

L'integrazione con Node.js è effettuata grazie al modulo **mongoose.js**, che semplifica le operazioni che coinvolgono il database aggiungendo metodi per validare e controllare i dati e per eseguire *query*.

Per facilitare la lettura dei dati, inoltre, è stato utilizzato **Robomongo**, che verrà descritto in seguito.

Confronto con un database relazionale

MongoDB è un *database* non relazionale (o *Not only SQL (NoSQL)*_G) appartenente alla famiglia dei *database orientati ai documenti*: è in grado quindi di memorizzare oggetti con struttura complessa e non fissata inizialmente. I **documenti** memorizzati vengono organizzati in **collezioni**, con la possibilità di definire indici di vario tipo per velocizzare le ricerche ed esprimere vincoli. L'utilizzo di tali indici è inoltre necessario per mantenere le relazioni tra collezioni di documenti diversi.

In Tabella 5.1 è mostrato un confronto di terminologia tra i due tipi di *database*.

Tabella 5.1: Confronto di terminologia tra un database relazionale e MongoDB

Database relazionale	MongoDB
Database	Database
Tabella	Collezione
Riga	Documento
Colonna	Campo

Caratteristiche principali

Le caratteristiche principali di MongoDB sono:

- **Prestazioni elevate:** MongoDB riesce a garantire alte prestazioni nella persistenza dei dati. Fornisce modelli che riducono le operazioni di I/O_G nel *database* e consente la creazione di indici su qualsiasi tipo di dato.
- **Alta disponibilità:** MongoDB è provvisto di mezzi di replica, chiamati *replica sets*, ovvero gruppi di server che mantengono lo stesso insieme di dati.
- **Scalabilità automatica:** MongoDB scala, automaticamente, in orizzontale, ovvero aumenta le prestazioni aggiungendo altre macchine.

5.2 Strumenti

Nel corso del progetto sono stati utilizzati numerosi strumenti di supporto per facilitare lo svolgimento delle diverse attività, dal controllo di versione all'analisi del codice scritto. La scelta si è basata su un attento studio delle funzionalità offerte, sul supporto presente *online* e sulla facilità di utilizzo, in modo da assicurare un'alta qualità. Di seguito saranno descritte le principali funzionalità di ogni strumento.

5.2.1 Git

Git (logo in Figura 5.7) è un *software* di controllo di versione distribuito creato da Linus Torvalds nel 2005. Il controllo di versione permette di tener traccia di tutte le versioni di un progetto, con la possibilità di ripristinare un file o un intero lavoro ad uno stato precedente, evitando quindi la perdita dei dati e le sovrascritture accidentali.

Le sue principali caratteristiche sono:

- supporto allo sviluppo non lineare (*branching_G* e *merging_G*);
- sviluppo distribuito: ad ogni sviluppatore viene fornita una copia locale dell'intera cronologia di sviluppo. Quasi tutte le operazioni fatte da Git sono in locale, quindi non è soggetto a latenze di rete.
- gestione efficiente di grandi progetti: git è veloce e scalabile;
- gestione della cronologia: il nome di un *commit* dipende dall'intera cronologia di sviluppo che ha condotto a quel *commit*.

⁷Fonte: <https://git-scm.com/> ([22])



Figura 5.7: Logo di Git⁷

5.2.2 Bitbucket

Bitbucket (logo in Figura 5.8) è un sistema web di *hosting*, dedicato a progetti che usano Mercurial o git per il controllo di versione. Permette di avere un *account* gratuito e un numero illimitato di *repositories*_G private con la possibilità di aggiungere al progetto fino a cinque membri del *team*. Bitbucket offre un sistema di discussione su codice sorgente con commenti in linea, visualizzazione per *branch* o *tag* per vedere i progressi del *team* e richieste di *pull*.



Figura 5.8: Logo di Bitbucket⁸

5.2.3 JSHint

JSHint è uno strumento di analisi statica del codice usato per controllare se il codice JavaScript è conforme ad alcune regole di codifica. È presente sia una versione installabile come modulo di Node.js che una *online*, oltre ad una grande varietà di *plugins* per i principali *editor* di testo.

5.2.4 Mocha.js

Mocha.js è un *framework*_G di *test* per JavaScript eseguito su ambiente Node.js. Tra le sue caratteristiche principali troviamo:

- supporto per i *test* su *browser*;
- *testing* asincrono;
- report sulla copertura dei *test*;
- utilizzo di una qualsiasi libreria di asserzioni.

In particolare, come libreria di asserzioni è stato usato Express.js.

⁷Fonente: <https://bitbucket.org/> ([19])

⁸Fonente: <https://mochajs.org/>



Figura 5.9: Logo di Mocha.js

5.2.5 DHC

DHC (logo in Figura 5.10) è uno strumento per rendere più semplice l'uso e il *testing* delle risorse $HTTP_G/REST_G$ ed è disponibile come *plug in* nel *browser* Google Chrome. Oltre alla sua funzione principale: inviare o ricevere rispettivamente richieste o risposte $HTTP_G/REST_G$, esso permette di salvare in modo permanente una richiesta e le sue variabili in un *repository_G* locale per un successivo riutilizzo.



Figura 5.10: Logo di DHC⁹

5.2.6 Robomongo

Robomongo è uno strumento di gestione di MongoDB che permette di utilizzare tutte le funzionalità della *shell* di MongoDB fornendo però un'intuitiva interfaccia grafica. Tra i vantaggi vi sono, oltre alla maggiore leggibilità dei dati contenuti nel *database*, la possibilità di avere più finestre con i risultati visibili contemporaneamente, di poter visualizzare nella stessa *shell* i risultati di due *query* differenti e un aiuto nella scrittura delle interrogazioni, dovuta alla funzione di auto-completamento.



Figura 5.11: Logo di Robomongo¹⁰

⁹Fonente: <https://restlet.com/products/dhc/>

¹⁰Fonente: <https://robomongo.org/>

5.2.7 APIDoc e JSDoc

Per documentare il codice JavaScript e le *Application Programming Interface (API)*_G scritte sono stati usati, rispettivamente, JSDoc e APIDoc, che consentono di documentare *inline* il codice e di generare, a partire dalla documentazione, dei piccoli siti web che consentono una facile e veloce consultazione.

5.3 Linguaggi

L'unico linguaggio utilizzato è JavaScript: la scelta è stata imposta dal fatto che l'applicazione viene eseguita su un server Node.js. Di seguito verranno descritte le principali caratteristiche di questo linguaggio.

5.3.1 JavaScript

JavaScript (logo in Figura 5.12) è un linguaggio di programmazione importante perché è il linguaggio più usato nei *browser* web, ma al contempo è uno dei più "disprezzati". Presenta notevoli differenze rispetto a tutti gli altri e molte delle sue caratteristiche possono essere viste come un bene o un male.

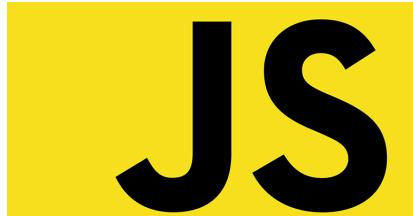


Figura 5.12: Logo di JavaScript¹¹

Una di queste è la tipizzazione debole. In un linguaggio fortemente tipizzato il compilatore è in grado di riconoscere gli errori di tipo e di segnalarli al programmatore, impedendo che questi si tramutino in errori di esecuzione difficili da trovare. D'altro canto, però, nel mondo delle applicazioni web una tipizzazione forte costringerebbe ad usare complicate gerarchie di classi che in JavaScript possono essere evitate. Il lato negativo è che il programmatore deve essere molto più attento e sapere molto meglio quello che sta facendo. Questo si traduce in un maggior numero di *test*.

Un punto forte è la facilità con cui possono essere creati gli oggetti: basta semplicemente elencare tutte le componenti (o proprietà) di un oggetto per crearlo. In JavaScript anche le funzioni sono oggetti, quindi i metodi si definiscono esattamente come gli attributi. Tuttavia, come sarà descritto in seguito, questa flessibilità porta alla mancanza di encapsulazione.

Un altro punto controverso è l'ereditarietà prototipale. In JavaScript gli oggetti possono ereditare liberamente proprietà da altri oggetti: questo meccanismo è molto potente, ma è difficile padroneggiarlo, soprattutto per chi è abituato all'ereditarietà "classica" dei linguaggi fortemente tipizzati. Il problema si riflette soprattutto nell'applicazione dei *design pattern* per la progettazione ad oggetti: è impossibile applicarli così come sono, e imparare ad applicarli può risultare frustrante.

¹¹Fonte: <https://code.support/category/code/javascript/>

L'enorme diffusione di JavaScript è dovuta principalmente al fiorire di numerose librerie nate allo scopo di semplificare la programmazione sul *browser*, ma anche alla nascita di *framework_G* lato server e nel mondo *mobile* che lo supportano come linguaggio principale. Node.js, infatti, si basa su JavaScript. Di conseguenza ogni applicazione eseguita su un server Node.js deve essere scritta in JavaScript.

Capitolo 6

Analisi dei requisiti

6.1 Attori coinvolti

L'attore principale dell'applicazione Catalogue Manager è l'Utente Autenticato. Una gerarchia di utenti, ognuno associato a permessi e funzionalità crescenti, è prevista per le successive versioni del gestore del catalogo applicativo di Monokee, ma non è stata inserita tra i requisiti del progetto per volontà del tutor aziendale.

L'autenticazione è di tipo federato e fa uso di $SAML_G$: non è pertanto presente una fase di registrazione. L'applicazione ha il ruolo di SP_G e utilizza Monokee come IdP_G . Un utente non autenticato (ma autenticato presso Monokee) dipendentemente dal suo ruolo può:

- aggiungere manualmente l'applicazione ad un dominio;
- accedere all'applicazione.

Tutte le funzionalità di Catalogue Manager sono accessibili solamente dopo aver effettuato l'accesso e, come già detto, non esiste nessuna distinzione tra utenti: questo significa che un amministratore di dominio e un semplice utente di Monokee in Catalogue Manager hanno gli stessi poteri.

6.2 Casi d'uso ad alto livello

Considerando l'elevato numero di funzionalità previste in Catalogue Manager di seguito vengono riportati solo i casi d'uso di alto livello, in modo da poter dare una visione d'insieme più precisa del prodotto sviluppato.

Ad ogni caso d'uso è associato un identificatore univoco così formato:

UCTX.Y.Z

dove:

- **T** corrisponde al tipo di caso d'uso:
 - **U** per l'Utente Non Autenticato;
 - **A** per l'Utente Autenticato;
- **X** corrisponde all'identificatore del caso d'uso "padre";

- **Y** corrisponde all'identificatore del caso d'uso "figlio" di primo livello;
- **Z** corrisponde all'identificatore del caso d'uso "figlio" di secondo livello;

Man mano che si scende nella gerarchia (da padre a figlio di primo e poi secondo livello) si aumenta il dettaglio del caso d'uso.

6.2.1 Operazioni permesse ad un Utente Non Autenticato

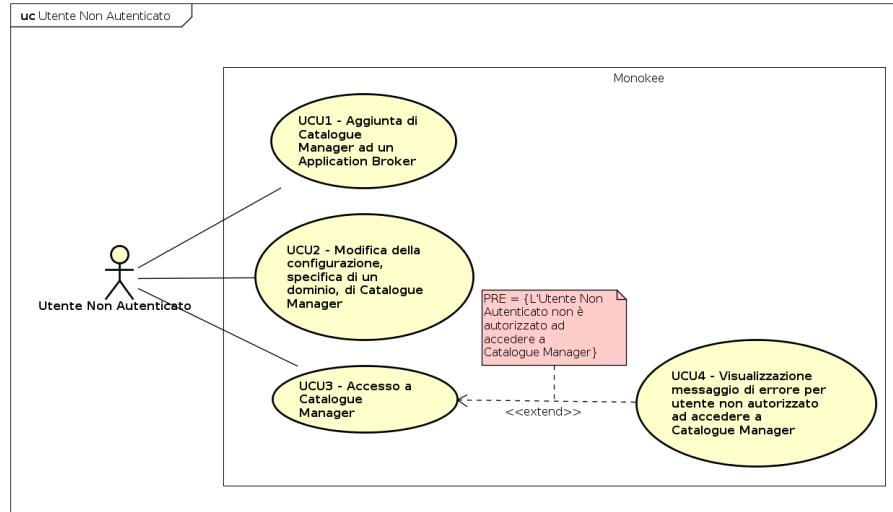


Figura 6.1: Operazioni di alto livello permesse ad un Utente Non Autenticato

Operazioni generali per l'Utente Non Autenticato	
Attori Primari	Utente Non Autenticato
Scopo e Descrizione	L'Utente Non Autenticato può aggiungere, tramite le funzionalità di Monokee, l'applicazione Catalogue Manager ad un <i>application broker</i> di un dominio. Dopo averla aggiunta può decidere in qualsiasi momento di cambiarne gli attributi di configurazione attraverso la pagina di modifica di Monokee. Infine può accedere all'applicazione per gestire il catalogo applicativo. Si ricorda che, sebbene venga definito come Utente Non Autenticato, l'utente in questione deve essere autenticato ed autorizzato tramite Monokee.
Precondizioni	L'Utente Non Autenticato ha effettuato l'accesso a Monokee.
Postcondizioni	Monokee ha preso in carico, ed eseguito, l'operazione voluta dall'Utente Non Autenticato.

Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Non Autenticato aggiunge l'applicazione Catalogue Manager ad un <i>application broker</i> di un dominio di Monokee. (UCU1) 2. L'Utente Non Autenticato modifica gli attributi di Catalogue Manager. (UCU2) 3. L'Utente Non Autenticato accede a Catalogue Manager. (UCU3)
Estensioni	<p>L'Utente Non Autenticato visualizza un messaggio di errore perché non è autorizzato ad accedere a Catalogue Manager. (UCU4)</p>

6.2.1.1 UCU1 - Aggiunta di Catalogue Manager ad un Application Broker

UCU1 - Aggiunta di Catalogue Manager ad un Application Broker	
Attori Primari	Utente Non Autenticato
Scopo e Descrizione	L'Utente Non Autenticato può aggiungere l'applicazione Catalogue Manager ad un <i>application broker</i> di un dominio di Monokee. Dato che l' <i>autenticazione_G</i> avviene tramite $SAML_G$, per aggiungere l'applicazione è necessario inserire tutti i parametri richiesti da questo standard.
Precondizioni	L'Utente Non Autenticato ha effettuato l'accesso a Monokee e si trova nella pagina dell'aggiunta di una nuova applicazione $SAML_G$.
Postcondizioni	Monokee ha aggiunto all' <i>application broker</i> del dominio selezionato dall'Utente Non Autenticato l'applicazione Catalogue Manager.

Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Non Autenticato inserisce l'URL_G dell'<i>assertion consumer service</i>. 2. L'Utente Non Autenticato inserisce l'URI_G del SP_G. 3. L'Utente Non Autenticato inserisce il certificato del SP_G. 4. L'Utente Non Autenticato inserisce l'URL_G della pagina successiva al <i>login</i>. 5. L'Utente Non Autenticato inserisce le regole dell'asserzione $SAML_G$. 6. L'Utente Non Autenticato inserisce l'algoritmo di firma. 7. L'Utente Non Autenticato inserisce l'URI_G della pagina di <i>log out</i>. 8. L'Utente Non Autenticato inserisce l'URI_G della pagina di risposta al <i>log out</i>.
--------------------------	--

6.2.1.2 UCU2 - Modifica della configurazione specifica di un dominio di Catalogue Manager

UCU2 - Modifica della configurazione specifica di un dominio di Catalogue Manager	
Attori Primari	Utente Non Autenticato
Scopo e Descrizione	L'Utente Non Autenticato può modificare gli attributi di configurazione di Catalogue Manager per un dominio tramite la pagina di modifica di un'applicazione prevista da Monokee.
Precondizioni	L'Utente Non Autenticato ha effettuato l'accesso a Monokee e si trova nella pagina di modifica di un'applicazione esistente.
Postcondizioni	Monokee ha modificato l'applicazione Catalogue Manager per il dominio selezionato dall'Utente Non Autenticato.

Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Non Autenticato modifica l'URL_G dell'<i>assertion consumer service</i>. 2. L'Utente Non Autenticato modifica l'URI_G del SP_G. 3. L'Utente Non Autenticato modifica il certificato del SP_G. 4. L'Utente Non Autenticato modifica l'URL_G della pagina successiva al <i>login</i>. 5. L'Utente Non Autenticato modifica le regole dell'asserzione $SAML_G$. 6. L'Utente Non Autenticato modifica l'algoritmo di firma. 7. L'Utente Non Autenticato modifica l'URI_G della pagina di <i>log out</i>. 8. L'Utente Non Autenticato modifica l'URI_G della pagina di risposta al <i>log out</i>.
--------------------------	--

6.2.1.3 UCU3 - Accesso a Catalogue Manager

UCU3 - Accesso a Catalogue Manager	
Attori Primari	Utente Non Autenticato
Scopo e Descrizione	L'Utente Non Autenticato può accedere a Catalogue Manager tramite l' <i>application broker</i> di un dominio di Monokee.
Precondizioni	L'Utente Non Autenticato ha effettuato l'accesso a Monokee e si trova nell' <i>application broker</i> di un dominio.
Postcondizioni	L'Utente Non Autenticato ha effettuato l'accesso a Catalogue Manager e si trova nella pagina principale della nuova applicazione.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Non Autenticato richiede (tramite un <i>click</i>) l'accesso a Catalogue Manager.
Estensioni	L'Utente Non Autenticato visualizza un messaggio di errore perché l' IdP_G di Monokee non gli ha consentito l'accesso a Catalogue Manager.

6.2.1.4 UCU4 - Visualizzazione messaggio di errore per utente non autorizzato

UCU4 - Visualizzazione messaggio di errore per utente non autorizzato	
Attori Primari	Utente Non Autenticato
Scopo e Descrizione	L'Utente Non Autenticato visualizza un messaggio di errore in seguito all'accesso negato, da parte dell' IdP_G di Monokee, all'applicazione Catalogue Manager.
Precondizioni	L'Utente Non Autenticato ha effettuato l'accesso a Monokee e si trova nell' <i>application broker</i> di un dominio. ha inoltre richiesto l'accesso all'applicazione Catalogue Manager, ma gli è stata negata dall' IdP_G di Monokee.
Postcondizioni	L'Utente Non Autenticato ha visualizzato il messaggio di errore per autenticazione non riuscita.
Flusso Principale	1. L'Utente Non Autenticato visualizza il messaggio di errore conseguente all'accesso negato a Catalogue Manager.

6.2.2 Operazioni generali per l'Utente Autenticato

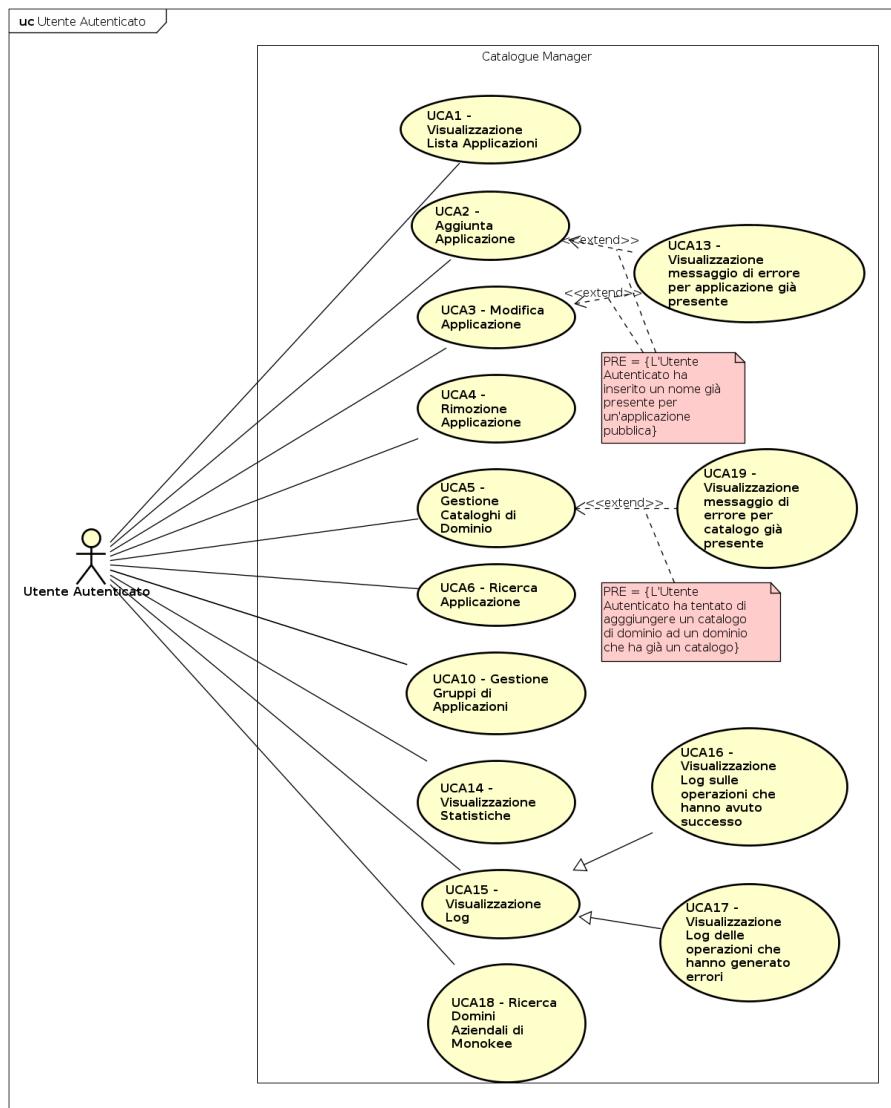


Figura 6.2: Operazioni di alto livello permesse ad un Utente Autenticato

Operazioni generali per l'Utente Autenticato	
Attori Primari	Utente Autenticato
Scopo e Descrizione	All'Utente Autenticato viene mostrata una <i>dashboard</i> a partire dalla quale può eseguire varie operazioni. Queste operazioni permettono di gestire l'intero catalogo applicativo, i raggruppamenti di applicazioni e i cataloghi specifici per i domini aziendali. È possibile anche cercare le applicazioni sulla base di vari filtri, visualizzare statistiche e <i>log</i> .
Precondizioni	L'Utente Autenticato ha effettuato l'accesso a Catalogue Manager a partire da Monokee attraverso $SAML_G$.
Postcondizioni	Catalogue Manager ha preso in carico, ed eseguito, l'operazione voluta dall'Utente Autenticato.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato visualizza la lista di applicazioni presenti nel catalogo. (UCA1) 2. L'Utente Autenticato aggiunge un'applicazione al catalogo. (UCA2) 3. L'Utente Autenticato modifica i dati di un'applicazione del catalogo. (UCA3) 4. L'Utente Autenticato rimuove un'applicazione. (UCA4) 5. L'Utente Autenticato gestisce i cataloghi specifici dei domini aziendali. (UCA5) 6. L'Utente Autenticato ricerca un'applicazione. (UCA6) 7. L'Utente Autenticato gestisce i raggruppamenti di applicazioni. (UCA10) 8. L'Utente Autenticato visualizza le statistiche sul numero di applicazioni e gruppi (UCA14) 9. L'Utente Autenticato visualizza i <i>log</i> sulle operazioni effettuate da lui e dagli altri utenti dell'applicazione. (UCA15) <p>I <i>log</i> riguardano sia le operazioni eseguite con successo (UCA16) sia quelle che hanno generato errori (UCA17).</p>
Estensioni	L'Utente Autenticato cerca di inserire un'applicazione pubblica che è già presente nel catalogo. (UCA13)

6.2.2.1 UCA1 - Visualizzazione Applicazioni

UCA1 - Visualizzazione Applicazioni	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato può visualizzare le applicazioni presenti nel catalogo di Monokee.
Precondizioni	Catalogue Manager presenta all'Utente Autenticato una pagina contenente l'elenco delle applicazioni presenti in Monokee.
Postcondizioni	L'Utente Autenticato ha visualizzato le applicazioni presenti in Monokee.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato visualizza i dettagli di un'applicazione presente in Monokee.

6.2.2.2 UCA2 - Aggiunta Applicazione Pubblica

UCA2 - Aggiunta Applicazione Pubblica	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato può aggiungere una nuova applicazione al catalogo di Monokee.
Precondizioni	Catalogue Manager presenta all'Utente Autenticato una pagina contenente un <i>form</i> per l'aggiunta di un'applicazione al catalogo di Monokee.
Postcondizioni	L'Utente Autenticato ha aggiunto un'applicazione al catalogo di Monokee.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato inserisce il nome dell'applicazione. 2. L'Utente Autenticato inserisce la descrizione dell'applicazione. 3. L'Utente Autenticato inserisce l'URL_G dell'applicazione. 4. L'Utente Autenticato inserisce l'immagine dell'applicazione. 5. L'Utente Autenticato seleziona le categorie di appartenenza dell'applicazione. 6. L'Utente Autenticato seleziona la tipologia di $autenticazione_G$. 7. L'Utente Autenticato seleziona il nome del gruppo di applicazioni nel quale inserire l'applicazione.

Estensioni	<ol style="list-style-type: none"> 1. L’Utente Autenticato visualizza un messaggio di errore come conseguenza al tentativo di inserimento dello stesso nome di un’altra applicazione pubblica di Monokee.
-------------------	--

6.2.2.3 UCA3 - Modifica Applicazione Pubblica

UCA3 - Modifica Applicazione Pubblica	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L’Utente Autenticato può modificare un’applicazione esistente nel catalogo di Monokee.
Precondizioni	Catalogue Manager presenta all’Utente Autenticato una pagina contenente un <i>form</i> per la modifica di un’applicazione nel catalogo di Monokee.
Postcondizioni	L’Utente Autenticato ha modificato un’applicazione nel catalogo Monokee.
Flusso Principale	<ol style="list-style-type: none"> 1. L’Utente Autenticato modifica il nome dell’applicazione. 2. L’Utente Autenticato modifica la descrizione dell’applicazione. 3. L’Utente Autenticato modifica l’URL_G dell’applicazione. 4. L’Utente Autenticato modifica l’immagine dell’applicazione. 5. L’Utente Autenticato seleziona le categorie di appartenenza dell’applicazione. 6. L’Utente Autenticato modifica i dati necessari all’$autenticazione_G$. Tali dati possono riguardare l’accesso <i>form-based</i>, tramite $SAML_G$ o di terzo tipo. 7. L’Utente Autenticato seleziona il nome del gruppo di applicazioni nel quale modificare l’applicazione. 8. L’Utente Autenticato decide se pubblicare l’applicazione. 9. L’Utente Autenticato decide se mettere l’applicazione in manutenzione.
Estensioni	<ol style="list-style-type: none"> 1. L’Utente Autenticato visualizza un messaggio di errore come conseguenza al tentativo di inserimento dello stesso nome di un’altra applicazione pubblica di Monokee.

6.2.2.4 UCA4 - Rimozione Applicazione Pubblica

UC4 - Rimozione Applicazione Pubblica	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato può rimuovere un'applicazione pubblica dal catalogo di Monokee.
Precondizioni	L'applicazione selezionata è presente nel catalogo di Monokee e l'Utente Autenticato ha selezionato il comando di rimozione su di essa.
Postcondizioni	L'applicazione selezionata è stata rimossa dal catalogo di Monokee.
Flusso Principale	1. L'Utente Autenticato conferma l'operazione di rimozione.
Inclusioni	Richiesta di conferma rimozione.

6.2.2.5 UCA5 - Gestione Cataloghi di Dominio

UCA5 - Gestione Cataloghi di Dominio	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato può gestire i cataloghi associati a domini aziendali.
Precondizioni	Catalogue Manager presenta all'Utente Autenticato la pagina per la gestione dei cataloghi associati a domini aziendali.
Postcondizioni	Catalogue Manager ha preso in carico, ed eseguito, le operazioni richieste dall'Utente Autenticato.

Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato può aggiungere un nuovo catalogo di dominio ad un dominio senza catalogo. 2. L'Utente Autenticato può visualizzare i cataloghi di dominio esistenti. 3. L'Utente Autenticato può aggiungere un'applicazione ad un catalogo di dominio esistente. 4. L'Utente Autenticato può rimuovere un'applicazione da un catalogo di dominio esistente. 5. L'Utente Autenticato può rimuovere un catalogo di dominio esistente. 6. L'Utente Autenticato può cercare un catalogo di dominio tra quelli esistenti. La ricerca può avvenire in base al nome del dominio.
Estensioni	<ol style="list-style-type: none"> 1. L'Utente Autenticato visualizza un messaggio di errore dovuto all'esistenza di un altro catalogo per il dominio selezionato.

6.2.2.6 UCA6 - Ricerca Applicazione

UCA6 - Ricerca Applicazione	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato può cercare un'applicazione del catalogo di Monokee. La ricerca può avvenire per nome o per categoria.
Precondizioni	Catalogue Manager ha mostrato all'utente la pagina di ricerca. L'Utente Autenticato può effettuare ricerche di vario tipo.
Postcondizioni	Catalogue Manager ha mostrato i risultati della ricerca.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato seleziona le modalità di ricerca e inserisce le informazioni richieste.

6.2.2.7 UCA10 - Gestione Gruppi di Applicazioni

UCA10 - Gestione Gruppi di Applicazioni	
Attori Primari	Utente Autenticato

Scopo e Descrizione	L'Utente Autenticato può gestire i gruppi di applicazioni del catalogo di Monokee.
Precondizioni	Catalogue Manager mostra all'Utente Autenticato la pagina di gestione dei gruppi di applicazioni.
Postcondizioni	Catalogue Manager ha preso in carico le richieste dell'Utente Autenticato e le ha eseguite.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato può aggiungere un gruppo di applicazioni. 2. L'Utente Autenticato può visualizzare i gruppi di applicazioni presenti. 3. L'Utente Autenticato può gestire un gruppo di applicazioni esistente. In particolare, L'Utente Autenticato può aggiungere o rimuovere un'applicazione dal gruppo selezionato. 4. L'Utente Autenticato può modificare gli attributi di un gruppo di applicazioni esistente. 5. L'Utente Autenticato può rimuovere un gruppo di applicazioni esistente.

6.2.2.8 UCA13 - Visualizzazione di un messaggio di errore per applicazione pubblica già presente

UCA13 - Visualizzazione di un messaggio di errore per applicazione pubblica già presente	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato ha cercato di inserire un nome corrispondente ad un'applicazione pubblica già presente nel catalogo di Monokee. Catalogue Manager presenta all'Utente Autenticato un messaggio di errore.
Precondizioni	L'Utente Autenticato ha inserito (nella pagina di aggiunta o di modifica di un'applicazione) un nome già utilizzato per un'applicazione pubblica.
Postcondizioni	L'Utente Autenticato ha visualizzato il messaggio di errore presentato da Catalogue Manager.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato visualizza il messaggio di errore.

6.2.2.9 UCA14 - Visualizzazione Statistiche

UCA14 - Visualizzazione Statistiche	
Attori Primari	Utente Autenticato
Scopo e Descrizione	<p>L'Utente Autenticato può visualizzare delle statistiche riguardanti l'applicazione Catalogue Manager. In particolare, le statistiche riguardano:</p> <ol style="list-style-type: none"> 1. numero di applicazioni aggiunte e rimosse in intervalli di tempo definiti a priori: ultime 24 ore, ultima settimana, ultimo mese e ultimo anno; 2. numero di accessi in intervalli di tempo definiti a priori: ultime 24 ore e ultima settimana; 3. numero di utenti attivi; 4. numero di applicazioni e gruppi pubblici e privati; 5. numero di applicazioni, pubbliche e private, appartenenti ad ogni categoria (intesa come "sotto categoria"); 6. numero di applicazioni, pubbliche e private, appartenenti ad una specifica "sovra categoria".
Precondizioni	L'Utente Autenticato ha richiesto la visualizzazione delle statistiche dell'applicazione Catalogue Manager.
Postcondizioni	L'Utente Autenticato ha visualizzato le statistiche dell'applicazione Catalogue Manager.

Flusso Principale	<ol style="list-style-type: none"> 1. L’Utente Autenticato visualizza il numero di applicazioni aggiunte e rimosse in intervalli di tempo definiti a priori: ultime 24 ore, ultima settimana, ultimo mese e ultimo anno. 2. L’Utente Autenticato visualizza il numero di accessi in intervalli di tempo definiti a priori: ultime 24 ore e ultima settimana. 3. L’Utente Autenticato visualizza il numero di utenti attivi. 4. L’Utente Autenticato visualizza il numero di applicazioni e gruppi pubblici e privati. 5. L’Utente Autenticato visualizza il numero di applicazioni, pubbliche e private, appartenenti ad ogni categoria (intesa come “sotto categoria”). 6. L’Utente Autenticato visualizza il numero di applicazioni, pubbliche e private, appartenenti ad una specifica “sovra categoria”.
--------------------------	--

6.2.2.10 UCA15 - Visualizzazione Log

UCA15 - Visualizzazione Log	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L’Utente Autenticato può visualizzare i <i>log</i> delle operazioni eseguite nell’applicazione Catalogue Manager. In particolare i <i>log</i> possono riguardare le operazioni eseguite con successo (UCA16) e quelle che hanno generato un errore (UCA17). I <i>log</i> salvati possono essere filtrati per intervallo di date, per <i>keywords</i> e per tipologia.
Precondizioni	L’Utente Autenticato ha richiesto la visualizzazione dei <i>log</i> dell’applicazione Catalogue Manager.
Postcondizioni	L’Utente Autenticato ha visualizzato i <i>log</i> dell’applicazione Catalogue Manager.
Flusso Principale	<ol style="list-style-type: none"> 1. L’Utente Autenticato visualizza i <i>log</i> dell’applicazione.

6.2.2.11 UCA16 - Visualizzazione Log sulle operazioni che hanno avuto successo

UCA16 - Visualizzazione Log sulle operazioni che hanno avuto successo
--

Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato può visualizzare i <i>log</i> delle operazioni eseguite con successo nell'applicazione Catalogue Manager. I <i>log</i> salvati possono essere filtrati per intervallo di date, per <i>keywords</i> e per tipologia.
Precondizioni	L'Utente Autenticato ha richiesto la visualizzazione dei <i>log</i> dell'applicazione Catalogue Manager. Ha successivamente selezionato la visualizzazione dei <i>log</i> delle operazioni eseguite con successo.
Postcondizioni	L'Utente Autenticato ha visualizzato i <i>log</i> delle operazioni eseguite con successo dell'applicazione Catalogue Manager.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato visualizza i <i>log</i> delle operazioni eseguite con successo.

6.2.2.12 UCA17 - Visualizzazione Log delle operazioni che hanno generato errori

UCA17 - Visualizzazione Log delle operazioni che hanno generato errori	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato può visualizzare i <i>log</i> delle operazioni che hanno generato errori nell'applicazione Catalogue Manager. I <i>log</i> salvati possono essere filtrati per intervallo di date, per <i>keywords</i> e per tipologia.
Precondizioni	L'Utente Autenticato ha richiesto la visualizzazione dei <i>log</i> dell'applicazione Catalogue Manager. Ha successivamente selezionato la visualizzazione dei <i>log</i> delle operazioni che hanno generato errori.
Postcondizioni	L'Utente Autenticato ha visualizzato i <i>log</i> delle operazioni che hanno generato errori dell'applicazione Catalogue Manager.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato visualizza i <i>log</i> delle operazioni che hanno generato errore.

6.2.2.13 UCA18 - Ricerca Domini Aziendali di Monokee

UCA18 - Ricerca Domini Aziendali di Monokee	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato può effettuare una ricerca tra i domini aziendali di Monokee. La ricerca avviene per nome.
Precondizioni	Catalogue Manager mostra all'Utente Autenticato la pagina di gestione dei cataloghi di dominio di Monokee.
Postcondizioni	Catalogue Manager ha preso cercato tra i domini aziendali di Monokee e ha mostrato all'Utente Autenticato i risultati.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato cerca un dominio aziendale di Monokee.

6.2.2.14 UCA19 - Visualizzazione di un messaggio di errore per catalogo già presente

UCA19 - Visualizzazione di un messaggio di errore per catalogo già presente	
Attori Primari	Utente Autenticato
Scopo e Descrizione	L'Utente Autenticato ha cercato di aggiungere un catalogo di dominio ad un dominio che ha già un catalogo. Catalogue Manager presenta all'Utente Autenticato un messaggio di errore.
Precondizioni	L'Utente Autenticato ha cercato di aggiungere un catalogo di dominio ad un dominio che ha già un catalogo.
Postcondizioni	L'Utente Autenticato ha visualizzato il messaggio di errore presentato da Catalogue Manager.
Flusso Principale	<ol style="list-style-type: none"> 1. L'Utente Autenticato visualizza il messaggio di errore.

6.3 Requisiti

I requisiti funzionali e di vincolo individuati sono riportati nelle seguenti tabelle. Viene inoltre indicato se si tratta di un requisito fondamentale, desiderabile o facoltativo e

una sua descrizione.

Ogni requisito è identificato da un codice, che segue il seguente formalismo:

RXY Gerarchia

Dove:

- **X** corrisponde alla tipologia del requisito e può assumere i seguenti valori:

1 = Funzionale;

2 = Vincolo.

- **Y** corrisponde alla priorità del requisito e può assumere i seguenti valori:

O = Obbligatorio;

D = Desiderabile;

F = Facoltativo o Opzionale.

- **Gerarchia** identifica la relazione gerarchica che c'è tra i requisiti di uno stesso tipo. Vi è dunque una struttura gerarchica per ogni tipologia di requisito.

6.3.1 Principali requisiti funzionali

Nella Tabella 6.21 vengono elencati i principali requisiti funzionali dell'applicazione Catalogue Manager.

Requisito	Descrizione
R1O 1	Un utente deve poter visualizzare la lista delle applicazioni del catalogo di Monokee.
R1O 1.1	Un utente deve poter visualizzare i dettagli di un'applicazione presente nel catalogo di Monokee.
R1O 1.2	Un utente deve poter visualizzare il nome dell'applicazione del catalogo di Monokee.
R1O 1.3	Un utente deve poter visualizzare la descrizione dell'applicazione del catalogo di Monokee.
R1O 1.4	Un utente deve poter visualizzare l'immagine di un'applicazione del catalogo di Monokee.
R1O 1.5	Un utente deve poter visualizzare la lista delle categorie associate ad un'applicazione del catalogo di Monokee.
R1O 1.6	Un utente deve poter visualizzare il tipo di <i>autenticazione_G</i> di un'applicazione del catalogo di Monokee.
R1O 1.7	Un utente deve poter visualizzare ogni dettaglio di <i>autenticazione_G</i> per un'applicazione del catalogo di Monokee.

R1O 1.8	Un utente deve poter visualizzare la visibilità (pubblica o privata) dell'applicazione.
R1O 1.9	Un utente deve poter visualizzare il nome del gruppo nel quale è inserita l'applicazione.
R1O 1.10	Un utente deve poter visualizzare se l'applicazione è in manutenzione o meno.
R1O 1.10	Un utente deve poter visualizzare se l'applicazione è stata pubblicata o meno.
R1O 2	Un utente deve poter inserire una nuova applicazione.
R1O 2.1	Un utente deve poter inserire il nome della nuova applicazione.
R1O 2.2	Un utente deve poter inserire la descrizione della nuova applicazione.
R1O 2.3	Un utente deve poter inserire l' URL_G della nuova applicazione.
R1O 2.4	Un utente deve poter inserire l'immagine della nuova applicazione.
R1O 2.5	Un utente deve poter selezionare le categorie di appartenenza della nuova applicazione.
R1O 2.6	Un utente deve poter selezionare il tipo di <i>autenticazione_G</i> della nuova applicazione.
R1O 2.7	Un utente deve poter inserire i dati di <i>autenticazione_G</i> della nuova applicazione.
R1O 2.8	Un utente deve poter inserire il nome del gruppo nel quale sarà inserita la nuova applicazione.
R1O 3	Un utente deve poter modificare un'applicazione.
R1O 3.1	Un utente deve poter modificare il nome dell'applicazione.
R1O 3.2	Un utente deve poter modificare la descrizione dell'applicazione.
R1O 3.3	Un utente deve poter modificare l' URL_G dell'applicazione.
R1O 3.4	Un utente deve poter modificare l'immagine dell'applicazione.

R1O 3.5	Un utente deve poter selezionare le categorie di appartenenza dell'applicazione.
R1O 3.6	Un utente deve poter modificare i dati di autenticazione dell'applicazione.
R1O 3.7	Un utente deve poter modificare il nome del gruppo nel quale sarà inserita l'applicazione.
R1O 3.8	Un utente deve poter mettere un'applicazione in manutenzione.
R1O 3.9	Un utente deve poter pubblicare un'applicazione.
R1O 4	Un utente deve poter rimuovere un'applicazione.
R1O 5	Un utente deve poter gestire i cataloghi di dominio di Monokee.
R1O 5.1	Un utente deve poter aggiungere un catalogo di dominio.
R1O 5.2	Un utente deve visualizzare un messaggio di errore se il dominio aziendale è già collegato ad un catalogo di dominio.
R1O 5.3	Un utente deve poter visualizzare i cataloghi di dominio esistenti.
R1O 5.4	Un utente deve poter gestire le applicazioni presenti in un catalogo di dominio.
R1O 5.4.1	Un utente deve poter visualizzare le applicazioni nel catalogo.
R1O 5.4.2	Un utente deve poter aggiungere un'applicazione al catalogo.
R1O 5.4.3	Un utente deve poter rimuovere un'applicazione dal catalogo.
R1O 5.5	L'utente deve poter rimuovere un catalogo di dominio esistente.
R1O 5.6	La rimozione di un catalogo di dominio deve comportare la rimozione di tutte le applicazioni collegate a quel catalogo.
R1O 6	Un utente deve poter cercare un'applicazione nel catalogo.
R1O 10	Un utente deve poter gestire i gruppi di applicazioni.

R1O 10.1	Un utente deve poter aggiungere un gruppo.
R1O 10.1.1	Un utente deve poter inserire il nome del gruppo.
R1O 10.1.2	Un utente deve poter inserire la descrizione del gruppo.
R1O 10.1.3	Un utente deve poter inserire l'immagine del gruppo.
R1O 10.2	Un utente deve poter visualizzare i gruppi di applicazioni esistenti.
R1O 10.3	Un utente deve poter gestire un gruppo di applicazioni esistente.
R1O 10.3.1	Un utente deve poter aggiungere un'applicazione al gruppo.
R1O 10.3.2	Un utente deve poter rimuovere un'applicazione dal gruppo.
R1O 10.3.3	Un utente deve visualizzare un messaggio di errore se l'applicazione selezionata è già presente nel gruppo.
R1O 10.4	Un utente deve poter modificare un gruppo.
R1O 10.4.1	Un utente deve poter inserire il nuovo nome del gruppo.
R1O 10.4.2	Un utente deve poter inserire la nuova descrizione del gruppo.
R1O 10.4.3	Un utente deve poter inserire la nuova immagine del gruppo.
R1O 10.5	Un utente deve poter rimuovere un gruppo.
R1O 13	Un utente deve visualizzare un messaggio di errore se l'applicazione che si sta cercando di inserire è già presente.
R1D 14	Un utente deve poter visualizzare le statistiche dell'applicazione Catalogue Manager.
R1D 14.1	Un utente deve poter visualizzare il numero di applicazioni aggiunte e rimosse in intervalli di tempo definiti a priori: ultime 24 ore, ultima settimana, ultimo mese e ultimo anno.
R1D 14.2	Un utente deve poter visualizzare il numero di accessi in intervalli di tempo definiti a priori: ultime 24 ore e ultima settimana.

R1D 14.3	Un utente deve poter visualizzare il numero di utenti attivi.
R1D 14.4	Un utente deve poter visualizzare il numero di applicazioni e gruppi pubblici e privati.
R1D 14.5	Un utente deve poter visualizzare il numero di applicazioni, pubbliche e private, appartenenti ad ogni categoria (intesa come "sotto categoria").
R1D 14.6	Un utente deve poter visualizzare il numero di applicazioni, pubbliche e private, appartenenti ad una specifica "sovra categoria".
R1D 15	Un utente deve poter visualizzare i <i>log</i> dell'applicazione Catalogue Manager.
R1D 16	Un utente deve poter visualizzare i <i>log</i> delle operazioni eseguite con successo.
R1D 17	Un utente deve poter visualizzare i <i>log</i> delle operazioni che hanno generato errori.
R1O 18	Un utente deve poter effettuare una ricerca tra i domini aziendali di Monokee.
R1O 19	Un utente deve visualizzare un messaggio di errore se il dominio ha già un catalogo associato.

Tabella 6.21: Principali requisiti funzionali

6.3.2 Requisiti di vincolo

Nella Tabella 6.22 vengono elencati i requisiti di vincolo dell'applicazione.

Requisito	Descrizione
R2O 1	L'applicazione deve utilizzare Node.js e JavaScript.
R2O 2	L'applicazione deve memorizzare i dati su MongoDB.
R2O 3	L'accesso all'applicazione deve avvenire con $SAML_G$.

Tabella 6.22: Requisiti di vincolo

Capitolo 7

Progettazione

7.1 Interfaccia REST-like

Il back end si basa su uno stile $REST_G$ -like, ovvero con le seguenti caratteristiche:

- stato dell'applicazione e funzionalità divisi in risorse web;
- ogni risorsa è unica e indirizzabile attraverso un URI_G ;
- tutte le risorse sono condivise come interfaccia uniforme per il trasferimento di stato tra client e risorse. Questo trasferimento consiste in:
 - un insieme vincolato di operazioni ben definite;
 - un insieme vincolato di contenuti, optionalmente supportato da codice a richiesta;
 - un protocollo:
 - * client-server;
 - * privo di stato;
 - * memorizzabile in cache;
 - * a livelli.

$REST_G$ utilizza il concetto di risorsa (aggregato di dati con un nome e una rappresentazione interna), sulla quale è possibile invocare operazioni *Create*, *Read*, *Update*, *Delete* ($CRUD_G$) secondo la corrispondenza indicata in Tabella 7.1.

Metodo HTTP	Operazione CRUD	Descrizione
GET	Read	Ricava e ritorna informazioni su una risorsa
PUT	Update	Aggiorna una risorsa
POST	Create	Crea una risorsa
DELETE	Delete	Cancella una risorsa

Tabella 7.1: Corrispondenza tra CRUD e HTTP

Per la rappresentazione dei dati si è scelto di utilizzare *JavaScript Object Notation (JSON)*_G perché si integra molto bene con le tecnologie utilizzate e con il linguaggio JavaScript. Questo non è vero per *XML*_G o *Comma Separated Values (CSV)*_G, che richiederebbero librerie specifiche. Inoltre *JSON*_G è molto meno verboso e molto più flessibile di *XML*_G, e si adatta molto bene al dominio dell'applicazione.

Uno stile architettonico di questo tipo permette l'indipendenza completa tra back end e front end, permettendo così espansioni su altre piattaforme senza dover modificare il back end dell'applicazione.

7.2 Architettura

In Figura 7.1 è rappresentata l'architettura di Catalogue Manager. Il diagramma dei *package* rappresenta i componenti ad un livello di dettaglio molto basso, ma sufficiente a capire le relazioni principali. Come si nota, Catalogue Manager utilizza i modelli di *mongoose.js* (*package Monokee.models*) di Monokee. Questa scelta è stata imposta dall'architettura esistente: alcuni servizi di Monokee utilizzavano, e utilizzano, alcuni modelli riguardanti il catalogo. Uno spostamento completo avrebbe causato numerosi problemi e cambiamenti. È stato pertanto deciso di importare solo e soltanto i modelli necessari allo svolgimento delle operazioni di Catalogue Manager. Successivamente verranno descritti i modelli importati.

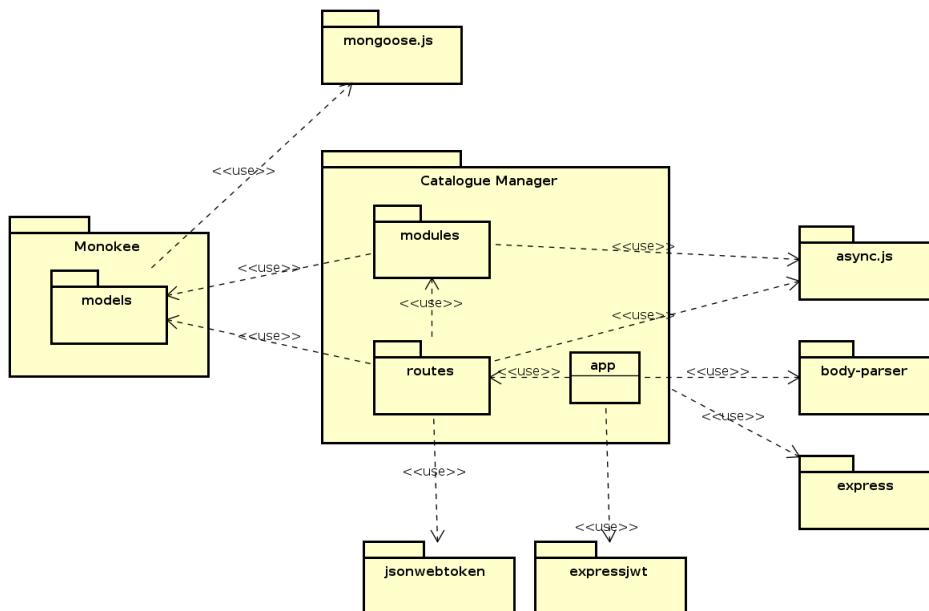


Figura 7.1: Architettura generale

7.2.1 Moduli esterni

Catalogue Manager fa uso di numerosi moduli esterni: nel diagramma in Figura 7.1 sono stati riportati solamente quelli principali.

mongoose.js

mongoose.js è uno strumento di modellazione ad oggetti per MongoDB progettato per lavorare in ambiente asincrono (e quindi ottimo per Node.js) che offre grande supporto per le interrogazioni al *database*. La modellazione ad oggetti consente di progettare con precisione le collezioni attraverso la definizione degli attributi, dei loro tipi e delle loro relazioni: in questo modo è possibile definire uno schema sul quale basarsi.

La definizione dei tipi, in particolare, consente di controllare la consistenza dei dati inseriti. Grazie alla definizione di *middlewares*¹, inoltre, è possibile eseguire delle operazioni prima, o dopo, le interrogazioni al *database*, evitando la replicazione di codice e aderendo al principio *Don't Repeat Yourself (DRY)*_G. Esistono due tipi di *middleware*: **document**, che agiscono a livello dell'intero *document* MongoDB e che possono essere definiti su operazioni come salvataggio, rimozione o validazione, e **query**, che agiscono durante le interrogazioni al *database*, in particolare per il **find** (ricerca), l'**update** (aggiornamento) e il **count** (conteggio). La possibilità di eseguire codice personale prima dello svolgimento di queste operazioni consente, ad esempio, di effettuare controlli specifici sui dati e di modificarli se necessario. È uno strumento molto potente del quale si è fatto grande uso.

Un *middleware* meno conosciuto, ma importantissimo per la gestione degli errori in Catalogue Manager, è quello che viene eseguito dopo la sollevazione di un errore (**post error**). Durante la sua esecuzione si dispone dell'errore sollevato, ed è possibile aggiungerci delle informazioni sfruttando il permissivo paradigma ad oggetti di JavaScript. Questi dati aggiuntivi consentono di salvare dei *log* precisi ed accurati.

Dipendenze Come si nota dal diagramma, mongoose.js è utilizzato dal *package models* di Monokee.

async.js

async.js è un modulo di utilità che fornisce potenti funzioni per lavorare in modo asincrono con JavaScript. È stato progettato per un uso con Node.js, ma è utilizzabile anche direttamente nel *browser*. Tra le circa 70 funzioni presenti si trovano molte utili per operare con gli *array* (come **map**, **reduce**, **filter**, eccetera) e altre che implementano *pattern* comuni per un flusso di controllo asincrono. Tutte queste seguono le convenzioni di Node.js e prevedono una sola funzione di **callback** (con due parametri: l'errore sollevato, se presente, e i risultati dell'intera esecuzione o fino al verificarsi dell'errore) che va chiamata una sola volta.

Dipendenze async.js è utilizzato da tutto Catalogue Manager, sia dal *package routes* sia da *modules*.

body-parser

body-parser effettua il *parsing* del *body* delle richieste alle *API*_G e lo rende disponibile nella proprietà **body** dell'oggetto **req** (*Request*) di Express.js. Mette a disposizione numerose funzioni che definiscono come deve essere effettuato il *parsing*: Catalogue Manager utilizza la funzione **json** in modo da analizzare solamente *body* di tipo *JSON*_G.

¹I *middlewares* sono delle funzioni alle quali è passato il controllo durante l'esecuzione di funzioni asincrone. Sono specifici a livello di schema.

Dipendenze body-parser è utilizzato dallo *script* JavaScript utilizzato per avviare l'applicazione (`app.js`).

Express.js

Come abbondantemente descritto in 5.1.2, **Express.js** è utilizzato per definire gli *endpoints* del back end di Catalogue Manager.

Dipendenze Ogni elemento del *package routes* dipende da Express.js, oltre allo script principale per l'avvio dell'applicazione, che lo utilizza per definire gli URI_G dei servizi esposti.

expressjwt e jsonwebtoken

Questi due moduli permettono di utilizzare i **JSON Web Token (JWT)**_G. **jsonwebtoken** è un'implementazione che rispetta il documento RFC7519 ([8]) ed è utilizzato per generare e firmare *token* JWT_G . **expressjwt**, invece, è utilizzato per validare i JWT_G e per inserire il contenuto del *token* nella proprietà **user** dell'oggetto **req** di Express.js. Quest'ultimo modulo consente di autenticare richieste $HTTP_G$ utilizzando *token* JWT_G in applicazioni Node.js.

Dipendenze jsonwebtoken è utilizzato da un unico servizio, `/acs`, ovvero quello che, dopo aver ricevuto la *SAMLResponse* dall' IdP_G , genera il *token* e lo invia al front end di Catalogue Manager.

expressjwt, invece, è utilizzato nello *script* per l'avvio dell'applicazione e "protegge" i servizi per i quali è richiesta l'*autenticazione*_G.

7.2.2 Models di Monokee

Il *package models* di Monokee contiene i modelli di mongoose.js utilizzati da Catalogue Manager, che verranno descritti in dettaglio successivamente (vedi 7.4.1). In Figura 7.2 sono riportati quelli di maggior interesse.

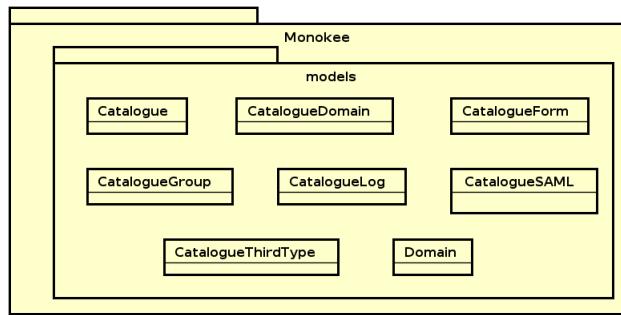


Figura 7.2: Package models

Come si può notare dai nomi, la quasi totalità di questi riguarda esclusivamente il catalogo. Il modello principale è **Catalogue**, che specifica lo schema delle applicazioni da catalogo, non importa se quello di Monokee o uno di dominio. Le applicazioni appartenenti ad uno stesso dominio sono raggruppate nei *document* di **CatalogueDomain**: ad ogni *document* corrisponde un catalogo di dominio.

`CatalogueForm`, `CatalogueSAML` e `CatalogueThirdType` definiscono le informazioni specifiche per i tre tipi di accesso diversi.

`CatalogueGroup` quelle dei gruppi di applicazioni del catalogo, mentre `CatalogueLog` specifica lo schema dei *log*.

`Domain`, invece, è utilizzato solamente per mantenere la consistenza dei dati e per recuperare il giusto catalogo da `CatalogueDomain`: definisce le informazioni dei domini di Monokee.

7.2.3 Modules di Catalogue Manager

Il *package modules* di Catalogue Manager (Figura 7.3) contiene i moduli di utilità usati dalle *routes*, che verranno descritti in dettaglio successivamente (vedi 7.4.2). Questi moduli permettono di raggruppare le operazioni comuni, evitando la duplicazione di codice e rendendo, di conseguenza, il prodotto più manutenibile. Ogni modulo è fortemente coeso, e dipende in misura quasi completamente nulla dagli altri moduli. L'unica eccezione è rappresentata dalla dipendenza nei confronti dei due moduli di *logging*.

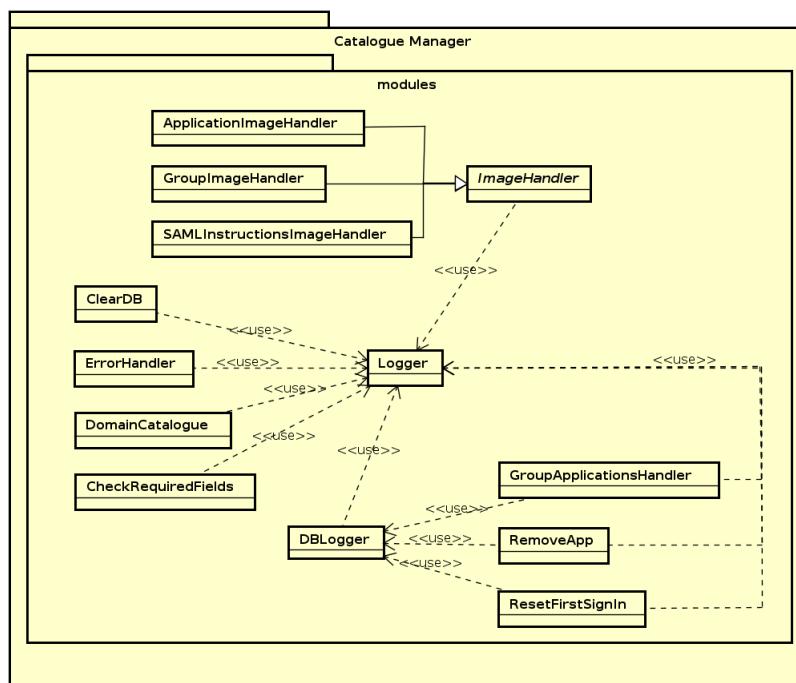


Figura 7.3: Package modules

`Logger` e `DBLogger` si occupano di salvare i *log* rispettivamente su *file* e nel *database* (attraverso il modello `CatalogueLog`).

La gerarchia di `ImageHandler` implementa il *design pattern Template Method* per il salvataggio e la rimozione delle immagini dei gruppi (`GroupImageHandler`), delle applicazioni (`ApplicationImageHandler`) e delle istruzioni per la configurazione di applicazioni di tipo *SAML_G* (`SAMLInstructionsImageHandler`).

ClearDB è utilizzato per eliminare i residui della *soft deletion_G* dal *database* di Monokee. Per consentire il *rollback_G* dei dati, infatti, i *document* non sono direttamente eliminati dal *database*, ma viene impostato a *true* un *flag* (*removed*). Alla fine del processo di cancellazione, se non sono stati rilevati errori, **ClearDB** si occupa di eliminare tutto ciò che ha *removed* a *true*.

CheckRequiredFields controlla semplicemente che tutti i campi obbligatori per il servizio che lo invoca siano presenti nella proprietà **body** dell'oggetto **req** di Express.js.

DomainCatalogue popola il catalogo di un dominio specifico.

ErrorHandler è utilizzato per gestire gli errori riscontrati: oltre a salvare il *log* invia risposte diverse al client in base a quanto è successo. È inoltre in grado di intercettare gli errori sollevati direttamente da mongoose.js (ad esempio, una validazione fallita) e di effettuare un *parsing* per presentare al client un *JSON_G* conforme alla definizione definita durante la progettazione.

GroupApplicationsHandler raggruppa le operazioni che coinvolgono gruppi di applicazioni, come l'aggiunta e la rimozione di applicazioni e il controllo sull'associazione gruppo/applicazione.

RemoveApp si occupa di rimuovere (tramite *soft deletion_G*) un'applicazione dal *database* di Monokee, ed è usato anche come *rollback_G* se si verificano errori durante una creazione.

ResetFirstSignIn, infine, reimposta i *flag* di *first sign in* in seguito alla modifica delle informazioni sull'autenticazione_G *form-based*.

7.2.4 Routes di Catalogue Manager

Il *package routes* (Figura 7.4) di Catalogue Manager contiene gli *endpoints* esposti dal server. In generale, ogni *route* corrisponde a (e soddisfa un) requisito funzionale di alto livello. I servizi *REST_G* definiti sono circa 40, e un diagramma delle classi che li mostri tutti risulterebbe illeggibile e inutile. Una descrizione di ciascuna *route* viene fornita in 8.2.

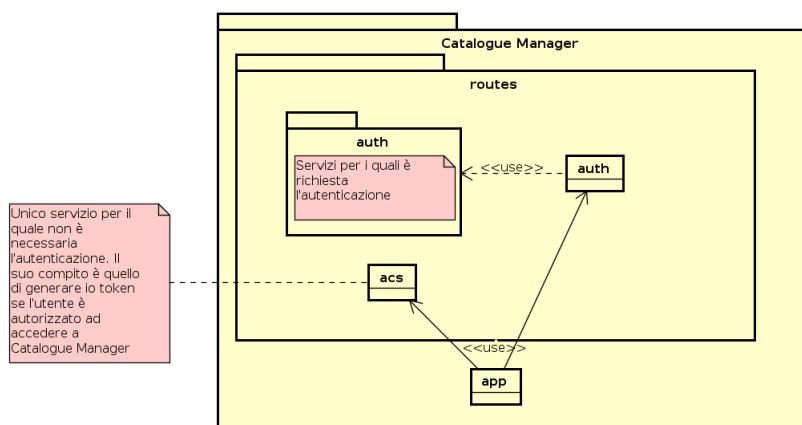


Figura 7.4: Package routes

7.3 Modalità di autenticazione

Come già visto, l' $autenticazione_G$ a Catalogue Manager deve avvenire tramite $SAML_G$. Tuttavia non esiste un modo univoco di effettuare questa $autenticazione_G$, in quanto $SAML_G$ prevede due diverse modalità di accesso, ognuna corrispondente ad uno specifico caso d'uso: in 7.3.2 verranno analizzate entrambe.

Risolto il problema dell'accesso, è importante stabilire come verrà mantenuta la sessione con l'utente autenticato, ovvero se questa sarà memorizzata sul server o meno. La scelta è stata quella di lasciare al client l'onere di far sapere se l' $autenticazione_G$ è avvenuta o no, attraverso l'uso dei JWT_G , descritti in 7.3.1.

7.3.1 JSON Web Token

Descrizione

JWT_G (logo in Figura 7.5) è uno standard *open* ([8]) che definisce un modo **compatto** e **self-contained** per trasmettere informazioni in modo sicuro sotto forma di oggetti $JSON_G$. Le informazioni possono essere verificate dato che sono firmate: la firma può avvenire attraverso una stringa (il cosiddetto **secret**), con l'algoritmo *keyed-Hash Message Authentication Code (HMAC)*_G, o usando una coppia di chiavi pubbliche e private, grazie a RSA_G .

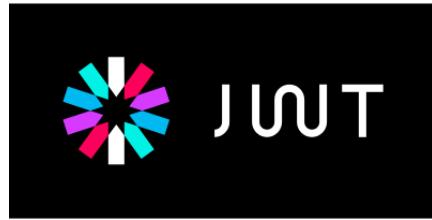


Figura 7.5: Logo di JWT²

JWT_G è **compatto** perché grazie alla sua dimensione ridotta può essere inviato attraverso un URL_G , come parametro di una richiesta POST o dentro un *header* $HTTP_G$. Inoltre, occupando poco spazio, può essere trasmesso velocemente.

JWT_G è **self-contained** perché il **payload** contiene tutte le informazioni riguardo l'utente, evitando di dover interrogare il *database* più di una volta.

I due scenari di utilizzo più comuni sono i seguenti:

- **autenticazione_G**: è il caso d'uso più comune. Dopo la prima $autenticazione_G$ ogni successiva richiesta conterrà il *token* e permetterà all'utente di accedere a tutto ciò che il suo ruolo gli consente (servizi, risorse, eccetera). JWT_G è molto usato con il SSO_G grazie alla sua interoperabilità e alla facilità di utilizzo;
- **scambio di informazioni**: JWT_G è un ottimo modo per trasmettere informazioni in modo sicuro tra parti diverse, grazie alla possibilità di firmarli. Visto che la firma è calcolata sul contenuto, si può anche controllare che le informazioni non siano state alterate.

²Immagine tratta da [25]

Struttura

In Figura 7.6 è mostrato un esempio di *token* utilizzato in Catalogue Manager. Come si può notare è composto da tre parti separate da ':':

- **Header;**
- **Payload;**
- **Signature.**

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
eyJpZCI6IjU3ODM5NjIYmY5NDczNDIiYjM0ZjdiNy  
IsImVtYWlsjoidGF0aWFuYS5jYXjb25pQHlvcG1h  
aWwuY29tliwiaWF0IjoxNDY4NDEzNTc2LCJleHaiO  
E0NzAyMTM1NzZ9.  
_gG9qFghimbheL95aUU5REaOK_1SuchhpSX_Km29948
```

Figura 7.6: Esempio di token di Catalogue Manager

L'**header** tipicamente consiste di due parti: il tipo del *token* (**typ**, che deve essere **JWT**) e l'algoritmo di firma utilizzato (**alg**), che può essere $HMAC_G$ con **Secure Hash Algorithm (SHA)**_G-256 o RSA_G . Nel Frammento di codice 7.1 è mostrato un esempio di *header*.

Frammento di codice 7.1: Esempio di header JWT

```
{
  typ: "JWT",
  alg: "RS256" // RSA
}
```

Il tutto è poi codificato con $Base64_G$ e forma la prima parte del *token*.

La seconda parte è il **payload** e contiene le "affermazioni" (o **claims**), ovvero delle asserzioni di sicurezza riguardo un'entità (tipicamente l'utente) e dati aggiuntivi. Ci sono tre tipi di affermazioni:

- **riservate:** insieme di asserzioni predefinite, non obbligatorie, ma raccomandate per fornire un *token* utile. In particolare sono:
 - **iss:** indica l'**issuer** (emittente) del *token*, e il suo utilizzo generalmente è fortemente dipendente dall'applicazione;
 - **sub:** indice il **subject** (soggetto) del *token* e deve essere unico globalmente o per singolo **issuer**. Come per il campo **iss**, anche il suo utilizzo è fortemente dipendente dall'applicazione;
 - **aud:** indica l'**audience** (pubblico) del *token*. Chiunque voglia utilizzare il *token* deve identificarsi in uno dei casi descritti in questo campo. Se questo non accade, il JWT_G deve essere scartato. Solitamente è un *array* di stringhe che identificano i diversi casi d'uso;
 - **exp:** indica l'**expiration time** (tempo di scadenza) oltre il quale il *token* non deve essere accettato;
 - **nbf:** indica il **not before time** (non prima di) prima del quale il *token* non deve essere accettato;

- **iat**: indica il momento in cui il *token* è stato emesso (**issued at**) e può essere usato per determinare l'età del JWT_G ;
- **jti**: fornisce un identificatore univoco (JWT_G ID) per il *token*. L'ID deve essere assegnato in modo tale da garantire l'unicità; se l'applicazione usa più di un **issuer**, le collisioni devono essere previste ed evitate. È una stringa *case sensitive*, e in generale è utilizzata per evitare la replicazione dei *token*.

I nomi sono tutti di tre caratteri per enfatizzare la compattezza di JWT_G ;

- **pubbliche**: possono essere definite dagli utilizzatori dei JWT_G , ma devono essere registrate presso il **JWT_G Internet Assigned Numbers Authority (IANA) Registry** o definite come URI_G , in modo da evitare collisioni;
- **private**: personalizzate ed utilizzate per scambiare informazioni tra due parti in accordo sulle modalità di utilizzo.

Un esempio di *payload* è riportato nel Frammento di codice 7.2.

Frammento di codice 7.2: Esempio di payload JWT

```
{
  sub: "1234567890",
  name: "John Doe",
  admin: true
}
```

Il tutto è poi codificato con $Base64_G$ e forma la seconda parte del *token*.

Per creare la terza parte tutto quello che serve è l'*header* codificato, il *payload* codificato, una chiave (il **secret**) e l'algoritmo specificato nell'*header*. Tutto questo va firmato. Nel Frammento di codice 7.3 è mostrato un esempio utilizzando $HMAC_G$ con SHA_G -256.

Frammento di codice 7.3: Esempio di signature JWT

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

La firma è utilizzata per verificare che chi invia il JWT_G sia chi dice di essere e per assicurare che il messaggio non venga modificato.

L'*output* finale è un insieme di tre stringhe codificate con $Base64_G$ separate da punti ('.') che può essere facilmente trasmessa con il protocollo $HTTP_G$. La stringa ottenuta è inoltre molto più compatta di altre alternative, come le asserzioni $SAML_G$. Un esempio è stato mostrato in Figura 7.6.

È importante notare come le informazioni siano solo codificate con $Base64_G$ e non criptate: questo le rende visibili da chiunque intercetti il *token*. Vanno quindi inseriti solamente dati pubblici e assolutamente non sensibili.

Utilizzo

Nel caso d'uso dell'*autenticazione_G*, quando un utente effettua il *login* utilizzando le proprie credenziali viene generato, e ritornato, un JWT_G che deve essere salvato localmente. Questa modalità differisce da quella "classica", che prevedeva di creare una sessione lato server e di ritornare un $cookie_G$.

Ogniqualvolta l'utente vuole accedere ad un servizio, o risorsa, protetto, il *token* deve essere inviato al server. L'invio avviene tipicamente nell'*header Authorization* usando lo schema **Bearer**. Il contenuto dell'*header* sarà dunque quello mostrato nel Frammento di codice 7.4

Frammento di codice 7.4: Esempio di header HTTP per l'invio di un JWT

```
Authorization: Bearer <token>
```

L'autenticazione_G effettuata in questo modo è *stateless* perché lo stato dell'utente non è mai salvato nella memoria del server. I servizi protetti del server controlleranno se il JWT_G inviato è valido e se permette di accedere alla risorsa richiesta. Dato che i *token* JWT_G sono *self-contained*, tutte le informazioni necessarie sono già presenti, e non è necessario interrogare, nuovamente, il *database*. Il diagramma in Figura 7.7 mostra l'intero processo.

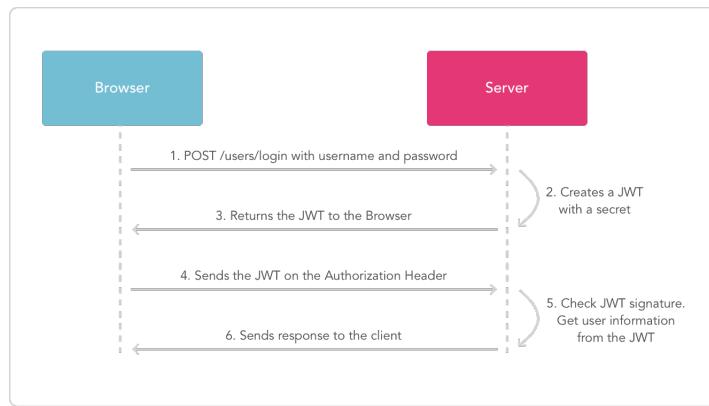


Figura 7.7: Autenticazione con JWT³

Vantaggi

- **Compattezza:** le dimensioni ridotte dei JWT_G li rendono molto versatili.
- **Completezza:** il *payload* può contenere tutti i dati necessari a definire l'identità dell'utente, evitando numerose e ricorrenti interrogazioni al *database*.
- **Sicurezza:** i *token* vengono firmati in modo da poter verificare che nessuno modifichi i dati che contengono.
- **Supporto a Cross-Origin Resource Sharing (CORS)_G:** essendo il processo *stateless* non importa quale dominio fornisce le API_G . Le informazioni necessarie sono tutte nel *token* e nessun $cookie_G$ viene memorizzato lato server, quindi l'autenticazione_G è assolutamente indipendente dal dominio.

³Immagine tratta da [25]

Svantaggi

- **Cross-site scripting (XSS)_G**: nonostante risultino molto più sicuri dei *cookie* per certi tipi di minacce, anche i JWT_G sono vulnerabili ad attacchi di tipo XSS_G ; la grande differenza, però, sta nel fatto che mentre i *cookie* hanno la possibilità di rendersi invisibili a codice JavaScript, i *tokens* (salvati nello *storage* del *browser*) non possono impedire a codice malevolo di accedervi. Una buona strategia per mitigare i rischi di un attacco XSS_G è quello di impostare un valore basso per la scadenza del *token*, prediligendo un rinnovo più frequente dello stesso e limitando temporalmente la possibilità che un *token* rubato possa essere utilizzato per accedere a risorse protette.

Motivazioni della scelta

Si è scelto l'utilizzo dei *tokens* JWT_G per la natura di Catalogue Manager: essendo un sistema *SaaS_G* ogni azione effettuata dall'utente, attraverso il front end, genera una richiesta ad un apposito servizio fornito dal back end. La corretta identificazione dell'utente, quindi, risulta fondamentale, non tanto, attualmente, per una questione di *autorizzazione_G* (che comunque è prevista nelle versioni successive dell'applicazione), ma per una questione di *logging* delle operazioni svolte. I JWT_G grazie alla capacità di includere al loro interno informazioni sull'utente in modo sicuro, firmate in modo tale che qualsiasi tentativo di corruzione del *token* venga rilevato durante la verifica di integrità, sono lo strumento ideale a tale scopo. Si adattano, inoltre, molto bene a diversi tipi di dispositivi, consentendo agli utenti di poter utilizzare l'applicativo anche da ambienti *mobile*. La facilità con cui i JWT_G supportano comunicazioni di tipo *CORS_G* garantisce infine scalabilità all'applicazione.

7.3.2 SSO con SAML

In Figura 7.8 sono mostrate le due modalità di accesso in SSO_G con $SAML_G$: *IdP_G Initiated* e *SP_G Initiated*.

La differenza principale tra le due modalità è rappresentata dall'azione iniziale dell'utente. Con ***IdP_G Initiated*** (Figura 7.8a) l'utente richiede immediatamente di effettuare il *login* e, successivamente, di accedere alla risorsa. Al contrario, con ***SP_G Initiated*** (Figura 7.8b) l'utente cerca fin da subito di accedere alla risorsa. È compito del *SP_G* verificare che l'utente abbia effettuato il *login* e, in caso contrario, di reindirizzarlo alla pagina di *login* dell'*IdP_G*.

Di seguito viene descritta in dettaglio la sequenza di azioni di entrambe le modalità.

IdP Initiated

1. L'utente effettua il *login* presso l'*IdP_G*.
2. L'utente richiede l'accesso ad una risorsa protetta del *SP_G*.
3. (**Opzionale**) Alcuni attributi aggiuntivi possono essere aggiunti alla *SAMLResponse*. Questi attributi vengono ricavati dalla *database* delle *identità_G* e sono determinati sulla base dei requisiti dell'applicazione.
4. L'*IdP_G* ritorna un *form* al *browser* dell'utente con l'asserzione $SAML_G$ ed, eventualmente, gli attributi aggiuntivi. Il *browser* invia come POST il *form* al *SP_G*.

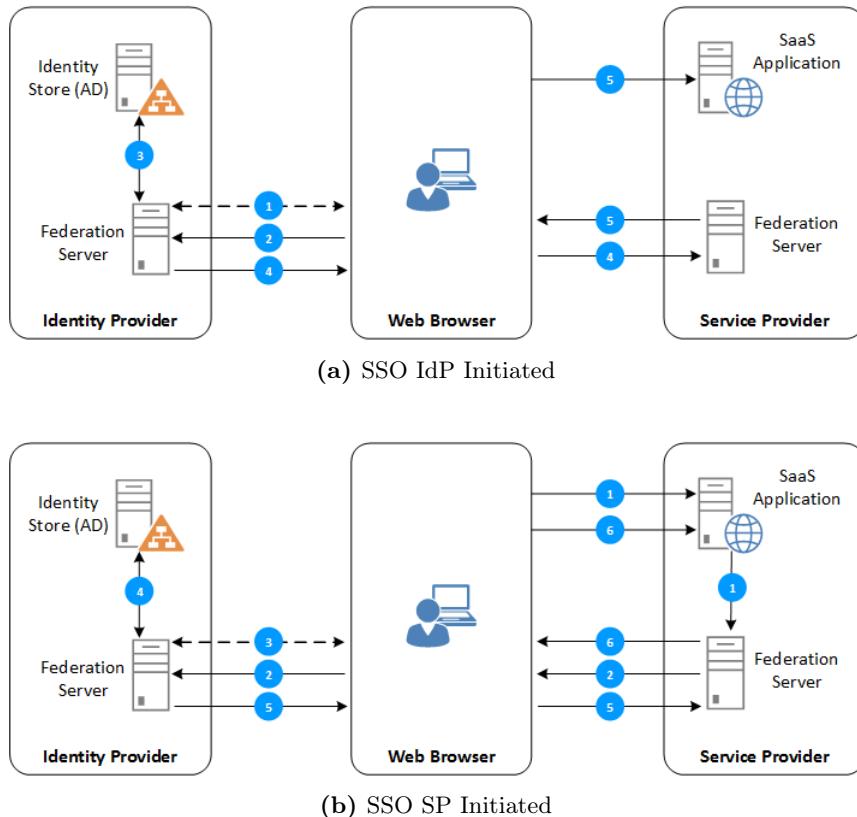


Figura 7.8: Confronto tra SSO SP e IdP Initiated⁴

- Dopo aver validato l'asserzione e la firma dell' IdP_G , il SP_G stabilisce una sessione con l'utente e il *browser* viene reindirizzato alla risorsa richiesta.

Come si può notare l'utente effettua inizialmente il *login* e, successivamente, cerca di accedere alla risorsa. Il primo componente che entra in gioco è l' IdP_G .

SP Initiated

- L'utente cerca di accedere ad una risorsa web attraverso una richiesta al suo SP_G . La richiesta viene reindirizzata verso il server che si occupa della federazione del SP_G per autenticare l'utente.
- Questo server invia un *form* contenente una *SAMLRequest* per l'*autenticazione_G* (**AuthN Request**) al *browser* dell'utente.
- Se l'utente non ha già effettuato il *login*, l' IdP_G gli richiede le credenziali.
- (Opzionale)** Alcuni attributi aggiuntivi possono essere aggiunti alla *SAMLResponse*. Questi attributi vengono ricavati dal *database* delle *identità_G* e sono determinati sulla base dei requisiti dell'applicazione.

⁴Immagini tratte da [24]

5. Il server di federazione dell' IdP_G ritorna un *form* contenente l'asserzione $SAML_G$, eventualmente con gli attributi aggiuntivi, al *browser* dell'utente. Automaticamente il *form* viene inoltrato al server di federazione del SP_G .
6. Dopo aver validato l'asserzione e la firma dell' IdP_G , il SP_G stabilisce una sessione con l'utente e il *browser* viene reindirizzato alla risorsa richiesta inizialmente.

In questo caso, dunque, l'utente prima richiede la risorsa e successivamente esegue il *login*, se non era già stato fatto in precedenza.

Modalità utilizzata

Lo scenario più comune di utilizzo di $SAML_G$ per un'applicazione web è quello SP_G Initiated: l'utente può salvare un segnalibro, o seguire un *link*, per arrivare all'applicazione. Il SP_G reindirizza, se necessario, l'utente verso l' IdP_G per permettergli di autenticarsi. Quest'ultimo crea ad-hoc un'asserzione e la rimanda al SP_G , che decide se concedere l'accesso alla risorsa richiesta.

Al contrario, nella modalità IdP_G Initiated, l'utente è già autenticato presso un IdP_G e da esso accede alle risorse che ha a disposizione.

Catalogue Manager rientra perfettamente in quest'ultima categoria: è, infatti, un'applicazione presente direttamente in Monokee e fortemente legata ad esso (tanto che, in questa prima versione, ne condivide il *database*). Catalogue Manager si fida di Monokee e del suo sistema di *autenticazione*: Monokee è l' IdP_G e Catalogue Manager è il SP_G .

Per questo motivo, nella progettazione dell'applicazione si è scelto di adottare la modalità di SSO_G IdP_G Initiated.

Questa modalità, per quanto più adatta alle esigenze del progetto, richiede che l' IdP_G sia configurato in modo tale da reindirizzare l'utente verso l'applicazione. Fortunatamente, Monokee è stato progettato per supportare entrambe le modalità. Di conseguenza non si è resa necessaria nessuna modifica o configurazione aggiuntiva.

7.4 Principali componenti

Di seguito verranno analizzati i principali componenti dell'architettura dell'applicazione, in particolare modelli (7.4.1) e moduli (7.4.2). Per aiutare la spiegazione, ogni componente viene presentato con un diagramma delle classi che ne mostra attributi e metodi (se presenti). La trattazione non è completa: le *routes*, infatti, non vengono discusse in questa sede, ma lo saranno più avanti. La scelta è motivata dal fatto che di seguito vengono descritte solo le componenti rilevanti ai fine dell'architettura e quelle che meglio mostrano l'applicazione di una progettazione orientata agli oggetti.

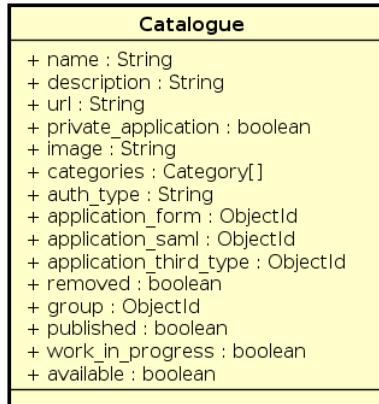
7.4.1 Modelli

Catalogue

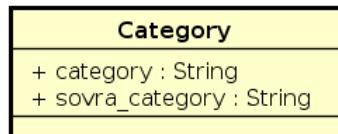
In Figura 7.9 è mostrato il modello Catalogue, che rappresenta il catalogo delle applicazioni. Ogni *document* della *collection* descrive un'applicazione da catalogo, non importa se di quello di Monokee o di quello di un dominio.

Attributi

- **name:** nome dell'applicazione;

**Figura 7.9:** Modello Catalogue

- **description:** descrizione dell'applicazione;
- **url:** URL_G dell'applicazione;
- **private_application:** `true` se e solo se l'applicazione è privata, ovvero se appartiene al catalogo di un dominio; `false` altrimenti;
- **image:** *path* dell'immagine dell'applicazione, salvata su un server di Monokee;
- **categories:** *array* di oggetti **Category** rappresentante le categorie dell'applicazione. **Category** (Figura 7.10) è caratterizzato da due stringhe:
 - **category:** il nome della categoria;
 - **sovra_category:** il nome della "sovra categoria".

**Figura 7.10:** Attributi di Category

La consistenza dell'associazione categoria/sovra categoria è assicurata durante la validazione dei dati effettuata da mongoose.js. Tutte le categorie accettabili sono state definite in un *file*, **categories.json**, che viene letto per assicurare che vengano inseriti solo dati corretti;

- **auth_type:** identifica il tipo di $autenticazione_G$ per l'applicazione;
- **application_form:** riferimento al *document* contenente i dati per l' $autenticazione_G$ *form-based*. Se l'applicazione ha un tipo di $autenticazione_G$ diverso ($SAML_G$ o terzo tipo) è `null`;

- **application_saml**: riferimento al *document* contenente i dati per l'*autenticazione_G* *SAML_G*. Se l'applicazione ha un tipo di *autenticazione_G* diverso (*form-based* o terzo tipo) è **null**;
- **application_third_type**: riferimento al *document* contenente i dati per l'*autenticazione_G* del terzo tipo. Se l'applicazione ha un tipo di *autenticazione_G* diverso (*SAML_G* o *form-based*) è **null**;
- **removed**: *flag* per la *soft deletion_G*. **true** se l'applicazione deve essere rimossa, **false** altrimenti;
- **group**: riferimento al *document* contenente i dettagli del gruppo. È **null** se l'applicazione non appartiene ad un gruppo;
- **published**: **true** se e solo se l'applicazione è pubblicata, **false** altrimenti;
- **work_in_progress**: **true** se e solo se l'applicazione è in manutenzione; **false** altrimenti;
- **available**: **true** se l'applicazione può essere aggiunta agli *application brokers*, **false** altrimenti. Il valore di questo *flag* è modificato durante la rimozione dell'applicazione. Se quest'ultima ha associazioni con qualche utente, infatti, non può essere eliminata definitivamente perché non funzionerebbe più il *SSO_G* per gli utenti che la vedono nel loro *application broker*. Per questo motivo l'applicazione resta presente nel *database*, ma non è più aggiungibile ad alcun *application broker*. È compito di Monokee segnalare la rimozione dell'applicazione originale agli utenti che ne fanno uso.

CatalogueDomain

In Figura 7.11 è mostrato il modello CatalogueDomain, che rappresenta il catalogo privato di un dominio. Ogni *document* della *collection* rappresentata da questo modello contiene un *array* di riferimenti ad applicazioni (*documents* del modello Catalogue): l'*array* è il catalogo del dominio.

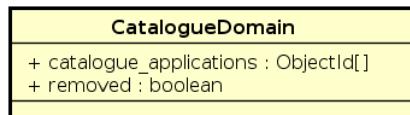


Figura 7.11: Modello CatalogueDomain

Attributi

- **catalogue_applications**: *array* contenente i riferimento alle applicazioni del catalogo del dominio;
- **removed**: *flag* per la *soft deletion_G*. **true** se il catalogo di dominio deve essere rimosso, **false** altrimenti.

CatalogueForm

In Figura 7.12 è mostrato il modello CatalogueForm, che rappresenta i dati necessari per il SSO_G form-based.

Si nota che gli attributi coprono i principali *browser* utilizzati. Sebbene la distinzione per *browser* non sia una buona tecnica per discriminare le azioni da eseguire, questa si è rivelata necessaria a causa delle differenze, anche sostanziali, che essi presentano quando si cerca di effettuare richieste $AJAX_G$. A seconda della tipologia di SSO_G (tramite *form fulfillment* o $AJAX_G$), ciascun attributo contiene i dati necessari a compilare il *form* di accesso o per eseguire la richiesta POST.

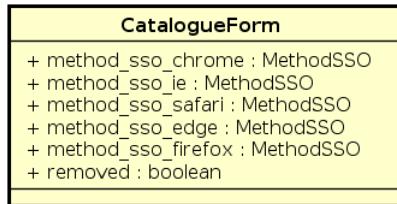


Figura 7.12: Modello CatalogueForm

Attributi

- `method_sso_chrome`: dati per Google Chrome;
- `method_sso_ie`: dati per Microsoft Internet Explorer;
- `method_sso_safari`: dati per Safari;
- `method_sso_edge`: dati per Microsoft Edge;
- `method_sso_firefox`: dati per Mozilla Firefox;
- `removed`: *flag* per la $soft deletion_G$. `true` se il *document* deve essere rimosso, `false` altrimenti.

CatalogueGroup

In Figura 7.13 è mostrato il modello CatalogueGroup, che rappresenta un gruppo di applicazioni del catalogo (di Monokee o di dominio).

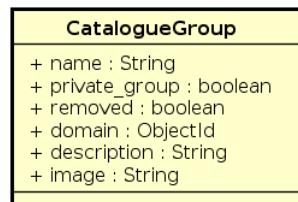


Figura 7.13: Modello CatalogueGroup

Attributi

- **name**: nome del gruppo;
- **private_group**: `true` se e solo se il gruppo è privato, ovvero se è specifico di un dominio; `false` altrimenti;
- **removed**: *flag* per la *soft deletion*_G. `true` se il gruppo deve essere rimosso, `false` altrimenti;
- **domain**: riferimento al dominio di appartenenza del gruppo. È `null` se il gruppo è pubblico;
- **description**: descrizione del gruppo;
- **image**: *path* dell'immagine del gruppo, salvata su un server di Monokee.

CatalogueLog

In Figura 7.14 è mostrato il modello CatalogueLog, che rappresenta un *log* salvato su *database*. Questo modello è utilizzato sia per i *log* delle operazioni eseguite con successo sia per quelle che hanno generato un errore: la distinzione si basa unicamente su un codice definito in fase di progettazione.

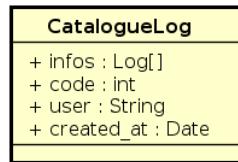


Figura 7.14: Modello CatalogueLog

Attributi

- **infos**: informazioni sull'operazione eseguita. Log è stato progettato per essere il più generico possibile e per poter contenere, di conseguenza, informazioni molto eterogenee tra loro. In Figura 7.15 sono mostrati i suoi attributi. Ogni istanza di Log è caratterizzata principalmente da una coppia chiave/valore, utilizzata per poter recuperare le informazioni durante la visualizzazione dei *log*.

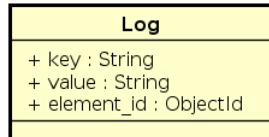


Figura 7.15: Attributi di Log

- **key**: chiave, utilizzata per recuperare l'informazione richiesta;

- **value**: valore dell’informazione;
- **element_id**: ID dell’elemento. È utilizzato per risalire all’elemento (applicazione, gruppo, dominio, eccetera) “bersaglio” dell’operazione eseguita e per recuperare, se necessario, informazioni aggiuntive. L’ID è inoltre usato per identificare univocamente l’elemento, soprattutto nel caso di *log* di errore.
- **code**: codice del *log*;
- **user**: indirizzo email dell’utente che ha eseguito l’operazione;
- **created_at**: *timestamp* di esecuzione dell’operazione;

La struttura del modello **CatalogueLog**, e in particolare di **Log**, rende difficile e possibilmente confusionario l’utilizzo delle informazioni: non tutti i *log* hanno le stesse informazioni, quindi sarebbero necessari molti controlli per capire quali attributi sono presenti di volta in volta. Per questo, prima di essere trasmesse al client, esse sono “riformattate” utilizzando una struttura più facilmente utilizzabile. Tale struttura differisce anche di molto in base al codice del *log*: se il client accetta la struttura specifica non deve eseguire nessun controllo aggiuntivo: nel $JSON_G$ ritornato sono presenti tutti e soli gli attributi necessari.

CatalogueSAML

In Figura 7.16 è mostrato il modello **CatalogueSAML**, che rappresenta le istruzioni per la configurazione del SSO_G con un’applicazione di tipo $SAML_G$. Dato che queste applicazioni devono essere configurate a mano dagli utilizzatori, il catalogo offre solo una serie di azioni da compiere per configurarle al meglio, eventualmente accompagnate da un’immagine.

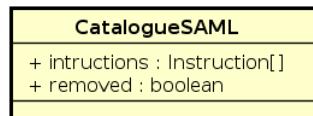


Figura 7.16: Modello CatalogueSAML

Attributi

- **instructions**: *array* contenente le istruzioni per la configurazione, viste come una serie di passi accompagnati da un’immagine. Ogni passo è caratterizzato da nome e descrizione (Figura 7.17);

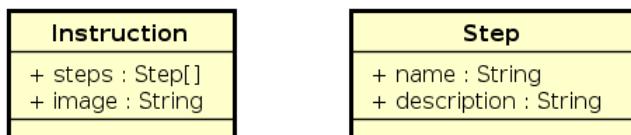


Figura 7.17: Attributi di Instruction e Step

- **removed:** flag per la *soft deletion*_G. true se il document deve essere rimosso, false altrimenti.

CatalogueThirdType

In Figura 7.18 è mostrato il modello CatalogueThirdType, che rappresenta le informazioni necessarie al SSO_G con applicazioni di terze parti. Un'autenticazione_G di questo tipo è caratterizzata da una richiesta POST al servizio di accesso dell'applicazione stessa. Di conseguenza vengono memorizzate tutte le informazioni necessarie a popolare la richiesta.



Figura 7.18: Modello CatalogueThirdType

Attributi

- **post_url:** URL_G verso il quale effettuare la richiesta POST;
- **properties:** dati necessari a popolare la richiesta. In Figura 7.19 viene mostrata in dettaglio la struttura di queste informazioni:

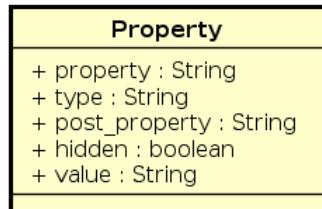
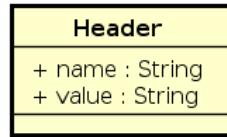


Figura 7.19: Attributi di Property

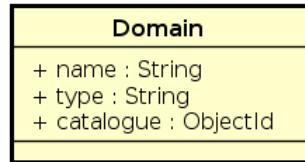
- **property:** nome della proprietà da mostrare all'utente nel form di inserimento dei dati;
- **type:** valore dell'attributo type del tag input corrispondente alla property nel form di inserimento dei dati;
- **post_property:** nome della property da utilizzare nella richiesta;
- **hidden:** true se e solo se il corrispondente tag input deve essere marcato come nascosto;
- **value:** valore della property.
- **headers:** headers da utilizzare nella richiesta. Come mostrato in Figura 7.20, un header è caratterizzato da un nome (**name**) e dal valore (**value**);

**Figura 7.20:** Attributi di Header

- **removed:** *flag* per la *soft deletion*_G. `true` se il *document* deve essere rimosso, `false` altrimenti.

Domain

Il modello Domain viene utilizzato principalmente dal modulo `DomainCatalogue` per recuperare l'ID del catalogo di dominio associato. In Figura 7.21 vengono riportati solo gli attributi utili all'applicazione Catalogue Manager e vengono tralasciati quelli utilizzati solamente da Monokee.

**Figura 7.21:** Modello Domain

Attributi

- **name:** nome del dominio. Viene utilizzato per ricercare un dominio a partire dall'applicazione Catalogue Manager;
- **type:** tipo del dominio. Come già detto, un dominio può essere **personale** (*personal*) o **aziendale** (*company*). I domini *personal* non possono avere un catalogo associato, mentre quelli *company* si. La consistenza dell'attributo `type` è controllata durante la validazione effettuata da `mongoose.js`;
- **catalogue:** riferimento al *document* di `CatalogueDomain` contenente il catalogo del dominio.

7.4.2 Moduli

Di seguito verranno trattati i tre moduli di appoggio principali e maggiormente degni di nota dal punto di vista progettuale, ovvero quelli per la gestione delle immagini, degli errori e per il salvataggio dei log su database.

Gestione delle immagini

In Figura 7.22 è mostrata la gerarchia di moduli utilizzata per gestire le immagini salvate. In particolare, `ImageHandler` contiene i due metodi principali: `save` e `remove`.

Di fatto è un'istanza del *design pattern Template Method*: `save` e `remove` chiamano dei metodi implementati dalle sottoclassi che ritornano i percorsi della cartella in cui ci sarà (o c'è) l'immagine. Questi metodi sono `make_directories` per `save` e `get_base_path` per `remove`. Nelle sottoclassi vengono anche salvati, come campi statici, i percorsi delle cartelle delle immagini.

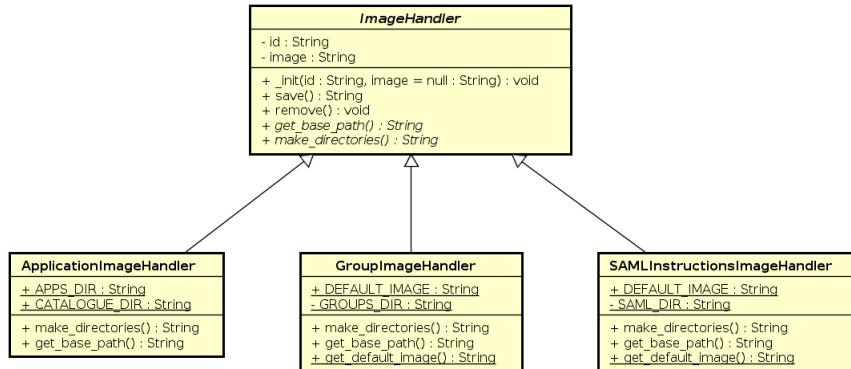


Figura 7.22: Gerarchia per la gestione delle immagini

Attributi

Gli attributi principali sono `id` e `image`: il primo contiene l'ID dell'elemento (applicazione, gruppo o istruzioni $SAML_G$), mentre il secondo rappresenta l'immagine codificata con $Base64_G$. Oltre a questi, ogni sottoclasse contiene dei campi statici per memorizzare i *path* delle cartelle delle immagini. In particolare:

- `APPS_DIR`, `CATALOGUE_DIR`, `GROUPS_DIR` e `SAML_DIR` contengono i *path* delle cartelle delle immagini di, rispettivamente, applicazioni (APPS e CATALOGUE), gruppi e istruzioni per applicazioni $SAML_G$;
- `DEFAULT_IMAGE`: *path* dell'immagine di *default*. Per le applicazioni l'immagine è obbligatoria, quindi non è prevista nessuna immagine di *default*.

Metodi

Come già detto, i due metodi principali sono `save` e `remove`. Il primo salva l'immagine, utilizzando come nome il valore dell'attributo `id`; il secondo rimuove l'immagine con nome uguale al valore dell'attributo `id`. Entrambi, però, sono metodi *template*, ovvero si appoggiano a metodi implementati nelle sottoclassi. In questo modo è possibile implementare le parti invarianti dei due algoritmi una volta sola, evitando la duplicazione del codice. Sono le sottoclassi a fornire il comportamento concreto implementando i metodi lasciati astratti: `ImageHandler` definisce solo lo scheletro e l'ordine delle operazioni, senza preoccuparsi di come saranno implementate.

`make_directories` crea le *directories* che conterranno l'immagine, se non sono già presenti. `get_base_path`, invece, ritorna il percorso generale della *directory* in cui sono salvate le immagini.

`_init` ha la funzione di costruttore, e viene richiamato automaticamente alla creazione di un oggetto tramite `new`.

`get_default_image`, infine, è un metodo statico che ritorna l'immagine di *default*.

ErrorHandler

In Figura 7.23 è mostrata il modulo utilizzato per gestire gli errori, `ErrorHandler`.

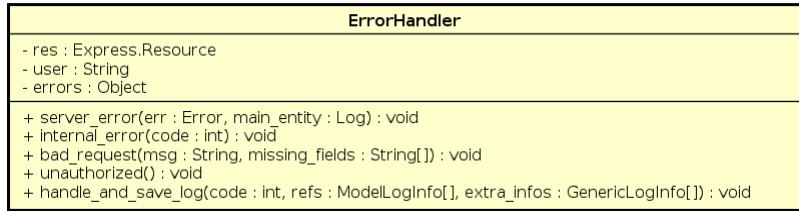


Figura 7.23: Modulo per la gestione degli errori

Ogni risposta di errore passa da qui, indipendentemente dal tipo di errore (non autorizzato, *bad request*, errore del server o interno). In caso di errore, il back end di Catalogue Manager invia al client un $JSON_G$ che segue la seguente struttura (Frammento di codice 7.5):

Frammento di codice 7.5: Struttura del JSON di errore

```
{
  status: false,
  message: "Messaggio di errore",
  error_code: code
}
```

Dove `code` è, ovviamente, il codice dell'errore registrato. Se si tratta di un errore del server, la proprietà `error_code` ha valore `undefined` e non compare nel $JSON_G$ ritornato.

Attributi

- `res`: oggetto `Resource` di `Express.js`. Viene utilizzato per inviare la risposta al client;
- `user`: indirizzo email dell'utente. Viene utilizzato per salvare il *log* dell'errore;
- `errors`: oggetto contenente tutti i codici degli errori utilizzati da Catalogue Manager e il messaggio di errore associato. Un esempio è mostrato nel Frammento di codice 7.6:

Frammento di codice 7.6: Esempio di oggetto contenente i messaggi di errore

```
{
  3000: "Domain catalogue not found",
  3001: "auth_type value is not supported.",
  // ...
  3042: "Private application with public group."
}
```

Metodi

- **server_error**: viene chiamato nel caso in cui ci sia un errore interno del server (esempio quelli generati da mongoose.js). Il parametro `err` contiene l'errore. Per errori normali (ovvero non di validazione, ma sollevati durante l'esecuzione di un *middleware*) viene semplicemente inviata una risposta di errore e salvato il *log*. Altrimenti (errore di validazione) viene fatto il *parsing* per recuperare le informazioni errate da salvare nel *log* per funzioni di *debug*. Al fine di collegare il *log* ad un'entità di Catalogue Manager (applicazione, gruppo, eccetera), il parametro `main_entity` può contenere le informazioni necessarie per memorizzare tale collegamento;
- **internal_error**: invia una risposta di errore con codice `code` e messaggio `errors[code]`, senza salvare nessun *log*;
- **bad_request**: invia una risposta con *status* $HTTP_G$ 400. Nella risposta viene anche inserito l'*array* `missing_fields`, se presente, per segnalare se alcuni parametri obbligatori non sono stati ricevuti dal servizio invocato. Il messaggio di errore è contenuto in `msg`;
- **unauthorized**: invia una risposta con *status* $HTTP_G$ 401;
- **handle_and_save_log**: chiama `internal_error` per gestire l'errore e poi salva il *log*. Oltre alle informazioni sulle principali entità coinvolte nell'errore (applicazione, gruppo, ecc) presenti nell'*array* `refs`, possono essere incluse delle informazioni aggiuntive non collegabili ad un modello di mongoose.js che vanno inserite nell'*array* `extra_infos`. In Figura 7.24 sono mostrate le proprietà contenute nei due parametri. `refs` viene passato così com'è alla funzione di salvataggio: `model` rappresenta il nome del modello dell'entità coinvolta, `dispname` il valore dell'attributo `name` dell'entità e `element_id` il suo ID. In alternativa a `dispname` è possibile utilizzare `name`: in questo caso esso contiene il nome dell'attributo dell'entità da mostrare in fase di visualizzazione del *log*. La funzione di salvataggio effettuerà un'interrogazione al *database* per ricavare il valore dell'attributo rappresentato da `name` nel modello `model`. `extra_infos` contiene invece, come già detto, informazioni aggiuntive non collegabili a nessun modello. La sua struttura è molto semplice e simile a quella di Log (`element_id` viene lasciato ad `undefined`).

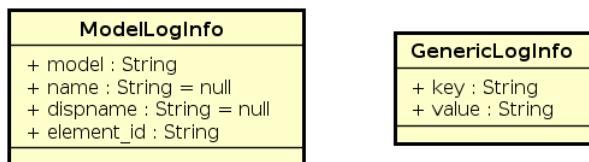


Figura 7.24: Proprietà dei parametri per il salvataggio dei log

Salvataggio dei log su database

In Figura 7.25 è mostrato il modulo DBLogger, utilizzato per il salvataggio dei

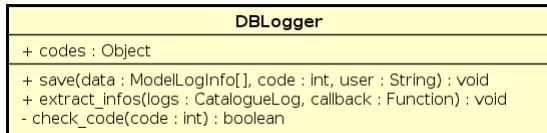


Figura 7.25: Modulo per il salvataggio dei log su database

log su *database*. È l'unico modulo di Catalogue Manager che scrive sulla *collection* rappresentata dal modello `CatalogueLog` e, grazie alla funzione `extract_infos`, rende facilmente utilizzabili le informazioni salvate su tale *collection*. Memorizza inoltre tutti i codici di *log* presenti nell'applicazione.

Attributi

L'unico attributo di `DBLogger` è `codes`, che contiene tutti i codici dei log corrispondenti a operazioni riuscite (quindi non quelli per gli errori, memorizzati in `ErrorHandler`). Questi codici sono accedibili tramite proprietà di questo attributo, che ha la seguente struttura (Frammento di codice 7.7):

Frammento di codice 7.7: Esempio di oggetto contenente i codici dei log

```
{
    APP_CREATED: 1,
    PRIVATE_APP_CREATED: 2,
    APP_REMOVED: 3,
    PRIVATE_APP_REMOVED: 4,
    // ...
    ACCESS_ALLOWED: 29,
    ACCESS_DENIED: 30
}
```

Metodi

- `save`: salva i *log* su *database*. `user` rappresenta l'indirizzo email dell'utente e `code` il codice del *log*. Quest'ultimo viene validato attraverso la funzione `check_code`, che ritorna `true` se e solo se il codice è contenuto in `codes` o se è un codice di errore. Nell'`array` `data`, invece, sono contenute le informazioni da salvare. La struttura di questo parametro è conforme a quella di `ModelLogInfo`, in modo da mantenere la flessibilità data dalla doppia proprietà per il nome da mostrare. Come già detto, infatti, `model` rappresenta il nome del modello dell'entità coinvolta nell'operazione, `dispname` il valore dell'attributo `name` dell'entità e `element_id` il suo ID. In alternativa a `dispname` è possibile utilizzare `name`: in questo caso esso contiene il nome dell'attributo dell'entità da mostrare in fase di visualizzazione del *log*. La funzione di salvataggio effettuerà un'interrogazione al *database* per ricavare il valore dell'attributo rappresentato da `name` nel modello `model`. Ad esempio, dato il modello *M*, se `name` contiene la stringa "description", verrà memorizzato nell'attributo `value` di `CatalogueLog` il valore `description` del *document* del

modello M identificato univocamente da `element_id`. In Tabella 7.2 è mostrato un esempio con dei valori.

ID	<i>description</i>
5770fadf7a1725abe77e0382	Lorem ipsum dolor sit amet
575ffd1c8259f70d4b775646	Mauris suscipit semper dui

Tabella 7.2: Esempio di salvataggio di un log

Supponiamo

```
data.element_id == 5770fadf7a1725abe77e0382 e
data.name == description
```

In tal caso verrà salvato come `CatalogueLog.value` la stringa "Lorem ipsum dolor sit amet".

Al contrario, se

```
data.element_id == 5770fadf7a1725abe77e0382 e
data.dispname == Mario Rossi
```

verrà salvato come `CatalogueLog.value` la stringa "Mario Rossi".

- `extract_infos`: questa funzione serve per rendere più facilmente utilizzabili le informazioni contenute il `CatalogueLog`. In particolare, dato un *log* con la struttura mostrata nel Frammento di codice 7.8 restituisce lo stesso *log*, ma con la struttura mostrata nel Frammento di codice 7.9:

Frammento di codice 7.8: Document di CatalogueLog

```
{
  _id: ObjectId,
  code: Number,
  user: String,
  created_at: Date,
  infos: Array[{
    key: String,
    value: String,
    element_id: ObjectId
  }]
}
```

Frammento di codice 7.9: Log riformattato

```
{
  date: Date,
  code: Number,
  infos: Array[{
    application: String,
    group: String,
    domain: String,
    browser: String
    user: String
  }]
}
```

```

        },
        errored: Array[{
            // struttura dipendente dal codice di errore
        }]
    }
}

```

Le proprietà di `infos` sono potenzialmente vuote. Ad esempio, se il `log` riguarda la creazione di un'applicazione pubblica, `group`, `domain` e `browser` saranno stringhe vuote. Le proprietà di `errored`, invece, dipendono dal codice di errore specifico e, in generale, riportano i valori dei dati errati.

7.5 Design Pattern

La progettazione ad oggetti presuppone l'applicazione di alcuni *patterns* noti e molto utilizzati per risolvere problemi comuni. Di seguito verranno esposti quelli utilizzati durante la definizione dell'architettura di Catalogue Manager.

7.5.1 Module Pattern

I moduli sono parte integrante di qualsiasi applicazione di grandi dimensioni, e tipicamente aiutano ad organizzare il codice. In JavaScript ci sono diverse opzioni per implementare un modulo; tra le più conosciute ci sono la **Object literal notation** e il **Module pattern**.

Object Literal

Nella *object literal notation* un oggetto viene descritto come un insieme di coppie nome/valore separate da virgole (',') e contenute tra parentesi graffe ('{}'). Il Frammento di codice 7.10 mostra un esempio:

Frammento di codice 7.10: Oggetto JavaScript in notazione classica

```

var contatore = {
    k: 0,
    incrementa_e_stampa: function() {
        this.k++;
        console.log(this.k);
    }
};

contatore.incrementa_e_stampa(); // stampa "1"
contatore.i = 50; // aggiunto i con valore 50
contatore.k = 20; // k ora vale 20
contatore.incrementa_e_stampa(); // stampa "21"

```

Questa notazione non richiede l'utilizzo dell'operatore `new`. Dall'esterno, tuttavia, chiunque può aggiungere proprietà all'oggetto `contatore`. L'istruzione `contatore.i = 50;` ne è un esempio. Oltre a questo, chiunque può modificare `contatore.k` senza utilizzare il metodo dedicato: non verrà quindi stampato a video il nuovo valore e l'utilizzatore non noterà, inizialmente, nessun risultato.

Questa soluzione manca completamente di **incapsulazione**: tutti possono vedere, modificare o aggiungere proprietà all'oggetto.

Una soluzione migliore: il Module Pattern

In JavaScript, il **Module Pattern** è utilizzato per *emulare* il concetto di classe, in modo da avere attributi e metodi pubblici e privati. È quindi possibile decidere quali parti del modulo esporre e quali no, semplicemente ritornando un oggetto contenente le proprietà pubbliche del modulo.

È importante notare che in JavaScript non è presente il concetto di "*privacy*", in quanto non esistono i modificatori di accesso presenti negli altri linguaggi. Le variabili non possono essere dichiarate *pubbliche* o *private*, ed è necessario utilizzare l'ambito di visibilità delle funzioni per simulare questo concetto. Con il Module pattern le variabili e le funzioni dichiarate dentro il modulo sono private; al contrario, quelle contenute nell'oggetto ritornato sono pubbliche.

Il Frammento di codice 7.11 mostra un esempio:

Frammento di codice 7.11: Module pattern

```
var contatore = (function() {
    var k = 0; // attributo privato

    var incrementa_e_stampa = function() {
        k++;
        stampa();
    };

    var decrementa_e_stampa = function() {
        k--;
        stampa();
    };

    var get_contatore = function() {
        return k;
    };

    // metodo privato
    var stampa = function() {
        console.log(k);
    };

    // pubbliche
    return {
        incrementa_e_stampa: incrementa_e_stampa,
        decrementa_e_stampa: decrementa_e_stampa,
        get_contatore: get_contatore
    };
})();

contatore.incremenata_e_stampa(); // stampa "1"
contatore.decrementa_e_stampa(); // stampa "0"
contatore.k = 20;
contatore.get_contatore(); // ritorna "0"
contatore.k; // "20"
```

Vantaggi

- Aggiunge il concetto di incapsulazione a JavaScript.
- Se vengono aggiunti dei metodi esternamente alla definizione del modulo, questi non possono accedere alle variabili private.

Svantaggi

- Complica l'utilizzo dell'ereditarietà.
- Rispetto alla notazione ad oggetti classica complica il *testing* perché alcuni membri sono inaccessibili.

Contestualizzazione

In Catalogue Manager tutti i moduli sono realizzati con il Module pattern. Questo ha consentito di progettarli in modo molto più *object-oriented* e di ottenere l'incapsulazione che sarebbe altrimenti assente.

7.5.2 Template Method

In Figura 7.26 è riportata la struttura del *design pattern* **Template Method**.

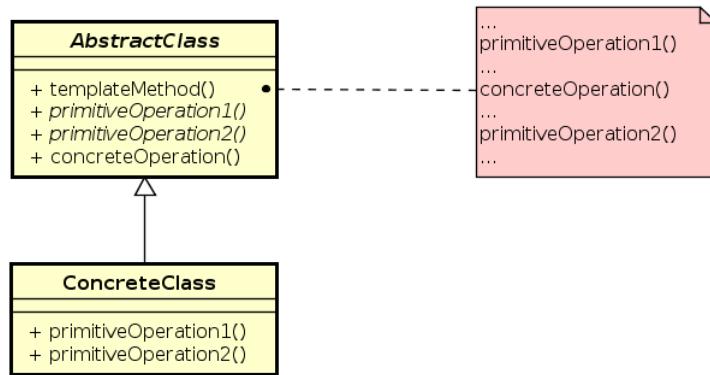


Figura 7.26: Design pattern Template Method

Questo *pattern* consente di definire la struttura di un algoritmo, lasciando alle sottoclassi il compito di implementare alcuni passi secondo le loro necessità. In questo modo si può ridefinire e personalizzare parte del comportamento nelle varie sottoclassi, senza dover riscrivere più volte il codice in comune. Inoltre evita la duplicazione di codice nelle sottoclassi e aderisce all'**Hollywood Principle**⁵.

Vantaggi

- Nessuna duplicazione di codice.

⁵"**Don't call us, we'll call you**". Proviene dalla filosofia di Hollywood, secondo lo quale sono le case produttrici a chiamare gli attori se hanno bisogno di loro. Contestualizzando, le componenti di alto livello (superclassi) decidono quando e come utilizzare le componenti di basso livello (sottoclassi)

- Il riutilizzo di codice avviene per ereditarietà e non per composizione; solo alcuni metodi devono subire l'*override*.
- Le sottoclassi decidono come implementare i passi dell'algoritmo, migliorando la flessibilità.

Svantaggi

- Aumenta la difficoltà di *debugging*.
- Rende più difficile comprendere il flusso di esecuzione.

Contestualizzazione

Il *design pattern* Template Method è utilizzato per la gerarchia di classi che gestiscono le immagini salvate su *file system*. `ImageHandler` espone due metodi, `save` e `remove`, che chiamano dei metodi astratti che devono essere implementati dalle sottoclassi.

7.5.3 Middleware

Questo *pattern* consente di definire uno o più intermediari tra i vari componenti *software* dell'applicazione in modo da semplificare la loro connessione e collaborazione. In generale è molto utile nello sviluppo e nella gestione di sistemi distribuiti complessi, contesto nel quale Catalogue Manager si colloca perfettamente.

Viene utilizzato da Express.js per fornire una libreria di funzioni comuni: definisce una serie di livelli (o funzioni) per gestire le varie richieste dell'applicazione. Tutti i componenti del *pattern* **Middleware** sono collegati l'uno con l'altro e ricevono a turno una richiesta in ingresso finché uno di questi non decide di partire con l'elaborazione: l'*output* di un *middleware* diventa l'*input* per il successivo. Per questo è anche conosciuto con il nome di **Pipeline**.

Anche mongoose.js utilizza questo concetto: come già spiegato è possibile definire delle funzioni da eseguire prima o dopo un'operazione specificata.

Middleware è strettamente legato al **Chain of Responsibility**, descritto più avanti.

Vantaggi

- Riutilizzo di codice comune a più parti dell'applicazione.
- Aderenza al **Single Responsibility Principle**.⁶

Svantaggi

- Se mal utilizzato rende difficile la comprensione del flusso di controllo.

Contestualizzazione

In Catalogue Manager il *pattern* Middleware viene utilizzato ampiamente sia nel contesto di Express.js sia in quello di mongoose.js.

Per Express.js serve per registrare funzioni di validazione del *token*, le *routes*, il gestore dell'errore 404 e così via.

⁶Ogni elemento di un *software* deve avere una sola responsabilità, e tale responsabilità deve essere interamente incapsulata dall'elemento stesso. Tutti i servizi offerti dall'elemento dovrebbero essere strettamente allineati a tale responsabilità.

Per mongoose.js, invece, serve, ad esempio, a validare i dati inviati (URL_G e categorie in particolare), a forzare il mantenimento di vincoli personalizzati (come quello che impone che applicazioni pubbliche non abbiano lo stesso nome), ad escludere dalle ricerche su *database* determinati valori (ad esempio le applicazioni con il `flag removed == true` o con `available == false`), eccetera.

7.5.4 Chain of Responsibility

Il *pattern Chain of Responsibility* permette ad un oggetto di inviare un evento senza sapere chi lo riceverà e chi lo gestirà. Questo passaggio rende i due (o più) oggetti parte di una catena: ciascun oggetto nella catena può gestire l'evento, passarlo al successivo o entrambi.

Lo scopo è quello di evitare un collegamento statico tra chi emette l'evento e chi lo gestisce: formando una catena la richiesta viene passata da un oggetto all'altro, senza sapere chi e quando la gestirà. Ogni nodo della catena decide se esaudire la richiesta o no, delegando, in quest'ultimo caso, l'onere al nodo successivo. La catena viene attraversata finché un nodo non può eseguire l'ordine del mittente. In questo modo si evita l'**accoppiamento** fra il mittente di una richiesta e il destinatario.

Un *pattern* di questo tipo si lega facilmente e fortemente al *Middleware*, e trova in lui una naturale istanziazione: Express.js, infatti, lo utilizza per la gestione dei *middlewares* (così come mongoose.js) e del *routing*. In Figura 7.27 è mostrata la struttura.

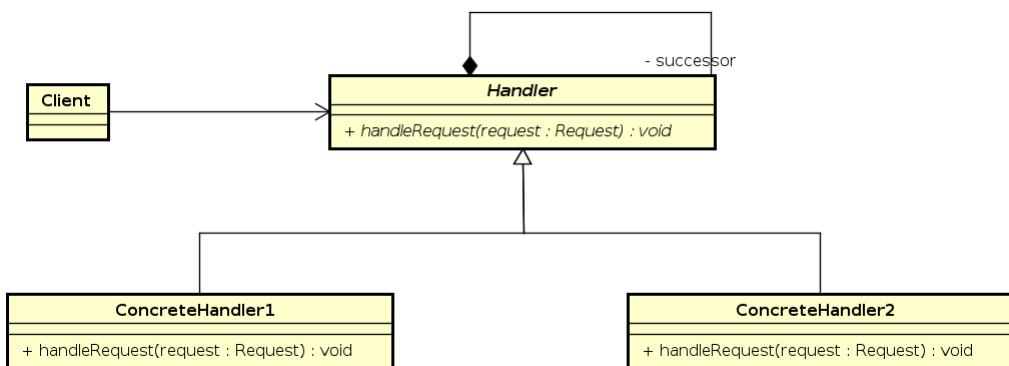


Figura 7.27: Design pattern ChainOfResponsibility

Nel gergo di Express.js i *middlewares* corrispondono agli oggetti *ConcreteHandler* del *design pattern*. Sebbene il comportamento e lo scopo sia quasi identico, l'implementazione di Express.js presenta alcune differenze:

- i *middlewares* di Express.js possono essere classi con un metodo `handle` o semplici funzioni, in pieno accordo con lo stile funzionale utilizzato dalla maggioranza delle librerie e delle applicazioni scritte in Node.js. In Catalogue Manager è stata utilizzata principalmente la seconda versione;
- il *design pattern* prevede che l'oggetto *Handler* abbia un riferimento *successor* all'*Handler* successivo. Express.js, invece, passa al metodo di esecuzione del *middleware* una *callback*. Il *middleware*, eseguendo la *callback*, passa nuova-

mente il controllo all'oggetto del server di Express.js il quale passerà il controllo al successivo *middleware*;

- Express.js divide i *middlewares* in due gruppi: standard e di gestione degli errori. Si distinguono per il numero di parametri che possono gestire (tre e quattro, rispettivamente). Ogni *middleware* può decidere a quale dei due passare il controllo semplicemente variando il numero di parametri (nel secondo caso va specificato l'errore da gestire). Questa funzionalità serve per permettere la gestione di errori senza utilizzare i costrutti `try catch`, tipici dei linguaggi imperativi, ma inefficaci quando si utilizza lo stile di programmazione asincrono;
- ogni *middleware* di Express.js deve essere invocato con i seguenti parametri: l'eventuale errore, l'oggetto della richiesta (`Express.Request`), l'oggetto della risposta (`Express.Response`), la `callback` da utilizzare per passare il controllo al successivo *middleware*. L'ordine è importante.

Vantaggi

- Riduzione dell'accoppiamento.
- Maggiore flessibilità nell'assegnazione delle responsabilità.

Svantaggi

- La gestione di una richiesta non è garantita, sia per la possibile mancanza di un *middleware* con quella specifica responsabilità sia per una configurazione sbagliata della catena.

Contestualizzazione

Come già detto per il *pattern* Middleware, il Chain of Responsibility viene utilizzato largamente da Express.js e mongoose.js.

Capitolo 8

Implementazione

8.1 Autenticazione a Catalogue Manager

8.1.1 Implementazione dei JWT

L'autenticazione a Catalogue Manager avviene tramite $SAML_G$ nella modalità **IdP_G Initiated**. L'accesso deve essere eseguito attraverso Monokee e, in seguito ad esso, viene generato un *token* JWT_G trasmesso al front end di Catalogue Manager, che, successivamente, invierà questo *token* al back end per ogni richiesta effettuata. Quest'ultimo verificherà la firma del *token* stesso per controllare che le informazioni non siano state alterate e che l'utente sia chi dice di essere.

Non essendo previsto un sistema di ruoli, l'unico controllo viene effettuato al momento della generazione del *token* e riguarda l'associazione tra utente e applicazione: in questo modo può accedere a Catalogue Manager solo chi ha questa applicazione nel proprio *application broker*.

Il periodo di validità del *token* è di nove ore, dopodiché è necessario autenticarsi nuovamente. È stato scelto questo periodo di tempo in modo da rendere valido il *token* per una giornata di lavoro e per imporre una nuova $autenticazione_G$ ogni giorno.

Il Frammento di codice 8.1 mostra un esempio di *header* di un *token* di Catalogue Manager.

Frammento di codice 8.1: Header di un JWT di Catalogue Manager

```
{  
  typ: "JWT",  
  alg: "HS256"  
}
```

Come descritto in precedenza, questa parte del *token* contiene le informazioni su tipo e algoritmo di firma. La scelta per la modalità di firma è caduta su **HS256** ($HMAC_G$ con SHA_G -256) e non su $RS256$ (RSA_G) per un motivo sostanzialmente legato alle *performance*: la firma e la verifica, nel primo caso, sono molto più veloci rispetto al secondo. Inoltre, la dimensione del *token* è molto minore.

La differenza sostanziale, comunque, risiede nella modalità di firma. Con $HMAC_G$ chi pensa all' $autenticazione_G$ ha la chiave (il "secret"); fornisce la chiave e il messaggio all'algoritmo scelto, che produce la versione firmata. Dopodiché invia il messaggio originale e quello firmato al verificatore, che, disponendo della stessa chiave, ricalcola la firma, controllando se quanto ottenuto è uguale a quanto ricevuto. Con RSA_G , invece, esistono due chiavi diverse (pubblica e privata). Il messaggio viene firmato

con quella privata e inviato al verificatore, che, attraverso l'algoritmo di verifica (ora diverso da quello di firma) controlla se il messaggio originale è uguale a quello ottenuto utilizzando la chiave pubblica.

È chiaro che nel primo caso entrambe le parti condividono la stessa chiave e il verificatore può, se vuole, generare messaggi che vengono validati senza problemi. Nel secondo caso non accade, perché la chiave pubblica funziona solo per verificare i messaggi e non per firmarli.

Nel caso specifico di Catalogue Manager il compito di creare i *token* e quello di verificarli ricadono su due componenti della stessa applicazione, quindi una può fidarsi, senza problemi, dell'altra (la verifica vera e propria viene effettuata utilizzando un noto modulo di Node.js, *expressjwt*). Tolto questo problema, $HMAC_G$ funziona molto più velocemente, e pertanto è stato preferito a RSA_G .

Il Frammento di codice 8.2 mostra un esempio di *payload* di un *token* di Catalogue Manager.

Frammento di codice 8.2: Payload di un JWT di Catalogue Manager

```
{
  id: "5783969ebf947349bb34f7b7",
  email: "matteo.dipirro@email.com",
  iat: 1468413576,
  exp: 1470213576
}
```

Come si nota le informazioni fornite sono molto basilari e, come già detto, nessuna di esse è riservata. Oltre ai vincoli sulla validità vengono inviati solo l'ID dell'applicazione Catalogue Manager e l'indirizzo email dell'utente, che sarà utilizzato per salvare i *log* delle operazioni svolte durante la sessione. È ovviamente compito del client inviare il *token* ad ogni richiesta, tramite *header* $HTTP_G$. L'ID di Catalogue Manager serve per verificare la presenza dell'associazione tra l'applicazione e l'utente.

8.1.2 Modalità SAML IdP Initiated

In Figura 8.1 è mostrato il processo di $autenticazione_G$ IdP_G Initiated utilizzato. L'utente deve innanzitutto autenticarsi a Monokee e avere in uno dei suoi *application brokers* Catalogue Manager. Il compito di accertare l'identità è quindi di Monokee.

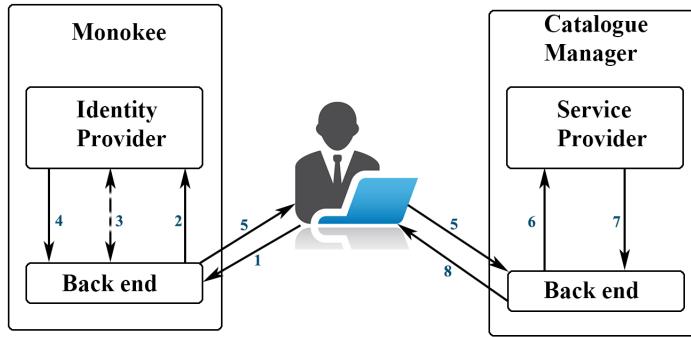


Figura 8.1: Single Sign On IdP Initiated in Catalogue Manager

Di seguito saranno descritti i passi compiuti, automaticamente, durante l'accesso.

1. L'utente clicca sull'applicazione Catalogue Manager. Questo causa l'invocazione del servizio di Monokee che si occupa del SSO_G IdP_G Initiated.
2. Il back end di Monokee effettua una chiamata all' IdP_G , parametrizzata con il *token* dell'utente, l'ID dell'applicazione (Catalogue Manager in questo caso) e l'ID del dominio di appartenenza dell'utente.
3. L' IdP_G può comunicare ulteriormente con il back end di Monokee per ricevere degli attributi aggiuntivi dell'utente. Nel caso di Catalogue Manager, viene prelevato l'indirizzo email.
4. L' IdP_G fornisce la *SAMLResponse* al back end di Monokee.
5. Il back end di Monokee invia la *SAMLResponse* tramite *form* al *browser* dell'utente. Questo, automaticamente, la inoltra ad un servizio di Catalogue Manager ($/acs$), l'*Assertion Consumer Service* di $SAML_G$. Il compito di questo servizio è di inoltrare la *SAMLResponse* al SP_G , aspettare la risposta, validarla e generare il *token* corrispondente.
6. Il back end di Catalogue Manager chiama il SP_G inviando la *SAMLResponse* ricevuta in precedenza.
7. Il SP_G risponde al back end di Catalogue Manager.
8. Il back end di Catalogue Manager chiama il front end in base alla risposta del SP_G :
 - **Errore nella risposta del SP_G :** viene chiamato il front end di Catalogue Manager e segnalato l'errore.
 - **Successo:** arriva l'indirizzo email dell'utente. Viene controllato che l'utente sia collegato all'applicazione:
 - se no, l'errore viene notificato al front end di Catalogue Manager;
 - se sì, viene generato il *token* dell'utente.

8.2 Elenco dei servizi implementati

In Tabella 8.1 vengono elencati i servizi $REST_G$ implementati ed esposti dal back end di Catalogue Manager. I nomi sono stati assegnati seguendo le convenzioni utilizzate dal *team* di sviluppo.

Tutti i servizi che contengono `auth/` nel loro URL_G richiedono l'invio di un *token* valido per essere eseguiti.

Nome	Descrizione
<code>/acs</code>	Data la <i>SAMLResponse</i> , verifica che l'utente abbia accesso all'applicazione Catalogue Manager e genera il <i>token</i> corrispondente. All'interno di esso viene memorizzato l'indirizzo email dell'utente e il suo identificativo all'interno del <i>database</i> di Monokee.
<code>/auth/add_applications_in_group</code>	Aggiunge un insieme di applicazioni ad un gruppo. Le applicazioni e il gruppo vengono identificate tramite i loro ID, e vengono effettuati tutti i controlli necessari per non creare inconsistenze. In particolare: <ul style="list-style-type: none"> • la visibilità dell'applicazione e del gruppo deve essere la stessa; • se il gruppo è privato (specifico di un dominio, l'applicazione deve appartenere al catalogo di quel dominio); • l'applicazione non deve essere già inserita in un altro gruppo.
<code>/auth/create_application</code>	Crea un'applicazione pubblica o privata e la aggiunge al catalogo di Monokee, nel primo caso, o al catalogo di un dominio, nel secondo.
<code>/auth/create_domain_catalogue</code>	Crea un catalogo di dominio per il dominio selezionato. Se il dominio ha già un catalogo viene sollevato un errore.
<code>/auth/create_group</code>	Crea un gruppo (pubblico o privato). Durante la creazione di un gruppo è possibile decidere quali applicazioni faranno parte di quel gruppo. Se vengono forniti degli ID di applicazioni vengono fatti gli stessi controlli descritti per <code>add_applications_in_group</code> .
<code>/auth/get_access_by_intervals</code>	Ritorna il numero di accessi effettuati nelle ultime 24 ore e nell'ultima settimana.

/auth/get_activities_from_to	Ritorna un qualsiasi tipo di <i>log</i> (riguardanti le attività su gruppi, applicazioni, accessi o domini) in un intervallo temporale deciso dall'utente.
/auth/get_addable_applications	Ritorna, dato l'ID di un gruppo, le applicazioni che possono essere aggiunte a tale gruppo.
/auth/get_application_details	Ritorna, dato l'ID di un'applicazione, i dettagli di tale applicazione. I dettagli includono nome, descrizione, URL_G , informazioni sul tipo di <i>autenticazione</i> _G , eccetera.
/auth/get_applications_by_dates	Ritorna il numero di applicazioni, pubbliche e private, aggiunte e rimosse in un intervallo temporale deciso dall'utente. Le informazioni vengono separate per settimana.
/auth/get_applications_by_group	Dato l'ID di un gruppo, ritorna le applicazioni contenute in quel gruppo.
/auth/get_applications_by_intervals	Ritorna le applicazioni aggiunte e rimosse nelle ultime 24 ore, nell'ultima settimana, mese e anno.
/auth/get_applications_list	Ritorna la lista di tutte le applicazioni pubbliche.
/auth/get_categories	Ritorna la lista delle categorie, complete delle sotto categorie.
/auth/get_domain_by_name	Ritorna una lista di domini con nome simile a quello inviato dall'utente.
/auth/get_domain_catalogue	Dato un dominio, ritorna le applicazioni del catalogo di quel dominio.
/auth/get_domains_with_catalogue	Ritorna i domini che hanno un catalogo di applicazioni associato.
/auth/get_group_details	Ritorna i dettagli di un gruppo di applicazioni, come il nome, la descrizione e l'immagine.
/auth/get_groups	Ritorna i gruppi pubblici di applicazioni presenti in Catalogue Manager.
/auth/get_properties	Ritorna le properties utilizzabili per l'accesso in SSO_G di tipo <i>form-based</i> .
/auth/get_sovra_category_stats	Ritorna il numero di applicazioni, pubbliche e private, presenti in ciascuna sotto categoria.
/auth/get_sso_actions	Ritorna una lista delle actions da eseguire per l'accesso in SSO_G di tipo <i>form-based</i> .
/auth/get_stats	Ritorna il numero totale di applicazioni e gruppi, pubblici e privati. Vengono inoltre ritornate il numero di applicazioni presenti in ciascuna categoria.
/auth/get_user_info	Ritorna le informazioni dell'utente che ha effettuato il <i>login</i> .

/auth/remove_application	Rimuove definitivamente un'applicazione dato il suo ID.
/auth/remove_applications_from_group	Rimuove un insieme di applicazioni da un gruppo.
/auth/remove_domain_catalogue	Dato l'ID di un dominio, rimuove il catalogo di quel dominio. Le applicazioni contenute in quel catalogo vengono eliminate definitivamente.
/auth/remove_group	Dato l'ID di un gruppo, lo rimuove. La rimozione del gruppo causa solo la cancellazione del collegamento con le applicazioni che vi erano inserite. Queste applicazioni restano nel catalogo di Monokee, o in quello di un dominio.
/auth/set_learning_catalogue	Consente di utilizzare il <i>plug in</i> di Monokee per il <i>learning</i> . Il <i>learning</i> serve per il SSO_G tramite <i>form fulfillment</i> .
/auth/set_learning_mobile	Consente inserire i dati per il <i>form fulfillment</i> per <i>browsers mobile</i> .
/auth/set_maintenance	Consente di aggiornare il <i>flag</i> di manutenzione di un'applicazione.
/auth/update_3rd_type_data	Consente di aggiornare i dati di $autenticazione_G$ per un'applicazione con accesso del terzo tipo. L' $autenticazione_G$ per queste applicazioni viene effettuata tramite una richiesta POST $AJAX_G$. È quindi possibile modificare tutti gli elementi di questa richiesta, come le <i>properties</i> utilizzate e gli <i>headers</i> della richiesta.
/auth/update_formSSO_data	Consente di aggiornare i dati di $autenticazione_G$ per un'applicazione di tipo <i>form-based</i> . Oltre all'utilizzo del <i>learning</i> , quindi, è possibile impostare questi dati anche manualmente. Questa possibilità si rivela molto utile se l'accesso viene effettuato tramite richieste POST $AJAX_G$ e non tramite <i>form-fulfillment</i> .
/auth/update_general_data	Consente di aggiornare i dati generali di un'applicazione, come il nome, la descrizione, l' URL_G o l'immagine.
/auth/update_group	Consente di aggiornare i dati generali di un gruppo, ma non di aggiungere o rimuovere applicazioni da esso.

/auth/update_saml_instructions	Consente di aggiornare le istruzioni per la configurazione dell'accesso tramite $SAML_G$. Essendo l'applicazione nel catalogo generale e indipendente dagli utilizzatori, è permesso solo configurare le istruzioni (viste come una serie di azioni da compiere) per configurare le informazioni richieste per l'utilizzo di $SAML_G$. È compito di chi aggiunge l'applicazione desiderata al proprio <i>application broker</i> inserire queste informazioni.
--------------------------------	---

Tabella 8.1: Servizi REST esposti dal back end di Catalogue Manager

Capitolo 9

Verifica e validazione

9.1 Verifica

Per **verifica** si intende il processo che ha il compito di accertare che l'esecuzione delle altre attività non abbia introdotto errori. Risponde quindi alla domanda "**Did I build the system right?**".

Per assicurare un'alta qualità, il *software* è stato analizzato con due diverse tecniche:

- **analisi statica**, non prevede l'esecuzione del prodotto;
- **analisi dinamica**, prevede l'esecuzione del prodotto.

Si è cercato di raggiungere la correttezza del prodotto **by construction**: la definizione di regole e strategie ha portato alla stesura di codice facile da verificare fin dall'inizio. L'alternativa è la cosiddetta correttezza **by correction**, che prevede di correggere al volo senza però fare nulla per evitare a priori gli errori più comuni.

9.1.1 Analisi statica

L'analisi statica si basa sulla verifica del codice senza eseguire lo stesso: attraverso strumenti esterni è possibile analizzare quanto scritto per rilevare le principali fonti di bugs_G .

Uno dei principali strumenti in questo campo è JSHint. Grazie alla semplicità e velocità di configurazione, JSHint si integra con moltissimi *editors* di testo e analizza il codice "al volo", durante la scrittura. Dopo averlo configurato con le regole di codifica più adatte, JSHint analizzerà il codice alla ricerca di violazioni delle regole precedentemente impostate. Molto spesso, infatti, un utilizzo errato del linguaggio di programmazione scelto inserisce nel prodotto dei bugs_G difficili da trovare. L'esempio più classico sono le variabili definite e non inizializzate, ma ce ne sono molti altri: variabili utilizzate e mai dichiarate, conversioni di tipo automatiche, variabili solo scritte e mai lette, eccetera. Un linguaggio estremamente permissivo come JavaScript, inoltre, rende molto difficile l'individuazione di errori di questo tipo, che diventano visibili solo a *run time*: come sarà spiegato successivamente, risalire alla causa degli errori rilevati a *run time* è spesso molto complesso.

JSHint, invece, permette di rilevare alcuni problemi senza eseguire il codice, velocizzando e facilitando il percorso verso la correttezza. La configurazione di regole di codifica ben precise, inoltre, aiuta a scrivere codice facilmente comprensibile, leggibile e, di conseguenza, manutenibile.

9.1.2 Analisi dinamica

Secondo quanto affermato da Edsger Wybe Dijkstra nel trattato **Notes On Structured Programming** (1970), il *testing*¹ di un *software* può solo dimostrare la presenza di bugs_G , ma mai la loro assenza:

*Program testing can be used to show the presence of bugs, but never to
show their absence!*

Edsger Wybe Dijkstra

È dunque necessario stabilire il giusto numero di *test* per poter trovare il maggior numero di bug_G , evitando però di cadere nella ridondanza. La progettazione e la scrittura dei *test* ha un costo che deve essere analizzato e non sottovalutato.

Quando si parla di *testing* del *software* è necessario distinguere tra fault_G e failure_G . L'esecuzione dei *test* rileva i failures_G , ma è compito di chi ha scritto il codice risalire ai faults_G : questo compito è particolarmente difficile, visto che non esiste un modo per risalire in modo univoco al fault_G presente.

Il bersaglio di un *test* può variare: un singolo modulo, un gruppo di moduli (collegati tra loro per uso, struttura, scopo o comportamento) o l'intero sistema. In base a questa distinzione possono essere identificati tre livelli di *testing*:

- **unità**: verificano il funzionamento in isolamento di un elemento *software* atomico (**unità**²);
- **integrazione**: verificano l'interazione tra i diversi componenti. Tipicamente si utilizza una strategia incrementale che mira a mettere insieme solo parti già provate, soprattutto per *software* di grandi dimensioni. La strategia opposta è detta *big bang testing* e consiste nel mettere insieme tutte le componenti un'unica volta e provarle tutte insieme. In questo modo, però, risalire ai faults_G può essere molto complicato;
- **sistema**: verificano il comportamento dell'intero sistema. A questo punto i *test* di unità e integrazione hanno rilevato la maggior parte dei difetti del prodotto: i *test* di sistema mirano quindi a provare le caratteristiche non funzionali del *software*, come sicurezza o velocità.

Esiste un altro tipo di *test*, di **regressione**, che viene eseguito in seguito alla modifica di una parte P del sistema S per verificare che la modifica di P non abbia introdotto errori né in P né nelle altre parti di S che hanno relazione con P .

Catalogue Manager è stato verificato utilizzando la strategia incrementale descritta in precedenza. Inizialmente sono stati provati i moduli comuni utilizzati dalle *routes* e le *routes* che non utilizzano nessun modulo. Successivamente, man mano che il loro funzionamento è stato verificato, le componenti sono state combinate fino a verificare l'intero sistema nel complesso.

Per essere efficaci i *test* devono essere **ripetibili**: le condizioni (stato iniziale del sistema, *input* e *output* attesi) di svolgimento devono quindi essere sempre uguali, in modo da poter valutare e confrontare i risultati ottenuti. Affinché la valutazione dei risultati sia obiettiva, inoltre, l'esecuzione dei *test* deve essere automatizzata. Per questo motivo, per la maggior parte della verifica del prodotto sviluppato è stato

¹Con *testing* si intendono le tecniche di analisi dinamica.

²Con **unità** si intende la più piccola quantità di *software* che conviene provare da sola. Nel presente documento una unità è rappresentata da un modulo JavaScript o da una *route* di Express.js.

utilizzato il *framework_G* Mocha.js, con la libreria di asserzioni offerta da Express.js. Per prove veloci è stato anche utilizzato lo strumento DHC: ogni *test* fallito (che ha rivelato *failure_G*) è successivamente stato aggiunto ai *test* automatizzati.

Lo sviluppo parallelo del front end, infine, ha portato alla luce diversi *failures_G* e ha contribuito alla definizione dei casi di *test*.

9.2 Validazione

Per **validazione** si intende il processo che conferma, attraverso misurazioni e prove oggettive, che le specifiche del *software* siano conformi ai bisogni dell'utente. Risponde quindi alla domanda **"Did I build the right system?"**.

I **test di accettazione** sono utilizzati per fornire queste prove oggettive, e controllano che i requisiti fissati dall'utente e identificati nell'iniziale processo di analisi siano stati rispettati.

Catalogue Manager rispetta tutti i requisiti definiti inizialmente, a ha soddisfatto tutte le aspettative.

Capitolo 10

Valutazione retrospettiva

10.1 Obiettivi fissati

Gli obiettivi fissati erano molto ambiziosi: è stato inizialmente richiesto un elevato numero di funzionalità, che è cresciuto ulteriormente durante lo svolgimento dell'attività di stage. Monokee è un prodotto in continua evoluzione: di conseguenza i requisiti sono soggetti a cambiamenti frequenti e, talvolta, inaspettati.

Le principali modifiche apportate, in corso d'opera, ai requisiti sono le seguenti:

- **tipologie di applicazioni permesse:** come descritto in 4.2 sono presenti tre tipi di autenticazione diverse: *form based*, tramite $SAML_G$ o di terzo tipo. Inizialmente, tuttavia, solamente le prime due erano state richieste. L'introduzione del nuovo tipo ha portato all'aggiunta di altri servizi $REST_G$, di un nuovo modello di mongoose.js e la modifica di alcuni servizi e moduli già scritti;
- **servizio di logging e statistiche:** sebbene non fossero funzionalità definite inizialmente, sono diventate importanti grazie al parallelo sviluppo di Monokee, e sono pertanto state inserite come requisiti desiderabili;
- **sviluppo di un'applicazione mobile:** ha richiesto la definizione di un ulteriore servizio $REST_G$ (`set_learning_mobile`) e la modifica di uno dei modelli utilizzati da Catalogue Manager (`CatalogueForm`).

Oltre a questo, lo sviluppo parallelo del front end ha, in qualche caso, portato alla luce degli aspetti tralasciati durante l'esposizione del progetto e durante le riunioni di analisi del prodotto e dei requisiti.

10.2 Obiettivi raggiunti

Grazie alla collaborazione del personale di *Athesys S.r.l.* e del tutor aziendale Roberto Griggio, il periodo di apprendimento e di analisi si è concluso velocemente e in anticipo rispetto a quanto pianificato. Monokee è un prodotto molto complesso, e capirne a pieno la struttura si è rivelato, inizialmente, molto complicato: al di là delle funzionalità offerte, il sistema di IdM_G ha richiesto del tempo per essere compreso a fondo. Inoltre, l'accesso a Catalogue Manager è di tipo federato con $SAML_G$, e riuscire a capire il funzionamento di questo standard si è rivelato difficile, ma fondamentale.

L'architettura riflette quella classica utilizzata per applicazioni web basate sul $framework_G$ Express.js ed ha potuto beneficiare della struttura esistente di Monokee.

Se da un lato questo è un aspetto positivo, perché molte scelte si sono rivelate già prese, adottate e provate, dall'altro è stata necessaria una particolare attenzione all'integrazione con gli elementi esistenti, in modo da preservare la scalabilità e la manutenibilità del prodotto.

Nonostante i cambiamenti nei requisiti e le nuove funzionalità richieste, tutti i requisiti, sia obbligatori che desiderabili, sono stati soddisfatti.

10.3 Conclusioni

Al momento della scelta dello stage, mi sono trovato davanti una lista di aziende proposte dall'Università di Padova, operanti in diversi settori. Dopo la giornata di presentazione di STAGE-IT, ho avuto modo di approfondire la conoscenza di alcune di esse tramite colloqui individuali e, in seguito, ho scelto *Athesys S.r.l.* per i contenuti dei progetti proposti ed il buon clima aziendale che si è percepito già in fase di contatto. L'ambito dell'*IAM_G* era a me sconosciuto, e l'impatto iniziale è stato difficile, ma grazie al clima di collaborazione e all'atteggiamento socievole del *team* è stato facile capire come funzionavano le cose.

Giunti al termine dell'attività risulta comunque doveroso valutare in modo critico quanto prodotto. Il sistema, nell'arco di tutto il suo sviluppo, è stato continuamente oggetto di confronto e discussione da parte del *team* di sviluppo dell'intera applicazione Monokee; questo approccio, se da una parte ha portato alcune volte a modifiche successive (talvolta onerose) di requisiti e componenti, ha indubbiamente permesso al prodotto di crescere come visione collettiva di un insieme di persone anziché di un unico responsabile, guadagnandone in termini di funzionalità individuate e completezza di visione del problema.

Ritengo quindi l'esperienza decisamente positiva. Tutti gli obiettivi fissati sono stati portati a termine ed è nata un'applicazione funzionante ed estensibile che diventerà parte di un progetto interessante e dal grande futuro. L'intera attività di stage si è svolta con il *team* di sviluppo di Monokee, che mi ha aiutato ad apprendere nozioni nuove ed affascinanti. Questa esperienza mi ha fatto capire che da soli possiamo fare molto poco e che grandi risultati possono derivare solo dalla collaborazione di persone motivate e sempre disponibili ad aiutarsi in caso di bisogno.

Una delle cose che mi ha colpito maggiormente durante tutta il periodo di stage è la serietà con cui è stata presa in considerazione ogni mia singola opinione; ognuno ha sempre potuto esprimere le proprie idee, i propri dubbi e le proprie perplessità su quanto stava sviluppando così come sulle altre parti dell'applicativo. Questo atteggiamento mi ha fatto superare la mia iniziale timidezza e mi ha spinto ad esprimere le mie opinioni ed i miei dubbi ogniqualvolta ne sentissi l'utilità, nella speranza di provare a contribuire con qualche piccola idea.

L'atteggiamento socievole del *team*, inoltre, mi ha permesso di integrarmi all'interno del gruppo con gran semplicità, favorendo la collaborazione con gli altri componenti e risultando parte attiva nella buona riuscita dell'attività.

Ho infine avuto la possibilità di applicare le nozioni apprese durante le molte ore di studio e di lezione di questi tre anni università, ma anche di studiare in modo autonomo nuovi concetti e tecnologie. Si sono rivelate molto utili tutte le conoscenze acquisite durante l'insegnamento di Ingegneria del Software, dall'esistenza di database *NoSQL_G*, all'utilizzo di Node.js e JavaScript per lo sviluppo del back end di applicazioni web, dalle tecniche di analisi dei requisiti alle modalità di stesura della documentazione di un'architettura e del codice.

Ho capito ancora meglio come l'obiettivo del Corso di Laurea in Informatica sia quello di fornire un "modo di pensare" che permetta allo studente (o laureato) di far fronte a qualsiasi situazione, di affrontare autonomamente le difficoltà e di saper scegliere sempre la strada giusta. Molti studenti, tuttavia, durante il loro percorso di studi, valutano la loro preparazione informatica in relazione alle tecnologie che hanno appreso fino a quel momento, ricercando e richiedendo le tecnologie più nuove e criticando i corsi che propongono linguaggi per loro obsoleti. Questa visione, a mio giudizio, è eccessivamente miope: l'informatica è un'area dinamica e in continua evoluzione. Le cose progrediscono e cambiano molto velocemente, e quello che oggi è attuale già domani potrebbe essere obsoleto. Al contrario, le capacità analitiche e di ragionamento sono utili in qualsiasi situazione e permettono di adattarsi al meglio alle difficoltà.

È quindi soprattutto merito di questa visione se mi sono travato ad apprendere velocemente nuove tecnologie e nozioni in ambiti a me sconosciuti e se non ho riscontrato significativi problemi nel corso dell'attività di stage. La capacità di gestire un progetto complesso, di affrontare difficoltà e scadenze e di ragionare a fondo su ogni aspetto sono state provvidenziali e utilissime.

It is not the task of the University to offer what society asks for, but to give what society needs.

Edsger Wybe Dijkstra

Glossario

AD Active Directory

Servizio di directory sviluppato da Microsoft che consente di autenticare e autorizzare utenti e computer in reti di dominio Windows, assegnando policy di sicurezza e aggiornando i software. 121

Affidabilità

Certezza di corretto funzionamento che un impianto, un apparecchio, un dispositivo può dare in base alle sue caratteristiche tecniche e di fabbricazione. 5

AgID Agenzia per l'Italia Digitale

Agenzia pubblica italiana che si occupa di perseguire il massimo livello di innovazione tecnologica nell'organizzazione e nello sviluppo della pubblica amministrazione. 121

Agile

Serie di principi per lo sviluppo di software secondo i quali i requisiti e le soluzioni si evolvono attraverso la collaborazione tra individui. Si basa su quattro principi fondamentali, definiti nel Manifesto for Agile Software Development:

- gli individui e le interazioni sono più importanti di processi e strumenti;
- il software funzionante è più importante di una documentazione esaustiva;
- la collaborazione con il cliente è più importante della negoziazione dei contratti;
- la rapida risposta al cambiamento è più importante di seguire una pianificazione.

. 5

AJAX Asynchronous JavaScript and XML

Tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Lo sviluppo di applicazioni web con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è detto che le richieste di caricamento debbano essere necessariamente asincrone. 121

AM Access Management

Sistema integrato di tecnologie in grado di consentire alle organizzazioni di definire politiche di accesso alle risorse. 9, 121

API Application Programming Interface

Insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per il completamento di un determinato compito all'interno di un certo programma. 121

Autenticazione

Processo attraverso il quale l'utente fornisce le credenziali necessarie per ottenere l'accesso ad un sistema o ad una particolare risorsa; una volta che l'utente ha effettuato l'autenticazione, viene creata una sessione, riferita in tutte le interazioni fra l'utente ed il sistema, finché l'utente effettua log out o la sessione viene terminata per altre ragioni (ad esempio per un timeout). 5, 8–10, 13–17, 24, 29, 30, 49, 55, 56, 64, 65, 72, 74, 75, 77, 78, 80–83, 87, 100, 101, 104, 105

Autorizzazione

Processo che garantisce che gli utenti correttamente autenticati possano accedere solo alle giuste risorse. Il collegamento tra utente e risorse viene stabilito in base alle politiche di accesso alla risorsa, precedentemente decise dal proprietario della risorsa stessa. 5, 9, 10, 14–17, 79

Base64

È un sistema di numerazione posizionale che usa 64 simboli che viene usato principalmente come codifica di dati binari nelle email. L'algoritmo che effettua la conversione suddivide il file in gruppi da 6 bit, i quali possono quindi contenere valori da 0 a 63. Ogni possibile valore viene convertito in un carattere ASCII. L'algoritmo causa un aumento delle dimensioni dei dati del 33%, poiché ogni gruppo di 3 byte viene convertito in 4 caratteri. Questo supponendo che per rappresentare un carattere si utilizzi un intero byte. 76, 77, 89

Best practice

Esperienza, procedura o azione più significativa, o comunque che ha permesso di ottenere i migliori risultati, relativamente a svariati contesti e obiettivi preposti. 4, 20

Branching

Duplicazione di un oggetto soggetto a controllo di versionamento, in modo che le modifiche su tale oggetto possano procedere in parallelo su due (o più) ramificazioni diverse. 42

Bug

Errore nella scrittura del codice sorgente di un programma software. 107, 108

Cloud

Paradigma di erogazione di risorse informatiche, come l'archiviazione, l'elaborazione o la trasmissione di dati, caratterizzato dalla disponibilità on demand attraverso Internet a partire da un insieme di risorse preesistenti e configurabili. Le risorse non vengono pienamente configurate e messe in opera dal fornitore apposta per l'utente, ma gli sono assegnate, rapidamente e convenientemente, grazie a procedure automatizzate, a partire da un insieme di risorse condivise con altri utenti lasciando all'utente parte dell'onere della configurazione. 12, 17–21

Cookie

Un cookie è simile ad un piccolo file, memorizzato nel computer da siti web durante la navigazione, utile a salvare le preferenze e a migliorare le prestazioni dei siti web. In questo modo si ottimizza l'esperienza di navigazione da parte dell'utente. Nel dettaglio, un cookie è una stringa di testo di piccola dimensione inviata da un web server ad un web client (di solito un browser) e poi rimandati indietro dal client al server (senza subire modifiche) ogni volta che il client accede alla stessa porzione dello stesso dominio web. 77, 78

CORS Cross-Origin Resource Sharing

Meccanismo che permette alle risorse sul web di essere richieste da domini differenti da quello di appartenenza della risorsa stessa . 121

CSV Comma Separated Values

Formato di file basato su file di testo utilizzato per l'importazione ed esportazione (ad esempio da fogli elettronici o database) di una tabella di dati. Non esiste uno standard formale che lo definisca, ma solo alcune prassi più o meno consolidate. 121

DAC Discretionary Access Control

Meccanismo di controllo degli accessi alle risorse messe a disposizione da un sistema informatico definito dalla Trusted Computer System Evaluation Criteria, nel quale i soggetti possiedono l'ownership degli oggetti da loro creati e possono concedere o revocare a loro discrezione alcuni privilegi ad altri soggetti. In un sistema informatico nel quale è realizzata una politica di controllo degli accessi di tipo DAC, l'autorizzazione degli accessi alle risorse del sistema è basata sull'identità degli utenti e/o sul gruppo utenti di appartenenza . 121

DBA DataBase Administrator

Professionista che, all'interno di un'azienda o di un ente, si occupa di installare, configurare e gestire sistemi di archiviazione dei dati, più o meno complessi, consultabili e spesso aggiornabili per via telematica.

Configura gli accessi al database, realizza il monitoraggio dei sistemi di archiviazione, si occupa della manutenzione del server, della sicurezza degli accessi interni ed esterni alla banca dati e definisce, al contempo, le politiche aziendali di impiego e utilizzo delle risorse costituite dal database.

Tra i principali compiti di questa figura professionale c'è quello di preservare la sicurezza e l'integrità dei dati contenuti nell'archivio. 121

DBMS DataBase Management System

Sistema software progettato per consentire la creazione, la manipolazione e l'interrogazione efficiente di database. È ospitato su architettura hardware dedicata (server) oppure su semplice computer. 121

DLM Data Lifetime Management

Processo di gestione dell'informazione di business durante tutto il suo ciclo di vita, a partire dalla creazione e memorizzazione iniziale fino alla sua obsolescenza e conseguente eliminazione.

I prodotti a supporto del DLM consentono di automatizzare le attività coinvolte, organizzando i dati in livelli diversi e semplificando la migrazione da un livello ad un altro secondo i criteri definiti. 121

DRY Don't Repeat Yourself

Principio di progettazione e sviluppo secondo cui andrebbe evitata ogni forma di ripetizione e ridondanza logica nell'implementazione di un sistema software. Il principio venne inizialmente enunciato da Andy Hunt e Dave Thomas nel loro libro *The Pragmatic Programmer*:

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

Il DRY viene spesso citato in relazione alla duplicazione del codice, ovvero nell'accezione stretta secondo cui il software non dovrebbe contenere sequenze di istruzioni uguali fra loro. Si tratta però di un concetto più ampio, che si applica a ogni parte di un sistema software. 121

Efficienza

Capacità di evitare sprechi di energia, risorse, tempo e denaro durante lo svolgimento di una specifica attività. In senso più matematico, è la misura di quanto l'input è ben impiegato per produrre un output. 5

Failure

Effetto indesiderato visibile durante l'esecuzione del prodotto. 108, 109

Fault

Causa del malfunzionamento del prodotto. Di solito è un errore nella scrittura del codice sorgente ed è chiamato comunemente bug. 108

File system

Meccanismo con il quale i file sono posizionati e organizzati o su un dispositivo di archiviazione o su una memoria di massa, come un disco rigido. 39

Framework

Modalità strutturata, pianificata e permanente, che supporta una prassi, una metodologia, un progetto, un sistema di gestione; nello sviluppo software, inoltre, indica più specificamente una logica di supporto su cui un software può essere progettato e realizzato. 5, 37, 40, 43, 46, 109, 110

HMAC keyed-Hash Message Authentication Code

Modalità per l'autenticazione di messaggi basata su una funzione di hash, utilizzata in diverse applicazioni legate alla sicurezza informatica. Tramite HMAC è possibile garantire sia l'integrità che l'autenticità di un messaggio: utilizza una combinazione del messaggio originale e una chiave segreta per la generazione del codice. Una caratteristica peculiare di HMAC è il non essere legato a nessuna funzione di hash in particolare, per rendere possibile una sostituzione della funzione nel caso non fosse abbastanza sicura. Nonostante ciò le funzioni più utilizzate sono MD5 (attualmente considerata poco sicura) e SHA-2 . 121

HTTP HyperText Transfer Protocol

Protocollo a livello applicativo usato come principale sistema per la trasmissione d'informazioni sul web, ovvero in un'architettura tipica client-server. Le specifiche del protocollo sono gestite dal World Wide Web Consortium (W3C). 44, 121

IAM Identity and Access Management

Disciplina che consente ai giusti individui di accedere alle giuste risorse nel giusto momento per le giuste ragioni. L'IAM risponde alla necessità di garantire un adeguato accesso alle risorse in ambiti tecnologici sempre più eterogenei. 11, 111, 121

IANA Internet Assigned Numbers Authority

Organismo che ha responsabilità nell'assegnazione degli indirizzi IP. 121

IDaaS IDentity as a Service

Identity and Access Management fornito as a Service, ovvero in modo cloud based. 121

Identità

Combinazione di attributi generici (come nome, cognome, indirizzo, ecc.) e specifici (rilevanti a livello aziendale) che consentono di identificare in modo univoco un utente. 5, 8, 9, 11–17, 21–24, 27, 79, 80

IdM Identity Management

Sistema integrato di tecnologie, criteri e procedure in grado di consentire alle organizzazioni di facilitare, e al tempo stesso controllare, gli accessi degli utenti ad applicazioni e dati critici . 8, 121

IDoT IDentity of Things

Assegnazione di identificatori univoci e attributi specifici ad oggetti per consentire loro di comunicare e interagire con le altre entità attraverso Internet. 122

IdP Identity Provider

Entità responsabile di:

- fornire un'identità agli utenti che interagiscono con un sistema;
- assicurare che l'utente sia chi dice di essere;
- fornire ulteriori informazioni sull'utente.

. 22, 28, 122

IGA Identity Governance and Administration

I prodotti IGA consentono ai responsabili di amministrare e gestire i ruoli, le licenze e i privilegi degli utenti nell'azienda estesa e anche nel cloud. Queste soluzioni consentono alle organizzazioni di evitare le violazioni relative alla suddivisioni degli incarichi, supportare le politiche di business ed eliminare gli accessi inappropriati. Controllando e verificando l'attività degli utenti e rafforzando il controllo degli accessi, le organizzazioni possono ottenere una governance più efficace, prevenire le minacce alle informazioni riservate e la frode di identità. 122

I/O Input/Output

Interfacce messe a disposizione dal sistema operativo, o più in generale da qualunque sistema di basso livello, ai programmi per effettuare uno scambio di dati o segnali con altri programmi, col computer o con lo stesso sistema. 121

IoT Internet of Things

Neologismo riferito all'estensione di Internet al mondo degli oggetti e dei luoghi

concreti. Gli oggetti si rendono riconoscibili e acquisiscono intelligenza grazie al fatto di poter comunicare dati su se stessi e accedere ad informazioni aggregate da parte di altri. 122

IT Information Technology

Settore caratterizzato dall'impiego di computer e tecnologie di telecomunicazione per immagazzinare, prelevare, trasmettere e manipolare dati in contesti aziendali. 122

ITIL Information Technology Infrastructure Library

Insieme di linee guida ispirate dalla pratica (best practice) nella gestione di servizi IT. Consistono in una serie di pubblicazioni che forniscono indicazione sull'erogazione di servizi IT di qualità e sui processi e mezzi necessari a supportarli. 122

JSON JavaScript Object Notation

Semplice formato completamente indipendente dal linguaggio di programmazione per lo scambio di dati. JSON è basato su due strutture:

- un insieme di coppie nome/valore;
 - un elenco ordinato di valori
- . 122

JWT JSON Web Token

Stringa che rappresenta un insieme di affermazioni come un oggetto JSON, in modo da poterle firmare o criptare digitalmente. 77, 122

MAC Mandatory Access Control

Tipo di controllo d'accesso attraverso il quale il sistema operativo vincola la capacità di un soggetto di eseguire diverse operazioni su un oggetto o un obiettivo. 122

Merging

Fusione delle modifiche effettuate su un oggetto su due ramificazioni (branch) distinti. Il risultato è un unico oggetto che contiene l'unione delle modifiche. 42

NoSQL Not only SQL

Movimento che promuove sistemi software dove la persistenza dei dati è caratterizzata dal fatto di non utilizzare il modello relazionale, di solito usato dai database tradizionali. L'espressione NoSQL fa riferimento al linguaggio SQL, che è il più comune linguaggio di interrogazione dei dati nei database relazionali, qui preso a simbolo dell'intero paradigma relazionale. Questi archivi di dati il più delle volte non richiedono uno schema fisso (schemaless), evitano spesso le operazioni di giunzione (join) e puntano a scalare in modo orizzontale . 122

On-premises

Installazione ed esecuzione del software direttamente su macchina locale, sia essa aziendale che privata. L'approccio on-premises per la distribuzione/utilizzo del software è stato ritenuto la norma fino al 2005, data oltre la quale si è progressivamente ampliato l'utilizzo di software che esegue su computer remoti. 12, 17–21

Phishing

Tipo di truffa effettuata su Internet attraverso la quale un malintenzionato cerca di ingannare la vittima convincendola a fornire informazioni personali, dati finanziari o codici di accesso, fingendosi un ente affidabile in una comunicazione digitale. 13

Provisioning

Insieme di attività attraverso le quali viene garantito all’utente l’autorizzazione presso un sistema o un’applicazione. Il processo include l’assegnazione di diritti e privilegi all’utente, in modo tale da garantire la sicurezza del sistema. 5

RBAC Role-Based Access Control

Approccio a sistemi ad accesso ristretto per utenti autorizzati. Si basa su tre regole fondamentali:

- assegnazione dei ruoli;
- autorizzazione dei ruoli;
- autorizzazione alla transazione.

. 122

Repository

Ambiente di un sistema informativo, in cui vengono gestiti i metadati, attraverso tabelle relazionali. 43, 44

Resilienza

Capacità di un sistema di adattarsi alle condizioni d’uso e di resistere all’usura in modo da garantire la disponibilità dei servizi erogati. 11

REST REpresentational State Transfer

Tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web. Si basa su tre principi:

- lo stato dell’applicazione e le funzionalità sono divisi in risorse web;
- ogni risorsa è unica e indirizzabile usando sintassi universale per uso nei link ipertestuali;
- tutte le risorse sono condivise come interfaccia uniforme per il trasferimento di stato tra client e risorse, questo consiste in:
 - un insieme vincolato di operazioni ben definite;
 - un insieme vincolato di contenuti, optionalmente supportato da codice a richiesta;
 - un protocollo:
 - * client-server;
 - * privo di stato (stateless)
 - * memorizzabile in cache (cacheable)
 - * a livelli

. 44, 122

Rollback

Operazione che permette di riportare il database a una versione o stato precedente. 9, 74

RSA

Algoritmo di crittografia asimmetrica, inventato nel 1977 da Ronald Rivest, Adi Shamir e Leonard Adleman utilizzabile per cifrare o firmare informazioni. 75, 76, 100, 101

SaaS Software as a Service

Modello di distribuzione del software applicativo dove un produttore di software sviluppa, opera (direttamente o tramite terze parti) e gestisce un'applicazione web che mette a disposizione dei propri clienti via Internet. I clienti non pagano per il possesso del software bensì per l'utilizzo dello stesso. 122

SAML Security Assertion Markup Language

Standard informatico per lo scambio di dati di autenticazione e autorizzazione (dette asserzioni) tra domini di sicurezza distinti, tipicamente un identity provider (entità che fornisce informazioni di identità) e un service provider (entità che fornisce servizi). 122

Scalabilità

Capacità di aumentare le risorse per ottenere un incremento (idealmente) lineare nella capacità del servizio. La caratteristica principale di un'applicazione scalabile è costituita dal fatto che un carico aggiuntivo richiede solamente risorse aggiuntive anziché un'estesa modifica dell'applicazione stessa. 5, 11

Servizio di directory

Programma (o insiemi di programmi) che provvede ad organizzare e memorizzare informazioni e a gestire risorse condivise all'interno di reti di computer, fornendo anche un controllo degli accessi sul loro utilizzo. 14, 20, 28, 34

SHA Secure Hash Algorithm

Famiglia di cinque diverse funzioni crittografiche di hash sviluppate a partire dal 1993 dalla National Security Agency (NSA) e pubblicate come standard federale dal governo degli USA. Il SHA produce un message digest, o "impronta del messaggio", di lunghezza fissa partendo da un messaggio di lunghezza variabile. La sicurezza di un algoritmo di hash risiede nel fatto che la funzione non sia reversibile (non sia cioè possibile risalire al messaggio originale conoscendo solo questo dato) e che non deve essere mai possibile creare intenzionalmente due messaggi diversi con lo stesso digest. Gli algoritmi della famiglia sono denominati SHA-1, SHA-224, SHA-256, SHA-384 e SHA-512: le ultime 4 varianti sono spesso indicate genericamente come SHA-2, per distinguerle dal primo. Il primo produce un digest del messaggio di soli 160 bit, mentre gli altri producono digest di lunghezza in bit pari al numero indicato nella loro sigla. 122

Soft deletion

Cancellazione unicamente "logica" di informazioni da un database. Solitamente viene effettuata impostando a true uno specifico flag, ad esempio (is_deleted). 74, 83–85, 87, 88

SP Service Provider

Organizzazione che fornisce agli utenti servizi di vario tipo. 23, 28, 122

SPID Sistema Pubblico Identità Digitale

Sistema unico di login per l'accesso ai servizi online della pubblica amministrazione e dei privati aderenti . 122

SSO Single Sign On

Il Single Sign On consente di autenticarsi una volta sola e di avere accesso, in modo del tutto automatico, a varie applicazioni di un sistema. Elimina il bisogno di autenticarsi separatamente a ciascuna applicazione e/o sistema. 122

Stakeholder

Soggetto (o un gruppo di soggetti) influente nei confronti di un'iniziativa economica, che sia un'azienda o un progetto. 7

TCO Total Cost of Ownership

Approccio sviluppato da Gartner nel 1987, utilizzato per calcolare tutti i costi del ciclo di vita di un'apparecchiatura informatica IT, per l'acquisto, l'installazione, la gestione, la manutenzione e il suo smaltimento. 122

Thread

Suddivisione di un processo in due o più filoni o sottoprocessi, che vengono eseguiti concorrentemente da un sistema di elaborazione monoprocesso (multithreading) o multiprocesso (multicore). 38, 39

UML Unified Modeling Language

Linguaggio di modellazione basato sul paradigma orientato agli oggetti. Svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa UML per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 122

URI Uniform Resource Identifier

Stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, un indirizzo di posta elettronica, eccetera. 122

URL Uniform Resource Locator

Sequenza di caratteri che identifica univocamente l'indirizzo di una risorsa in Internet, tipicamente presente su un host server, come ad esempio un documento, un'immagine, un video, rendendola accessibile ad un client che ne faccia richiesta attraverso l'utilizzo di un web browser. 122

Workflow

Applicazione che automatizza le procedure e i processi aziendali di lavoro cooperativo. 5

XML eXtensible Markup Language

Metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. 122

XSS Cross-site scripting

Vulnerabilità che affligge siti web dinamici che impiegano un insufficiente controllo dell'input nei form. Un XSS permette di inserire o eseguire codice lato client al fine di attuare un insieme variegato di attacchi. Nell'accezione odierna, la tecnica ricomprende l'utilizzo di qualsiasi linguaggio di scripting lato client tra i quali JavaScript. Secondo un rapporto di Symantec nel 2007 l'80% di tutte le violazioni è dovuto ad attacchi XSS. 122

Acronimi e abbreviazioni

ABAC Attribute Based Access Control. 9, 10, 12

AD Active Directory. 14, 28, 34

AgID Agenzia per l'Italia Digitale. 22

AJAX Asynchronous JavaScript and XML. 30, 84, 105

AM Access Management. 8, 11, 12, 115

API Application Programming Interface. 45, 71, 78

CORS Cross-Origin Resource Sharing. 78, 79

CRM Customer Relationship Management. 34

CRUD Create, Read, Update, Delete. 69

CSV Comma Separated Values. 70

DAC Discretionary Access Control. 9, 10

DBA DataBase Administrator. 4

DBMS DataBase Management System. 41

DLM Data Lifetime Management. 4

DRY Don't Repeat Yourself. 71

ERP Enterprise Resource Planning. 34

HMAC keyed-Hash Message Authentication Code. 75–77, 100, 101

HTTP HyperText Transfer Protocol. 17, 28, 30, 40, 72, 75, 77, 91, 101, 117

I/O Input/Output. 38, 39, 42

IAM Identity and Access Management. 2, 5, 8, 11, 17, 18, 20, 21, 27, 117

IANA Internet Assigned Numbers Authority. 77

IDaaS IDentity as a Service. 17, 19–21, 26, 34, 35

IdM Identity Management. 8, 9, 11, 110, 117

IDoT IDentity of Things. 11

IdP Identity Provider. 14–17, 23–25, 28, 47, 51, 52, 72, 79–81, 100–102, 117

IGA Identity Governance and Administration. 21

IoT Internet of Things. 8, 11, 12

IT Information Technology. 4, 8, 24, 34

ITIL Information Technology Infrastructure Library. 4

JSON JavaScript Object Notation. 70, 71, 74, 75, 86, 90

JWT JSON Web Token. 72, 75–79, 100, 117

MAC Mandatory Access Control. 9, 10

NoSQL Not only SQL. 41, 111

RBAC Role-Based Access Control. 9, 10, 12

REST REpresentational State Transfer. 28, 40, 69, 74, 103, 110, 118

SaaS Software as a Service. 17–21, 79

SAML Security Assertion Markup Langage. 13, 16, 17, 28, 30, 47, 49–51, 54, 56, 68, 73, 75, 77, 79, 81–83, 86, 89, 100, 102, 106, 110

SHA Secure Hash Algorithm. 76, 77, 100

SP Service Provider. 14, 17, 23, 24, 28, 47, 50, 51, 79–81, 102, 119

SPID Sistema Pubblico Identità Digitale. 22–24

SSO Single Sign On. 5, 12, 13, 15, 17, 18, 26–31, 75, 79, 81, 83, 84, 86, 87, 102, 104, 105

TCO Total Cost of Ownership. 21

UML Unified Modeling Language. 3

URI Uniform Resource Identifier. 40, 50, 51, 69, 72, 77

URL Uniform Resource Locator. 30, 31, 50, 51, 55, 56, 65, 75, 82, 87, 98, 103–105

XML eXtensible Markup Language. 16, 70

XSS Cross-site scripting. 79

Bibliografia

Riferimenti bibliografici

- [1] Alasdair Gilchrist. *An Executive Guide to Identity Access Management*. RG Consulting, apr. 2005.
- [2] Mary Ruddy. *Choosing Between On-Premises Federation and Federation IDaaS*. <https://www.gartner.com/doc/2677315/choosing-onpremises-federation-federation-idaas>. Settembre 2015 (cit. alle pp. 18, 21).
- [3] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Capitolo 5, pagine 251-263 e 360-366. Addison-Wesley, Ottobre 1994.
- [4] Derrick Rountree. *Federated Identity Primer*. Syngress, Dicembre 2012 (cit. a p. 14).
- [5] Vincent C. Hu et al. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. Rapp. tecn. <http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf>. Gennaio 2014.
- [6] IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge v3.0*. Capitolo 4. IEEE, 2014.
- [7] Gregg Kreizman. *How to Choose Between On-Premises and IDaaS Delivery Models for Identity and Access Management*. <https://www.gartner.com/doc/3380723/choose-onpremises-idaas-delivery-models>. Luglio 2016 (cit. a p. 21).
- [8] N. Sakimura M. Jones J. Bradley. *JSON Web Token (JWT)*. Rapp. tecn. RFC7519 <https://tools.ietf.org/pdf/rfc7519.pdf>. Internet Engineering Task Force (IETF), Maggio 2015 (cit. alle pp. 72, 75).
- [9] Addy Osmani. *Learning JavaScript Design Patterns*. Capitolo 9, pagine 27-37. O'Reilly Media, Luglio 2012.
- [10] Neil Wynne Gregg Kreizman. *Magic Quadrant for Identity and Access Management as a Service, Worldwide*. <https://www.gartner.com/doc/3337824/magic-quadrant-identity-access-management>. Giugno 2016 (cit. a p. 34).
- [11] Ant Allan. *Managing Identities, Privileges, Access and Trust Primer for 2016*. <https://www.gartner.com/doc/3187123/managing-identities-privileges-access-trust>. Gennaio 2016 (cit. a p. 11).

- [13] Fernando Doglio. *Pro Rest Api Development With Node.js*. Apress, Maggio 2015 (cit. a p. 39).
- [14] Agenzia per l'Italia Digitale. *REGOLAMENTO RECANTE LE MODALITÀ ATTUATIVE PER LA REALIZZAZIONE DELLO SPID (articolo 4, comma 2, DPCM 24 ottobre 2014)*. http://www.agid.gov.it/sites/default/files/circolari/spid-modalita_attuative_v1.pdf. 2014 (cit. a p. 23).
- [15] Nick Ragouzis et al. *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. Rapp. tecn. Committee Draft 02 <http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html>. OASIS Security Services TC, mar. 2008 (cit. a p. 16).
- [16] Ant Allan. *Technology Overview for Adaptive Access Control*. <https://www.gartner.com/doc/2754020/technology-overview-adaptive-access-control>. Settembre 2015.
- [17] Steve Krapes. *Use IAM Life Cycle Policies to Enforce Account Disabling and Deletion*. <https://www.gartner.com/doc/2856523/use-iam-life-cycle-policies>. Settembre 2014.

Siti Web

- [18] *A che punto è SPID (Sistema pubblico dell'identità digitale) e a cosa serve*. URL: http://www.agendadigitale.eu/identita-digitale/a-che-punto-e-il-sistema-pubblico-dell-identita-digitale-e-a-che-serve_2079.htm.
- [19] *Bitbucket*. URL: <https://bitbucket.org/> (cit. a p. 43).
- [20] *Express.js*. URL: <http://expressjs.com/it/>.
- [21] *Gartner IT Glossary*. URL: <http://www.gartner.com/it-glossary/>.
- [22] *Git*. URL: <https://git-scm.com/> (cit. a p. 42).
- [23] *HMAC vs ECDSA for JWT*. URL: <http://crypto.stackexchange.com/questions/30657/hmac-vs-ecdsa-for-jwt>.
- [24] *IDP-Initiated vs. SP-Initiated Single Sign-On (SSO)*. URL: <http://bitwyze.net/idp-initiated-vs-sp-initiated-single-sign-on-sso/> (cit. a p. 80).
- [25] *Introduction to JSON Web Tokens*. URL: <https://jwt.io/introduction/> (cit. alle pp. 75, 78).
- [26] *Introduction to MongoDB*. URL: <https://docs.mongodb.com/v3.0/introduction/>.
- [27] *Manifesto for Agile Software Development*. URL: <http://agilemanifesto.org/iso/it/>.
- [28] Jeff Harrell. *Node.js at PayPal*. <https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/>. 2013 (cit. a p. 39).
- [29] *SPID*. URL: <http://www.spid.gov.it/>.

- [30] *SSO Benefits*. URL: <https://www.uoguelph.ca/ccs/security/internet/single-sign-sso/benefits>.
- [31] *Scrum Roles Demystified*. URL: <https://www.scrumalliance.org/employer-resources/scrum-roles-demystified>.
- [32] *Scrum Theory*. URL: <https://www.scrumalliance.org/why-scrum/scrum-guide>.
- [33] Issac Roth. *What Makes Node.js Faster Than Java?* <https://strongloop.com/strongblog/nodejs-is-faster-than-java/>. 2014.
- [34] *Using middleware*. URL: <http://expressjs.com/en/guide/using-middleware.html>.

