



**UNIVERSITÀ DEGLI STUDI DI MILANO**  
**FACOLTÀ DI SCIENZE E TECNOLOGIE**

Corso di Laurea in Informatica Musicale

**SISTEMA DI RACCOMANDAZIONE BASATO SU  
COLLABORATIVE FILTER PER PIATTAFORMA  
MOON CLOUD FACENTE PARTE DELL'AMBITO  
DELLA SECURITY ASSURANCE**

Relatore:  
Claudio Agostino Ardagna  
Correlatore:  
Valerio Bellandi

Tesi di Laurea di:  
Andrea Michele Albonico  
Matricola: 886667

Anno Accademico 2018/2019

# Ringraziamenti

*Andrea Michele Albonico*



# Prefazione

I sistemi di raccomandazione (*Recommendation System*) hanno avuto un forte sviluppo negli ultimi decenni e nascono proprio con lo scopo di identificare quegli oggetti (detti generalmente *item*) all'interno di un vasto mondo di informazioni che possono essere di nostro interesse e tanto maggiore è il grado di conoscenza dell'individuo e tanto più vengono ritenuti affidabili.

Il motivo di questo successo risiede nella riuscita integrazione di tali sistemi in applicazioni commerciali, soprattutto nel mondo dell'E-commerce e nel fatto che sono in grado di aiutare un utente a prendere una decisione, che sia la scelta di un film per l'uscita con gli amici il sabato sera, di una playlist da ascoltare durante un viaggio in auto o in un momento di lettura, e via scorrendo.

Moon Cloud è una piattaforma erogata come servizio che fornisce un meccanismo di *Security Governance* centralizzato. Garantisce il controllo della sicurezza informatica in modo semplice e intuitivo, attraverso attività di test e monitoraggio periodiche e programmate (*Security Assurance*). L'obiettivo di questa tesi è stato quello di aggiungere, al già presente sistema per la scelta dei controlli all'interno delle attività di test, un sistema di raccomandazione che possa consigliare all'utente delle possibili *Evaluation* rispetto al target indicato; in questo modo anche l'utente meno esperto può usufruire dei servizi offerti da Moon Cloud in modo semplice e intuitivo.

La tesi è organizzata come segue:

**Capitolo 1 – Introduzione a Moon Cloud** in questo capitolo viene descritta la piattaforma Moon Cloud e il suo funzionamento in ambito di Security Assurance.

**Capitolo 2 – Tecnologie utilizzate** in questo capitolo vengono presentati gli studi e le analisi di soluzioni esistenti, studi delle tecnologie utilizzate per la realizzazione del progetto.

**Capitolo 3 – Collaborative filtering** in questo capitolo viene descritto in modo più approfondito gli studi compiuti sui Filtri Collaborativi

che hanno portato alla realizzazione dei sistemi di raccomandazione proposti nella soluzione implementata per la piattaforma Moon Cloud, inoltre verranno mostrate le relative porzioni di codice.

**Capitolo 4 – Descrizione della soluzione** in questo capitolo viene descritta in maniera dettagliata la realizzazione dell'applicativo, analizzando quali sono state le difficoltà maggiori, i risultati ottenuti e l'uso che se ne è fatto.

**Capitolo 5 – Conclusioni** in questo capitolo vengono esposte le conclusioni e i possibili sviluppi futuri delle attività svolte e del sistema realizzato.

# Indice

<b>Prefazione</b>	<b>v</b>
<b>1 Introduzione a Moon Cloud</b>	<b>1</b>
1.1 Moon Cloud overview . . . . .	1
1.2 Processo di Evaluation . . . . .	5
<b>2 Tecnologie utilizzate</b>	<b>7</b>
2.1 Perché Python e Django . . . . .	7
2.2 Docker . . . . .	11
2.3 Strutture dati gerarchiche . . . . .	12
2.3.1 The Adjacency List Model . . . . .	13
2.3.2 The Nested Set Model . . . . .	14
2.4 Sistemi di raccomandazione . . . . .	17
2.4.1 Content-based filtering . . . . .	19
2.4.2 Collaborative filtering . . . . .	19
2.4.3 Cold Start problem . . . . .	21
<b>3 Collaborative filtering</b>	<b>23</b>
3.1 Memory-based . . . . .	23
3.1.1 User-based filtering . . . . .	23
3.1.2 Item-based filtering . . . . .	27
3.1.3 Hybrid filtering . . . . .	29
<b>4 Descrizione della soluzione</b>	<b>33</b>
<b>5 Conclusioni</b>	<b>47</b>
<b>Bibliografia</b>	<b>49</b>





# Elenco delle figure

1.1	Security Compliance Evaluation . . . . .	4
2.1	Schema generico di funzionamento di un applicativo web sviluppato con Django . . . . .	10
2.2	Schematizzazione del contenuto di un Container in Docker . . . . .	11
2.3	Esempio della rappresentazione gerarchica parziale dei dati nel progetto in questione . . . . .	12
2.4	Esempio della gestione di dati in modo gerarchico secondo il Nested Set Model, utilizzando dati presi dal database del progetto in questione . . . . .	14
2.5	Esempio della gestione di dati in modo gerarchico secondo il Nested Set Model, utilizzando dati presi dal database del progetto in questione . . . . .	15
2.6	Categorizzazione generale dei sistemi di raccomandazione . . . . .	17
3.1	Esempio di applicazione di un sistema di raccomandazione User-based . . . . .	24
3.2	Esempio di applicazione di un sistema di raccomandazione IB-CF. . . . .	27
3.3	Esempio di esempio di una risposta in JSON a una chiamata Rest all'algoritmo di raccomandazione IB-CF per le Evaluation supportate da Moon Cloud. . . . .	29
3.4	Esempio di esempio di una risposta in JSON a una chiamata Rest all'algoritmo di raccomandazione IB-CF per i Target supportati da Moon Cloud. . . . .	30
3.5	Esempio di esempio di una risposta in JSON a una chiamata Rest all'algoritmo di raccomandazione Ibrido. . . . .	32
4.1	Struttura del database . . . . .	37
4.2	Home page . . . . .	38
4.3	Home page per la navigazione della tassonomia . . . . .	39

4.4	Dettagli della tassonomia sottoforma di tabella come nella base di dati . . . . .	39
4.5	Risultato dell'operazione selezionata sul nodo in questione . .	39
4.6	Admin page . . . . .	45
4.7	Esempio di Admin page per le Evaluation . . . . .	45
4.8	Esempio di Admin page per il caso in cui si vuole aggiungere una nuova Evaluation . . . . .	46

# Elenco delle tabelle

2.1	Esempio di una possibile tabella per gestire dati in modo gerarchico secondo l'Adjacency List Model . . . . .	13
2.2	Esempio di una tabella per gestire dati in modo gerarchico secondo il Nested Set Model . . . . .	15



# Capitolo 1

## Introduzione a Moon Cloud

In questo capitolo verrà descritto in modo più approfondito il funzionamento della piattaforma Moon Cloud e unitamente al motivo dell'implementazione della soluzione proposta.

### 1.1 Moon Cloud overview

La diffusione di sistemi *Information and Communications Technology* (acronimo ICT) ha avuto luogo nella maggiorparte degli ambienti lavorativi e privati in termini di servizi offerti, automazione di processi e incremento delle performance. L'uso di questa tecnologia ha assunto importanza a partire dagli anni novanta come effetto del boom di Internet e al giorno d'oggi le professionalità legate al mondo dell'ICT crescono in numero e si evolvono per specificità, per operare in ambienti fortemente eterogenei ma sempre più interconnessi fra di loro come il Cloud Computing, i Social Newtwork, il Marketing Digitale, i Sistemi IoT, la Realtà Virtuale, etc.

Il Cloud Computing ha portato un rivoluzionario paradigma nella creazione di un nuovo business virtuale accessibile in qualunque momento e in qualunque luogo; esso sfrutta le tecnologie messe a disposizione dai sistemi ICT come le operazioni di virtualized computing, internet e distributed computing, provvedendo un sistema integrato molto potente. Google, Microsoft, Amazon sono un esempio di aziende che forniscono servizi di Cloud Computing in business ICT. Si può definire il Cloud Computing come l'abilità di accedere a risorse (come database o applicazioni) in tutto il mondo attraverso una rete in poco tempo.

Gli immensi benefici del Cloud in termini di flessibilità, consumo delle risorse e gestione semplificata, la rende la prima scelta per utenti e industrie per il deploy dei loro sistemi IT. Tuttavia il Cloud Computing solleva diverse

problematiche legate alla mancanza di fiducia e trasparenza dove i clienti necessitano di avere delle garanzie sui servizi Cloud ai quali si affidano; spesso i fornitori di questi servizi non forniscono ai clienti le specifiche riguardanti le misure di sicurezza messe in atto.

Negli ultimi anni, sono state sviluppate tecniche e modi per rendere sicuri questi sistemi e proteggere i dati degli utenti, portando alla diffusione di approcci eterogenei che incrementano la confusione negli utenti. Tecniche tradizionali di verifica della sicurezza basati su metodi di analisi statistica non sono più sufficienti e devono essere integrati con processi di raccolta di prove (in inglese *evidence*) da sistemi Cloud in produzione e funzionanti. In generale il *Cloud Security* definisce i modi, come crittazione e controllo degli accessi, per proteggere attivamente gli asset da minacce interne ed esterne, e fornire un ambiente in cui i clienti possano affidarsi e interagire in totale sicurezza. Tutto questo non basta a rendere il Cloud degno di fiducia e trasparente, per questo sono state introdotte tecniche di *Security Assurance*, delle garanzie che permettono di ottenere la fiducia necessaria nelle infrastrutture e/o nelle applicazioni di dimostrare il rispetto di certe proprietà di sicurezza, e che operino normalmente anche se subiscono attacchi; grazie alla raccolta e allo studio di *evidence* è possibile che venga accertata la validità e l'efficienza delle proprietà di sicurezza messe in atto.

Il prezzo che si paga per i benefici di questa tecnologia è dato dall'incremento di violazioni di sicurezza, che oggi preoccupa tutte le aziende e di conseguenza anche i loro clienti, con l'incremento del rischio di fallimento per i servizi più importanti dovuti a violazioni della privacy e al furto di dati.

Il mercato sta lentamente notando che non è l'inadeguamento tecnologico dei sistemi di sicurezza che incrementa il rischio delle violazioni di sicurezza; piuttosto, la mal configurazione e l'errata integrazione di questi sistemi nei processi di business. [2] L'utilizzo di sistemi di sicurezza e di controllo migliori non garantisce in modo assoluto la sicurezza dell'infrastruttura; per garantire ciò è necessario implementare un processo continuo di diagnostica e verifica della corretta configurazione dei controlli, supervisionando il loro comportamento, accertandosi che sia quello aspettato.

Il *Security Assessment* diventa allora un aspetto importante specialmente negli ambienti Cloud e IoT. Questo processo, costituito da un insieme di attività mirate alla valutazione del rischio in sistemi IT, deve essere portato avanti in modo continuo e olistico, per correlare le *evidence* raccolte da sempre maggiori meccanismi di protezione. [1]

Moon Cloud è una soluzione PaaS (acronimo inglese di Platform as a Service) che fornisce una piattaforma B2B (Business To Business) innovativa per verifiche, diagnostiche e monitoraggio dell'adeguatezza dei sistemi ICT rispetto

alle politiche di sicurezza, in modo continuo e su larga scala. Moon Cloud supporta una semplice ed efficiente *ICT Security Governance*, dove le politiche di sicurezza possono essere definite dalle compagnie stesse (a partire da un semplice controllo sulle vulnerabilità a linee guida di sicurezza interna), da entità esterne, imposte da standard oppure da regolamentazioni nazionali o internazionali.

La sicurezza di un sistema o di un insieme di asset dipende solo parzialmente dalla forza dei singoli meccanismi di protezione isolati l'uno dall'altro; infatti, dipende anche dall'abilità di questi meccanismi di lavorare continuamente in sinergia per provvedere una protezione olistica. In più, quando i sistemi Cloud e i servizi IoT sono coinvolti, le dinamiche di questi servizi e la loro rapida evoluzione rende il controllo dei processi all'interno dell'azienda e le politiche di sicurezza più complesse e prone ad errori.

I requisiti ad alto livello fondamentali per poter garantire le Security Assurance sono:

**sistema olistico** è richiesta una visione globale e pulita dello status dei sistemi di sicurezza; inoltre è cruciale distribuire lo sforzo degli specialisti in sicurezza per migliorare il processo e le politiche messe in atto. Si parte da delle valutazioni fatte manualmente a quella semi-automatiche che vengono usate per ispezionare i meccanismi di sicurezza.

**monitoraggio continuo ed efficiente** è necessario un controllo continuo che valuti l'efficienza dei sistemi di sicurezza per ridurre l'impatto dell'errore umano, soprattutto dal punto di vista organizzativo. La coesistenza di componenti in conflitto o la mancata configurazione dovuta al cambiamento dell'ambiente possono essere scenari che richiedono un monitoraggio e un aggiornamento continuo.

**singolo punto di management** avere un solo punto d'accesso in cui poter gestire tutti gli aspetti relativi alla sicurezza, permette di avere sotto controllo le politiche di sicurezza. Inoltre disporre di un inventario degli asset da proteggere permette di poter conoscere quali meccanismi di protezioni applicare.

**reazioni rapide a incidenti di sicurezza** spesso la reazione ad incidenti di sicurezza è ritardata da due fattori: il tempo richiesto per rilevare l'incidente e il tempo per analizzare il motivo dell'accaduto; e avere un sistema che implementa un monitoraggio continuo permette di venire a conoscenza di questi problemi in breve tempo e agire di conseguenza.

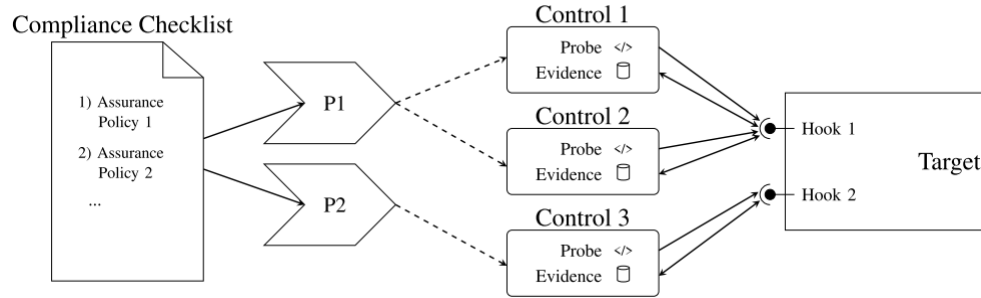


Figura 1.1: Security Compliance Evaluation

Moon Cloud è basato su una tecnica di Security Assurance garantendo che tutte le attività aziendali si compiano seguendo i requisiti prestabiliti da appropriate politiche e procedure precedentemente definite.

Viene definita una *Security Compliance Evaluation*, un processo di verifica a cui un target viene sottoposto e il cui risultato deve soddisfare i requisiti richiesti da standard e politiche. A partire da questi processi di controllo, che devono a loro volta essere affidabili, si ottengono delle evidenze; queste ultime possono essere raccolte monitorando l'attività del target oppure, come già menzionato, sottoponendo il target a scenari critici o di testing. In particolare, una Security Compliance Evaluation è un processo di verifica dell'uniformità di un certo target a una o più politiche attraverso una serie di controlli che a seconda del valore booleano (successo pari a 1 o insuccesso pari a 0) associato ad ogni controllo viene prodotto un valore booleano per le politiche; se un target supera tutti i controlli, quindi tutti i valori booleani associati ad ogni controllo è pari al successo, a cui è sottoposto allora significa che rispetta la politica scelta.



## 1.2 Processo di Evaluation

Moon Cloud implementa il processo di Security Compliance Evaluation in Figura 1.1 usando controlli di monitoraggio o di test personalizzabili. Inoltre garantisce, oltre a tutti i requisiti ad alto livello elencati precedentemente 1.1, anche i seguenti:

Moon Cloud è una piattaforma Cloud centralizzata presentando una visione olistica dello stato di sicurezza di un dato sistema.

Moon Cloud implementa un sistema di Security Assurance Evidence-based continuo, implementato come processo di Compliance, basato su politiche custom o standard.

Moon Cloud è offerto come un servizio (PaaS), dove le attività di Evaluation possono essere facilmente ed efficientemente configurate su un target asset, senza l'intervento dell'uomo.

Moon Cloud permette di schedulare delle ispezioni automatiche, grazie all'inventario di asset protetto.

Moon Cloud Evaluation Engine può ispezionare dall'interno un sistema, gestendo così delle minacce interne; permettendo anche reazioni rapide a incidenti di sicurezza e veloci rimedi, grazie alla raccolta continua di evidence.

In generale, l'architettura di Moon Cloud è costituita da un'Assurance Manager che gestisce i processi di Evaluation attraverso un set di *Execution Cluster*; ognuno dei quali gestisce ed esegue un set di probe che collezionano le evidence necessarie per effettuare i processi di valutazione. Tutte le attività di collezione sono eseguite dal probe, ognuno dei quali è uno script Python fornito come una singola immagine di Docker, che viene inizializzata quando viene triggerata una Evaluation ed è distrutta quando il processo di Evaluation è terminato.

Accedendo alla piattaforma di Moon Cloud, l'utente può definire le proprie politiche di sicurezza e attività di Evaluation come espressioni booleane di controlli di sicurezza e altre politiche predefinite. Una volta che una politica viene definita, l'utente può decidere quando schedulare l'Evaluation; e nel momento in cui un processo di Evaluation viene inizializzato, tutti i controlli vengono eseguiti e i risultati dell'espressioni booleane vengono memorizzati e restituiti all'utente. A questo punto l'utente può accedere a questi risultati a diversi gradi di precisione: una visione sommaria e generale di tutte le politiche implementate e dello stato generale del sistema di sicurezza, al risultato di una specifica politica oppure alle evidence raccolte per una Evaluation.

Per poter rendere ancora più intuitivo e semplice da utilizzare un sistema di questa importanza, si è pensato di introdurre un sistema che possa raccomandare agli utenti, in base agli asset che vuole proteggere e monitorare, una serie di Evaluation o politiche da applicare in quei casi; questo permette anche a utenti meno esperti di poter configurare in modo rapido ed efficiente dei meccanismi di protezione da minacce.

# Capitolo 2

## Tecnologie utilizzate

In questo capitolo verranno descritte le attività preliminari per la realizzazione di questo progetto, le tecnologie utilizzate unitamente alle motivazioni legate all'uso di questi sistemi rispetto ad altri. In particolare nel capitolo successivo verranno approfonditi a livello pratico i sistemi di raccomandazione Memory-based i quali sono stati utilizzati per l'implementazione della soluzione.

### 2.1 Perché Python e Django

**Python** Python è un linguaggio di programmazione ad alto livello, orientato agli oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing; ideato e rilasciato pubblicamente per la prima volta nel 1991 dal suo creatore Guido van Rossum, programmatore olandese.

Python supporta diversi paradigmi di programmazione, come quello object-oriented (con supporto all'ereditarietà multipla), quello imperativo e quello funzionale, ed offre una tipizzazione dinamica forte. È fornito di una libreria built-in estremamente ricca, che unitamente alla gestione automatica della memoria e a robusti costrutti per la gestione delle eccezioni fa di Python uno dei linguaggi più ricchi e comodi da usare.

Inoltre è anche semplice da usare e imparare. Python, nelle intenzioni di Guido van Rossum, è nato per essere un linguaggio immediatamente intuibile. La sua sintassi è pulita e snella così come i suoi costrutti, decisamente chiari e non ambigui. I blocchi logici vengono costruiti semplicemente allineando le righe allo stesso modo, incrementando la leggibilità e l'uniformità del codice anche se vi lavorano diversi autori.

Un aspetto inusuale del Python è il metodo che usa per delimitare i blocchi di programma, che lo rende unico fra i linguaggi più diffusi.

```
1 # Testing if two strings are equals
2 def test(got, expected):
3     if got == expected:
4         prefix = ' OK '
5     else:
6         prefix = ' X '
7     print('%s got: %s expected: %s' % (prefix, repr(got), repr(
8         expected)))
9
10 def main():
11     print('verbing')
12     test('hail', 'hailing')
13     test('swiming', 'swimmingly')
14     test('do', 'do')
15
16 if __name__ == '__main__':
17     main()
```

Listing 2.1: Esempio di programma scritto in Python

Nei linguaggi derivati dall'ALGOL come Pascal, C e Perl, i blocchi di codice sono indicati con le parentesi oppure con parole chiave (il C ed il Perl usano `{}`; il Pascal usa `begin` ed `end`). In questi linguaggi è solo una convenzione degli sviluppatori il fatto di indentare il codice interno ad un blocco, per metterlo in evidenza rispetto al codice circostante. In Python invece di usare parentesi o parole chiave, usa l'indentazione stessa per indicare i blocchi nidificati in congiunzione col carattere "due punti" (`:`). Si può usare sia una tabulazione, sia un numero arbitrario di spazi, ma lo standard Python è di 4 spazi.

Python è un linguaggio pseudocompilato: un interprete si occupa di analizzare il codice sorgente (semplici file testuali con estensione `.py`) e, se sintatticamente corretto, di eseguirlo e non esiste una fase di compilazione separata (come avviene in C, per esempio) che generi un file eseguibile partendo dal sorgente. [7]

**Django** Django è un web framework di alto livello basato su Python che permette di sviluppare rapidamente e con tutti i presupposti per un sistema sicuro, un sito web perfettamente mantenibile, Django si occupa della maggiori grane del sviluppo web, così da permetterti di concentrarti sulla scrittura della tua app; inoltre è open-source e ha una comunità attiva, una documentazione completa e semplice da consultare.

Django aiuta a scrivere software con le seguenti caratteristiche [3]:

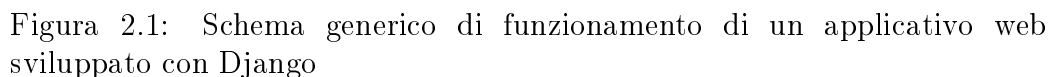
**Versatile** : è usato per la creazione di praticamente tutti i tipi di siti web, a partire a sistemi per la gestione di contenuti e wiki, social network e siti di notizie; può lavorare con qualunque client-side framework, gestire contenuti in quasi tutti i formati (inclusi HTML, RSS feeds, JSON, XML, etc). Internamente permette la scelta e l'implementazione di qualsiasi funzionalità (es. molti dei database più popolari, etc.).

**Sicuro** : aiuta gli sviluppatori a evitare gli errori più comuni in merito alla sicurezza provvedendo un framework costruito per eseguire le operazioni in modo corretto e sicuro. Ad esempio, Django fornisce un modo sicuro per gestire gli account degli utenti e le relative password, evitando errori comuni come inserire informazioni riguardanti la sessione dell'utente nei cookies dove sarebbero vulnerabili (invece i cookie contengono soltanto una chiave, e i valori effettivi sono salvati nel database) o salvare direttamente una password invece di una hash password.

**Mantenibile** : il codice di Django è scritto seguendo i principi e i pattern che incoraggiano la creazione di codice mantenibile e riusabile. Inoltre particolare, fa uso del principio "Don't Repeat Yourself" (DRY) così da ridurre al minimo le duplicazioni non necessarie, diminuendo la quantità di codice. Django raggruppa parti di codice letto in moduli seguendo le linee guida del pattern Model View Controller (MVC).

**Portatile** : Django essendo scritto in Python, un linguaggio multi piattaforma, lo rende indipendente dal sistema operativo eseguito sul server, che sia Linux, Windows o Mac OS X. Per di più, Django è ben supportato da molti web hosting provider, che spesso provvedono a specifiche infrastrutture e documentazione per l'hosting di siti web in Django.

Un tradizionale sito web attende delle richieste HTTP dal browser web (o altri client). Quando viene ricevuta una richiesta, di tipo POST o GET, l'applicazione legge le informazioni contenute nel URL e altri possibili dati a seconda del tipo di richiesta. A seconda della richiesta è possibile che vengano letti o scritti dati da un database o altre operazioni che portino al soddisfacimento della richiesta. A quel punto l'applicazione ritorna una risposta al browser web, spesso in modo dinamico, creando una pagina HTML da mostrare in cui è possibile inserire o recuperare dati in placeholder in un template HTML.



**URL** : mentre è possibile processare richieste da qualsiasi URL attraverso una singola funzione, è più mantenibile scrivere diverse funzioni chiamate View per gestire ogni risorsa. un URL mapper è usato per reindirizzare le richieste HTTP alla view corretta in base all'URL della richiesta; inoltre è possibile controllare se nell'URL è presente un particolare pattern di stringhe o numeri, e passare di conseguenza la richiesta alla funzione appropriata come dati da elaborare.

**Model** : i Model sono oggetti in Python che definiscono la struttura dei dati dell'applicazione, e provvedono meccanismi per gestirla (add, modify, delete) e query per interpellare il database.

**Template** : un template è un file di testo che definisce la struttura o il layout di un file (come una pagina HTML), attraverso placeholder per rappre-

sentare il contenuto effettivo. Una View può creare dinamicamente una pagina HTML usando un template HTML, popolandolo con dati presi dal Model.

## 2.2 Docker

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Docker raccoglie il software in unità standardizzate chiamate *container* che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito.

La tecnologia Docker utilizza solitamente il kernel di Linux e le sue funzionalità, come Cgroups e namespace, per isolare i processi in modo da poterli eseguire in maniera indipendente. Questa indipendenza è l'obiettivo dei container: la capacità di eseguire più processi e applicazioni in modo separato per sfruttare al meglio l'infrastruttura esistente pur conservando il livello di sicurezza che sarebbe garantito dalla presenza di sistemi separati.

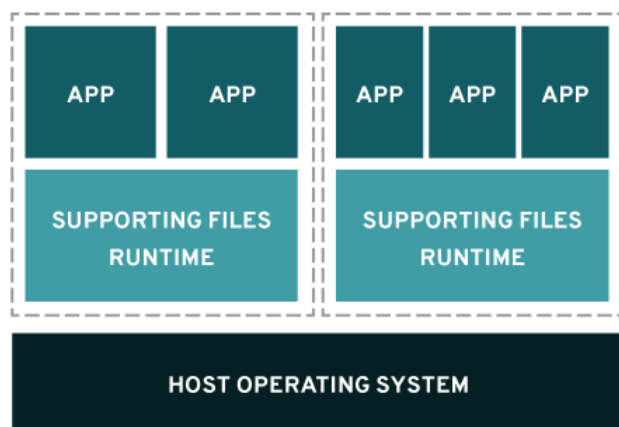


Figura 2.2: Schematizzazione del contenuto di un Container in Docker

Gli strumenti per la creazione di container, come Docker, consentono il deployment a partire da un'*immagine*, ciò semplifica la condivisione di un'applicazione o di un insieme di servizi, con tutte le loro dipendenze, nei vari ambienti. Docker, considera i container come macchine virtuali modulari estremamente leggere, offrendo la flessibilità di creare, distribuire, copiare e spostare i container da un ambiente all'altro, ottimizzando così le app per il cloud.

I contenitori forniscono una modalità standard per impacchettare il codice della tua applicazione, le configurazioni e le dipendenze, in un oggetto singolo e condividono un sistema operativo installato sul server, operando come processi con risorse isolate, assicurando velocità, affidabilità e distribuzioni coerenti, indipendentemente dall'ambiente.

## 2.3 Strutture dati gerarchiche

Le tabelle di un database relazionale non sono gerarchiche (come nel XML), ma sono delle semplici liste piatte. I dati gerarchici sono costituiti da relazioni padre-figlio che non possono essere rappresentate in modo naturale nelle tabelle di questo tipo. In questo caso, i dati gerarchici sono una collezione di informazioni dove ogni item ha un solo padre e nessuno o più figli (ad eccezione del nodo radice che non ha un nodo padre); questo genere di rappresentazione delle informazioni, simili a un albero, può essere trovato in diversi ambiti di applicazione di un database, incluse discussioni su forum e mailing list, grafici di organizzazione di un business, categorie per gestire contenuti e categorie di prodotti.

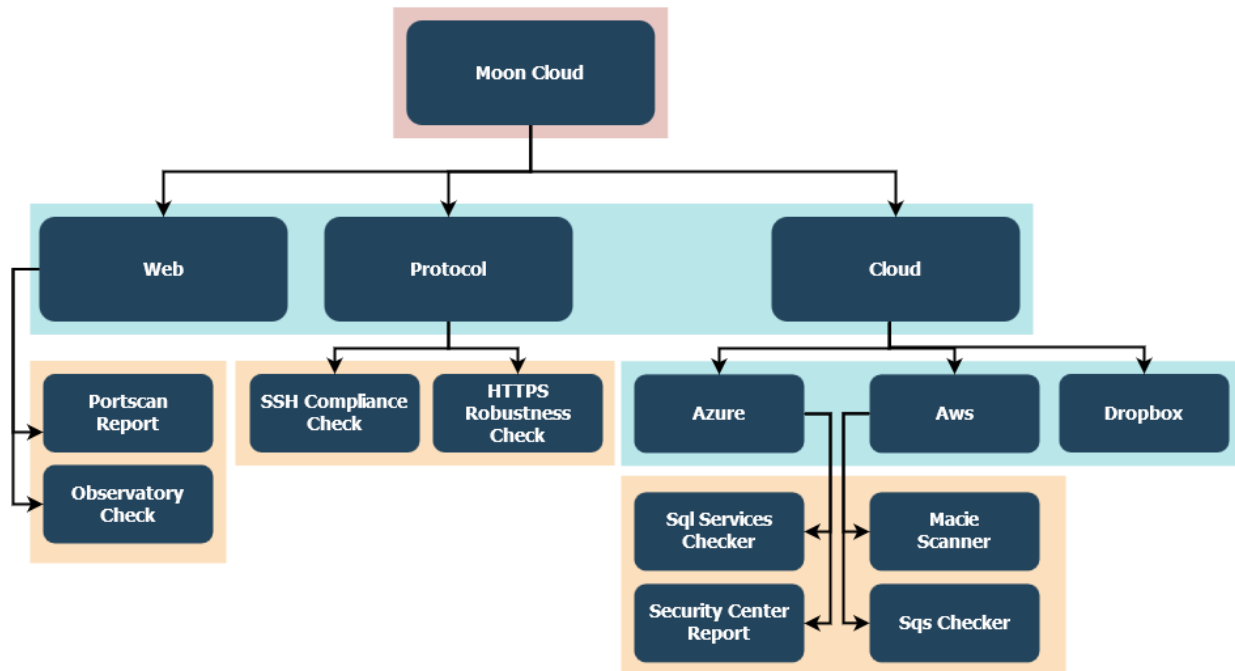


Figura 2.3: Esempio della rappresentazione gerarchica parziale dei dati nel progetto in questione



Ci sono differenti modelli per poter gestire dati in modo gerarchico, i più importanti presi in considerazione sono i seguenti:

### 2.3.1 The Adjacency List Model

Il primo approccio, e quello di più semplice implementazione, qui descritto è chiamato *Adjacency List Model* o metodo ricorsivo; è definito tale perchè il suo funzionamento si basa su una funzione che itera per tutto l'albero. In questo modello, ogni item (nodo dell'albero) nella tabella contiene un puntatore al suo item padre; invece il nodo radice avrà un puntatore a un valore NULL per l'item padre.

La Tabella 2.1 è un esempio di possibile rappresentazione parziale dei dati nel database implementato in questo progetto secondo questo approccio, seguendo come riferimento la Figura 2.3.

id	name	parent
1	Moon Cloud	NULL
2	Web	1
3	Protocol	1
4	Cloud	1
5	Portscan Report	2
6	Observatory Check	2
7	SSH Compliance Check	3
8	HTTPS Robustness Check	3
9	Azure	4
10	Aws	4
11	Dropbox	4
12	Sql Services Checker	9
13	Security Center Report	9
14	Macie Scanner	10
15	Sqs Checker	10

Tabella 2.1: Esempio di una possibile tabella per gestire dati in modo gerarchico secondo l'Adjacency List Model

Il vantaggio di usare questo modello sta nella sua semplicità di costruzione, soprattutto a livello di codice client-side, e di restituzione dei figli di un nodo. Mentre diventa problematico se si lavora in puro codice SQL e nella maggior parte dei linguaggi di programmazione, è lento e poco efficiente,

perchè è necessaria una query per ogni nodo dell'albero, e visto che ogni query impiega un certo periodo di tempo, questo rende la funzione molto lente quando si lavora con alberi di grandi dimensioni, inoltre molti linguaggi non sono ottimizzati per funzioni ricorsive. Per ogni nodo, la funzione crea una nuova istanza di se stessa e ogni istanza occupa una porzione di memoria e impiega un certo tempo per inicializzarsi, più grande è l'albero e più questo processo sarà portato a termine in maggior tempo.

### 2.3.2 The Nested Set Model

Il secondo approccio analizzato è il *Nested Set Model*, che permette di osservare la gerarchia in un modo diverso, non come nodi e linee (come se fosse un albero), ma come container innestati.

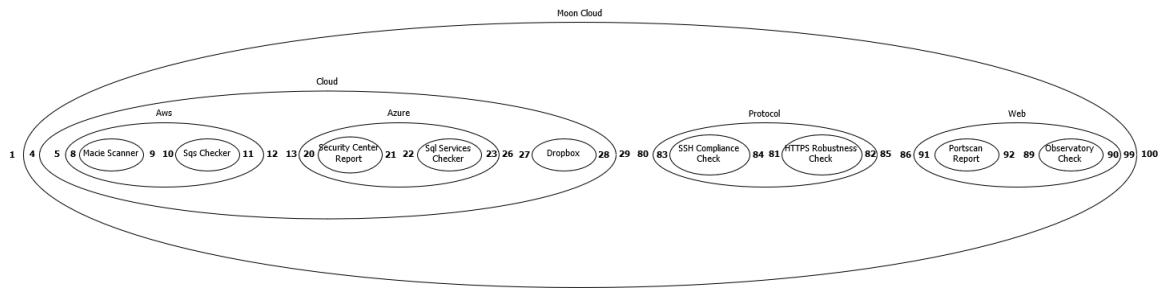


Figura 2.4: Esempio della gestione di dati in modo gerarchico secondo il Nested Set Model, utilizzando dati presi dal database del progetto in questione

Con questo sistema la gerarchia viene mantenuta, secondo il principio cui un nodo padre contiene e suoi figli e questa forma di gerarchia viene mantenuta in tabella attraverso l'uso di due attributi aggiuntivi come è possibile osservare dalla Tabella 2.2 seguente.

Come è possibile osservare dalla Tabella 2.2 la gerarchia dei dati viene rappresentata attraverso l'uso degli attributi 'left' e 'right' per rappresentare l'annidamento dei nodi (il nome delle colonne: 'left' e 'right', hanno significati speciali in SQL; per questo motivo si identificano questi campi con i nomi 'lft' e 'rft').

L'assegnazione di questi valori ad ogni nodo viene effettuata seguendo questo procedimento: ogni nodo dell'albero viene visitato due volte, assegnando i valori in ordine di visita, e in entrambe le visite. Quindi vengono associati ad ogni nodo due numeri, memorizzati come due attributi. Più precisamente

id	name	lft	rght
1	Moon Cloud	1	100
2	Web	86	99
3	Protocol	80	85
4	Cloud	4	29
5	Portscan Report	91	92
6	Observatory Check	89	90
7	SSH Compliance Check	83	84
8	HTTPS Robustness Check	81	82
9	Azure	13	26
10	Aws	5	12
11	Dropbox	27	28
12	Sql Services Checker	22	23
13	Security Center Report	20	21
14	Macie Scanner	8	9
15	Sqs Checker	10	11

Tabella 2.2: Esempio di una tabella per gestire dati in modo gerarchico secondo il Nested Set Model

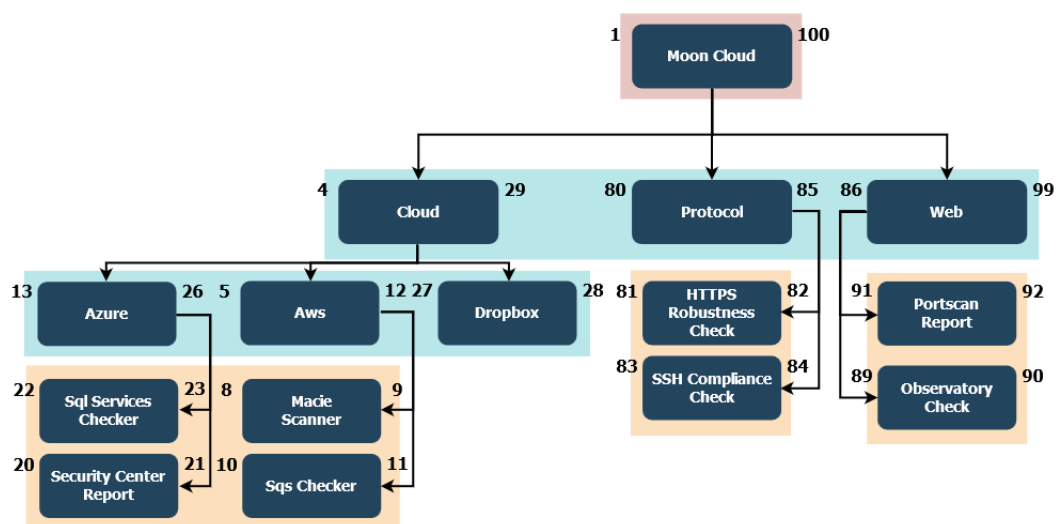


Figura 2.5: Esempio della gestione di dati in modo gerarchico secondo il Nested Set Model, utilizzando dati presi dal database del progetto in questione

si inizia la visita dell'albero partendo da sinistra e continuando verso destra, un livello alla volta, scendendo per ogni nodo i suoi figli, assegnando i valori al campo left, prima di assegnare un valore al campo right, e successivamente si continua verso destra. Questo approccio è chiamato *Modified preorder tree traversal algorithm* (MPTT). A partire da questa tecnica è stata ideata la struttura della tassonomia delle evaluation e dei controlli implementate nella soluzione proposta in questa tesi, con l'ausilio di un package di Python chiamato MPTT.

Più semplicemente se si osserva la parte superiore della Figura 2.4 possiamo notare che la numerazione dei nodi, viene effettuata a partire da container più esterno da sinistra e continua verso destra.

A prima vista questo approccio può sembrare più complicato da comprendere rispetto all'Adjacency List Model, ma quest'ultimo è molto più veloce quando si vuole recuperare i nodi, visto che basta una query, mentre è più lento per operazioni di aggiornamento e cancellazione dei nodi; in quest'ultimo il grado di complicatezza dell'operazione è determinato dal nodo che si vuole cancellare, a partire dal caso più semplice, il nodo foglia (un nodo senza figli) fino al caso più complicato, quando si vuole cancellare il nodo radice.

## 2.4 Sistemi di raccomandazione

Un sistema di raccomandazione (*Recommendation System*) è un sistema che consiglia ad utenti uno o più item esistenti in un database. L'*item* è un qualsiasi cosa di interesse agli utenti, come prodotti, libri o giornali. Quando si eseguire una raccomandazione si hanno delle aspettative che l'item raccomandato possa essere tra quelli di maggiore interesse; in altre parole, devono essere in accordo con i gusti degli utenti.

Oggigiorno si possono trovare due principali trend di sistemi di raccomandazione:

**Content-based filtering** (CBF): un item viene raccomandato ad un utente se esso è simile ad altri item di interesse o piaciuti in passato, vengo presi in considerazione prima gli item con alte valutazioni o quelli molto utilizzati. Ogni item ha associate delle informazioni che lo descrivono, questo insieme di dati viene definito metadati, e sono fondamentali per il processo di raccomandazione.

**Collaborative filtering** (CF): un item viene raccomandato ad un utente se i suoi vicini (altri utenti simili) sono interessati a quello stesso item.

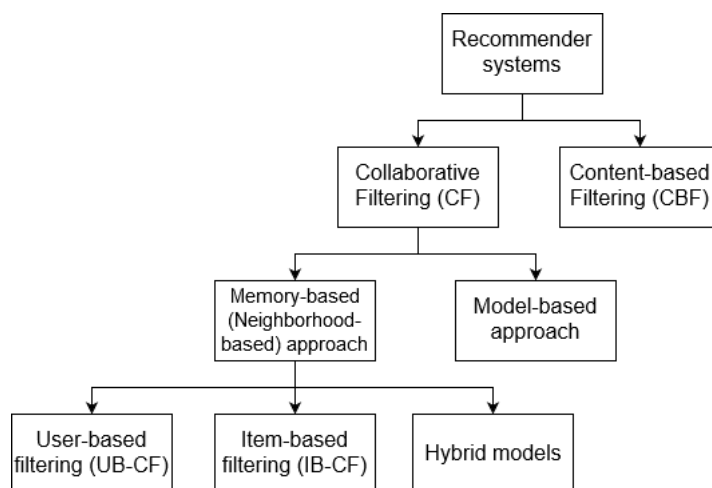


Figura 2.6: Categorizzazione generale dei sistemi di raccomandazione

Entrambi gli approcci (CBF e CF) hanno i loro punti di forza e di debolezza. Il primo algoritmo si focalizza sul contenuto degli item e sugli interessi del singolo utente e propone item differenti a utenti differenti, questo significa che ogni utente può ricevere raccomandazioni uniche; e questo è un

vantaggio. Tuttavia la più grande limitazione del CBF è il fatto di non poter determinare se un utente è interessato ad un item in modo implicito, perchè analizza solamente direttamente i metadati del prodotto e non considera gli interessi di altri utenti, i quali potrebbero suggerire item che non verrebbero notati con questo approccio. Per quanto riguarda il CF, nel caso siano presenti molti contenuti e proprietà associati agli item allora il sistema CF consuma molte risorse e tempo per poter analizzarli, nel contempo a questo algoritmo non interessano queste informazioni. Una raccomandazione viene fatta sulla base delle valutazioni degli utenti per gli item, o sugli usi che gli utenti fanno degli item e questo è il suo punto di forza perchè non si trova a dover analizzare item ricchi di informazioni. Allo stesso tempo è anche il suo punto debole, perchè può portare a suggerimenti che potrebbero essere considerati poco adatti sulla base della poca relazione con i profili di alcuni utenti. Questo problema è accentuato quando sono presenti nel database molti item che non hanno valutazioni o non sono stati mai usati dagli utenti. [4]

Un sistema di raccomandazione filtra i dati usando differenti algoritmi e raccomanda gli item più rilevanti agli utenti, attraverso un procedimento a 3 fasi:

**Raccolta di dati** : questo è il primo step e anche quello più importante per poter costruire un sistema di raccomandazione che produca risultati rilevanti e consistenti. I dati possono essere raccolti in due modi: esplicitamente, cioè attraverso la raccolta diretta di informazioni fornite dagli utenti, ad esempio le valutazioni di un prodotto; mentre attraverso l'approccio implicito, vengono raccolti dati che non sono prodotti in modo intenzionale dall'utente ma ottenuti dai costanti flussi di dati come la cronologia di ricerca, i click effettuati, lo storico degli ordini, etc.

**Memorizzazione di dati** : la quantità di dati definisce quanto efficace un modello di raccomandazione possa diventare. Ad esempio, in un sistema di raccomandazione per film, maggiori sono le valutazioni fornite dagli utenti, e migliore sarà il sistema di raccomandazione per gli altri utenti. Il tipo di dati che si vuole raccogliere determina anche il supporto di memorizzazione più adatto.

**Filtraggio dei dati** : dopo la fase di raccolta e memorizzazione dei dati, essi vanno filtrati per poter estrarre le informazioni rilevanti per poter effettuare le raccomandazioni finali, e sono già disponibili diversi algoritmi che semplificano quest'ultima fase.

I sistemi di raccomandazione possono essere suddivisi nelle seguenti categorie, ma spesso si preferisce degli approcci ibridi, delle combinazioni di sistemi di raccomandazione basati sul Contenuto (*Content-based filtering*) e di quelli Collaborativi (*Collaborative filtering*) in modo da essere più efficaci e sfruttare i pregi di entrambi gli approcci.

### 2.4.1 Content-based filtering

Un Content-based filtering (acronimo CBF) è un sistema di raccomandazione in cui vengono suggeriti, rispetto ad un item (oggetti o prodotti), quelli più simili, il confronto viene effettuato sfruttando i metadati, come il genere, una descrizione, uno o più autori, la categoria di appartenenza, etc.; l'idea base che si trova dietro questi sistemi, si basa sul fatto che se ad un utente piace o interessa un particolare item allora gli piaceranno anche altri con caratteristiche o proprietà simili.

Questo algoritmo suggerisce prodotti che piacevano all'utente nel passato ed è limitato a item dello stesso tipo. Un Content-based recommender fa riferimento a quegli approcci, che provvedono raccomandazioni comparando la rappresentazione del contenuto che descrive un item e la rappresentazione del contenuto dell'item interessato dall'utente.

Questi metodi sono usati quando si conoscono a priori i metadati sugli item che si vuole suggerire, ma nulla sugli utenti. In questo sistema, delle *keyword* sono utilizzate per caratterizzare gli item e un profilo dell'utente è costruito per memorizzare quali item sono di suo interesse. In altre parole, questi algoritmi cercano di raccomandare quello che l'utente ha valutato positivamente o usato nel passato e sta esaminando nel presente. La costruzione del profilo dell'utente, spesso temporaneo, non viene basata su un modulo di registrazione che l'utente stesso deve compilare, ma su informazioni lasciate indirettamente dall'utente, le quali possono essere: i prodotti che ha maggiormente cercato e acquistato, quelli che sono stati inseriti nella lista dei desideri, etc.. Più precisamente, tra vari item candidati da raccomandare all'utente si passa per un processo di confronto con gli item piaciuti dall'utente e gli item migliori vengono suggeriti.

### 2.4.2 Collaborative filtering

I Filtri Collaborativi (*Collaborative filtering*, acronimo CF) per poter funzionare come prima cosa costruiscono un database di preferenze degli utenti sulla base di un insieme di item (o prodotti), che a loro volta possono essere presenti un database, sfruttano tecniche di analisi dei dati per risolvere il problema di aiutare gli utenti a trovare gli item che gli potrebbero piacere

producendo, eventualmente una lista dei top-N item da raccomandare per un dato utente. Un utente è sottoposto ad un processo di matching all'interno del database per scoprire quali sono i possibili *neighbors*, che corrispondono ai possibili utenti aventi storicamente delle preferenze in comune al lui. A questo punto gli item maggiormente preferiti dai neighbors sono raccomandati all'utente visto che potrebbero essere di suo interesse.

Questi sistemi tentano di predire la valutazione o la preferenza che un utente darebbe a un item basandosi su preferenze date da altri utenti, queste ultime possono essere ottenute o in modo esplicito dagli utenti o tramite misurazioni implicite. Inoltre i Filtri Collaborativi non richiedono l'uso di metadati associati agli item come nella loro controparte, i filtri Content-based.

Tuttavia, restano ancora oggi alcune sfide significative a cui sono sottoposti i sistemi di raccomandazione basati su Filtri Collaborativi.

Il primo obiettivo è quello di migliorare la scalabilità degli algoritmi di Filtri Collaborativi; questi algoritmi sono in grado di cercare anche diecimila di potenziali neighbors (utenti simili) in tempo reale, ma la richiesta dei sistemi moderni è di cercare dieci milioni di potenziali neighbors, per questo motivo possono nascere problemi di performance con i singoli utenti quando essi hanno molte informazioni.

Il secondo obiettivo è quello di migliorare la qualità dei sistemi di raccomandazione per gli utenti. Gli utenti vogliono raccomandazioni di cui possono fidarsi e che possono aiutarli a trovare item che potrebbero essere di loro gusto.

Per certi versi questi due obiettivi sono in conflitto tra di loro e per ottenere dei risultati validi e di una certa importanza è necessario trattarli in contemporanea perchè aumentare solamente la scalabilità diminuirebbe la sua qualità e viceversa. [8]

Il principale modello di Filtri Collaborativi studiato in questo elaborato e approfondito nel capitolo successivo, è il metodo definito come *Memory-based* e il vantaggio di utilizzare queste tecniche sta nel fatto di essere semplici da implementare e i risultati ottenuti sono altrettanto semplici da interpretare; mentre ci possono essere anche Filtri Collaborativi che sfruttano metodi *Model-based* che si basano sulla fattorizzazione di matrici e sono molto più funzionali per gestire il problema della parsità dei dati. Questi ultimi sono sviluppati usando algoritmi di data mining e machine learning per predire le valutazioni di utenti su item senza valutazioni, tentando di comprimere grandi database in un modello ed effettuare il processo di raccomandazione



applicando dei meccanismi di riferimento all'interno di questo modello. I CF Model-based possono rispondere alle richieste degli utenti istantaneamente. [4]

### 2.4.3 Cold Start problem

Cosa succederebbe se un nuovo utente o un nuovo item venisse aggiunto al database? Questa situazione è chiamata *Cold Start*. Ci possono essere due tipi di Cold Start:

**Visitor Cold Start** : si verifica quando un nuovo utente è stato aggiunto al database, visto che non c'è alcuno storico relativo ad esso, il sistema non sa le sue preferenze; per questo motivo diventa molto più difficile raccomandare prodotti a quel particolare utente. Per risolvere questo problema, a livello teorico, si potrebbe applicare un procedimento di raccomandazione basata sulla popolarità dei prodotti, ma solo una volta che si è venuti a conoscenza delle preferenze dell'utente, sarà possibile generare delle raccomandazioni più precise e adeguate alle sue esigenze.

**Item Cold Start** : si verifica quando un nuovo item viene inserito nel sistema. L'azione dell'utente è quella più importante per determinare il valore di questo item; maggiore l'interazione un item riceve, più è facile che venga raccomandato all'utente giusto.



# Capitolo 3

## Collaborative filtering

In questo capitolo verranno approfonditi gli algoritmi di raccomandazione implementati nella soluzione, mostrando le porzioni di codice e spiegando i vari passaggi che portano ad ottenere delle raccomandazioni.

### 3.1 Memory-based

I Filtri Collaborativi Memory-based sono stati introdotti per via delle osservazioni che vennero fatte dalla comunità, dicenddo che gli utenti si fidano maggiormente delle raccomandazioni di altri che la pensano allo stesso modo. Questi metodi mirano a calcolare le relazioni tra utenti e item attraverso lo schema dei vicini che identifica sia coppie di item che tendono ad essere usati insieme o hanno un grado di similarità alto o utenti con uno storico di item usati simile. [6] Questi approcci divennero molto famosi grazie alla loro semplicità di implementazione, molto intuitivi, non necessitano di operazioni di training sui dati e regolazione di molti parametri, inoltre l'utente può capire la ragione che sta dietro ogni raccomandazione.

Questa categoria di sistemi di raccomandazione sono definiti anche *Neighborhood-based* e possono essere ulteriormente classificati in due sottocategorie:

#### 3.1.1 User-based filtering

Questo sistema, definiti anche con l'acronimo UB-CF (*User-based Collaborative Filter*) basa tutto il suo funzionamento sulla comunità di utenti, maggiore è la sua dimensione e l'attività degli utenti con item o servizi e migliori potranno essere le raccomandazioni. Questo algoritmo fornisce dei suggerimenti ad un utente sulla base di uno o più vicini, e la similarità tra utenti può essere determinata sulla base degli item che l'utente ha utilizzato o valutato.

Molti di questi approcci possono essere generalizzati dall'algoritmo organizzato nei seguenti passi:

1. Specificare qual'è l'utente a cui si vuole applicare l'algoritmo di raccomandazione e recuperare quali utenti possono avere dato valutazioni o usato item simili al utente target. Piuttosto che recuperare tutti gli utenti, per velocizzare l'esecuzione dell'algoritmo, è possibile selezionare soltanto un gruppo di utenti in modo casuale oppure associare dei valori di similarità tra tutti gli utenti e confrontando questi valori con quello dell'utente target, selezionare i relativi utenti che superano una soglia scelta, oppure utilizzare tecniche di clustering.
2. Estrarre quegli item a cui l'utente target non ha mai interagito e per questo motivo gli possono interessare, e mostrarli sottoforma di raccomandazioni.

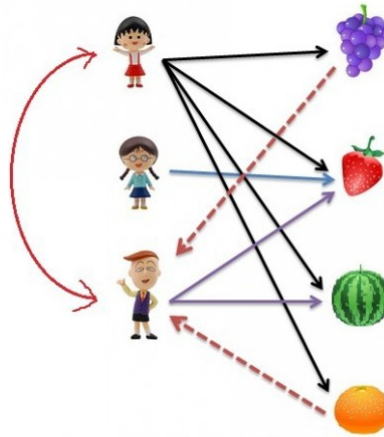


Figura 3.1: Esempio di applicazione di un sistema di raccomandazione User-based

Questi approcci sono facilmente implementabili, indipendenti dal contesto in cui sono applicati e possono essere più accurati rispetto a tecniche basate sul Content-based filtering; dall'altra parte all'aumentare del numero di utenti che vado a considerare per fare le raccomandazioni migliore è la precisione di questo processo ma anche è maggiore il costo in termini di tempo.

Nella soluzione proposta in questa tesi, l'algoritmo UB-CF viene implementato come funzione che prende in ingresso un parametro (*user\_other\_id*), come è possibile osservare da 3.1.1 corrispondente all'identificativo per l'utente, e restituisce una lista di raccomandazioni (*similar\_user\_evaluations*) corrispondenti alle Evaluation simili a quelle usate da altri utenti.

```

1 # User recommendation algortihm
2 def user_recommendation_alg(user_other_id):

```

Più precisamente come funziona l'algoritmo si suddivide nei seguenti passi:

il primo passo è quello di recuperare sulla base del parametro in ingresso alla funzione, lo `user_other_id`, tutte le evaluation che l'utente in questione ha utilizzato;

```

1     # Select the target user and its evaluations
2     target_user_evaluations = User.objects.get(other_id=
3     user_other_id).evaluations.all()\
4                                     .values('
5     other_id', 'id', 'parent_id')\
                                     .order_by('
6     other_id')

```

il secondo passo consiste nel selezionare le Evaluation usate dagli altri utenti, e viene create una lista di queste Evaluation;

```

1     # Select all other users and theirs evaluations
2     other_users = User.objects.exclude(other_id=
3     user_other_id)
4     # Creating a list with all the evaluations of other
5     users
6     other_users_evaluations = []
7     for o_users_evaluation in other_users:
8         for evaluation in o_users_evaluation.evaluations.
9         all().values('other_id', 'id', 'parent_id')\
10                                .order_by('other_id'):
11             other_users_evaluations.append(evaluation)

```

il terzo passo consiste nell'andare a determinare quali tra le Evaluation dell'utente a cui si vuole raccomandare quali sono quelle simili usate dagli altri utenti. Per determinare le Evaluation simili si è andato a confrontare il parametro *parent\_id* (identifica all'interno della tassonomia quale sia il nodo padre per quella Evaluation), associato ad ogni Evaluation, in questo modo si è andati a selezionare soltanto gli item appartenenti a una stessa categoria, eliminando eventuali nodi duplicati. E componendo una lista finale con le Evaluation restanti.

```

1     # Comparing target user's evaluations and other user'
2     s evaluations, and if there is a match the evaluation
3     is

```

```
2      # added to the 'similar_evaluations' list (the
3      matching is made comparing the 'parent_id')
4      similar_user_evaluations = []
5      for t_user_evaluation in target_user_evaluations:
6          for o_users_evaluation in other_users_evaluations:
7              # Taking only the evaluations that have:
8              different other_id (excluding the target evaluation
9              # in the recommendation) and same parent_id and
10             the evaluations that weren't added to '
11             target_user_evaluations'
12             # list and to 'similar_user_evaluations'
13             if ((t_user_evaluation['other_id'] !=
14                 o_users_evaluation['other_id']) and # Evaluations
15                 must have different 'other_id'
16                 (t_user_evaluation['parent_id'] ==
17                 o_users_evaluation['parent_id']) and # Evaluations
18                 must have the same 'parent_id'
19                 # Evaluation in all_other_evals list mustn't
20                 be already added to \
21                 not (o_users_evaluation in
22                 target_user_evaluations) and # the '
23                 target_user_evaluations' list or
24                 not (o_users_evaluation in
25                 similar_user_evaluations)): # the '
26                 similar_user_evaluations' list
27                 similar_user_evaluations.append(
28                 o_users_evaluation)
```

Nel capitolo successivo verranno mostrati degli esempio pratici in cui è stato applicato questo algoritmo.

### 3.1.2 Item-based filtering

Quando viene applicato per milioni di utenti e item, l'algoritmo UB-CF non è molto efficiente, per via della complessa computazione della ricerca di utenti simili; così in alternativa è stato introdotto l'algoritmo di filtraggio Item-based, definito anche IB-CF (*Item-based Collaborative Filter*) dove piuttosto che effettuare il confronto tra utenti simili, viene fatto un confronto tra gli item dell'utente a cui si vuole raccomandare e i possibili item simili.

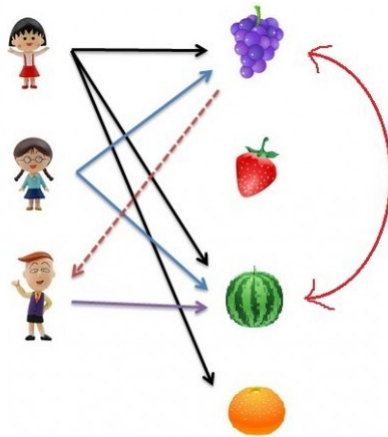


Figura 3.2: Esempio di applicazione di un sistema di raccomandazione IB-CF.

Questi sistemi sono estremamente simili ai sistemi di raccomandazione Content-based, e identificano item simili in base a come utenti gli hanno usati nel passato [8].

```

1 # Item recommendation algorithm
2 def item_recommendation_alg(item_other_id):
3     """
4     For a 'target' evaluation this algorithm suggest other
5     evaluations that belong to the same category (this means
6     that they have the same 'parent_id').
7     :param item_other_id: value representing the other_id of
8     the evaluation.
9     :return: a list of evaluations.
10    """
11    # Selecting the evaluation, which is applied this algorithm
12    # , from its other_id
13    # SELECT * FROM recommendation_app_evaluation WHERE
14    # other_id = %(item_other_id)s AND node_type = 'eva'
15    target_eval = \
16        Evaluation.objects.filter(Q(other_id=item_other_id) & Q(
17            node_type="eva"))\

```

```

13         .values('other_id', 'id', 'parent_id')[0]
14
15     # Selecting the other evaluations, excluding the target
    evaluation
16     # SELECT * FROM recommendation_app_evaluation WHERE
    other_id != %(item_other_id)s AND node_type = 'eva'
17     all_other_evals = Evaluation.objects.filter(~Q(other_id=
    item_other_id) & Q(node_type="eva"))\
18         .values('other_id', 'id', 'parent_id').
    order_by('other_id')
19
20     # Creating a list with all the evaluations that are similar
    to the target evaluation (comparing the parent_id)
21     similar_item_evaluations = []
22     for evaluation in all_other_evals:
23         # Taking only the evaluations that have: different
    other_id (excluding the target evaluation
24         # in the recommendation) and same parent_id and the
    evaluations that weren't added to similar_item_evaluations
25         # list
26         if ((target_eval['other_id'] != evaluation['other_id'])
    and # Evaluations must have different 'other_id'
27             (target_eval['parent_id'] == evaluation['parent_id'])
    and # Evaluations must have same 'parent_id'
28             # Evaluation in all_other_evals list mustn't be
    already added to \
29             not (evaluation in similar_item_evaluations)): # the
    'similar_item_evaluations' list
30         similar_item_evaluations.append(evaluation)
31
32     return similar_item_evaluations

```

Listing 3.1: Implementazione del CF-IB per le Evaluation presenti in Moon Cloud.

```

1 def target_recommendation_alg(target_id):
2     """
3     For a target chose by a user this algorithm search the
    possible evaluations that can be recommended for that user
4     .
5     :param target_id: identifier of a particular target.
6     :return: a list of evaluations.
7     """
8     # Retriving all the evaluations in the database
    evaluations = Evaluation.objects.filter(node_type="eva")
9
10    # Saving in the target_evaluations list the evaluations
    which controls have target_type_id equal to target_id
11    target_evaluations = []

```



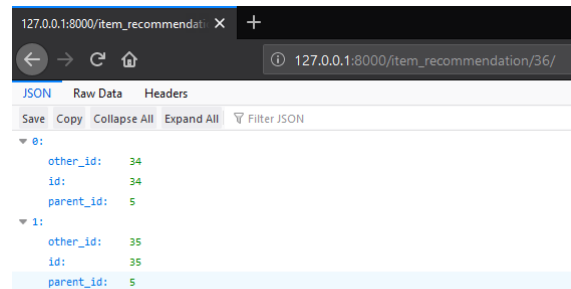


Figura 3.3: Esempio di esempio di una risposta in JSON a una chiamata Rest all'algorithmo di raccomandazione IB-CF per le Evaluation supportate da Moon Cloud.

```

12 for evaluation in evaluations: # Scanning all the
    evaluations
13     for evaluation_controls in evaluation.controls.filter(
        target_type_id=target_id):
14         if not(evaluation in target_evaluations): # Excluding
            evaluations duplicated
15             target_evaluations.append(evaluation)
16
17 # Converting the Evaluation model's instance in a dict and
    putting the evaluation, as a dict, in a list
18 target_evaluations_serializer = EvaluationSerializer(
    target_evaluations, many=True)
19
20 return target_evaluations_serializer.data

```

Listing 3.2: Implementazione del IB-CF per i Target supportati da Moon Cloud.

### 3.1.3 Hybrid filtering

```

1 # HYBRID IMPLEMENTATION OF USER RECOMMENDATION ALGORITHM AND
    ITEM RECOMMENDATION ALGORITHM
2 @api_view(['GET'])
3 def hybrid_recommendation(request, user_other_id):
4     """
5     For a User-X this algorithm compare his evaluations with
        other users's evaluations and return all the evaluations
6     that are similiar (same 'parent_id') to User-X's
        evaluations; then from all the evaluations used by the
        User-X other
7     recommendation are computed using the item recommendation
        algorithm; and all the recommendations from the first step

```

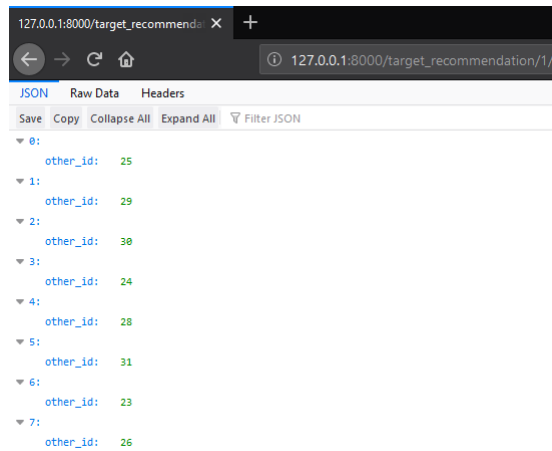


Figura 3.4: Esempio di esempio di una risposta in JSON a una chiamata Rest all'algoritmo di raccomandazione IB-CF per i Target supportati da Moon Cloud.

```

8  (here is used the user_recommendation_alg) and the second
   step (here is used the item_recommendation_alg)
9  are put together.
10 :param request: http GET request.
11 :param user_other_id: value representing the other_id of a
   User.
12 :return: json response with the evaluations to recommend.
13 """
14
15 # Trying to retrieve the actual User with user_other_id
16 user = User.objects.get(other_id=user_other_id)
17
18 # Taking from the user_recommendation_alg the evaluation
   recommended from this approach (similar_user_evaluations)
19 # and the user's evaluations (target_user_evaluations)
20 target_user_evaluations, similar_user_evaluations =
   user_recommendation_alg(user_other_id)
21
22 # For every evaluation used by users is extracted all other
   possible evaluations that have the same 'parent_id'
23 similar_item_evaluations = []
24 for t_user_evaluation in target_user_evaluations: # for
   every target user's evaluations
25     for item_evaluation in item_recommendation_alg(
   t_user_evaluation['other_id']): # is applied the
   item_recommendation algorithm
26         # Taking only the evaluations that have: different
   other_id (excluding the target evaluation

```

```

27     # in the recommendation) and same parent_id and the
    evaluations that weren't added to '
    similar_item_evaluations'
28     # list or to 'similar_user_evaluations' or to '
    target_user_evaluations'
29     if ((t_user_evaluation['other_id'] != item_evaluation['
    other_id']) and # Evaluations must have different 'id'
30         (t_user_evaluation['parent_id'] == item_evaluation[
    'parent_id']) and # Evaluations must have the same '
    parent_id'
31         # Evaluation in all_other_evals list mustn't be
    already added to \
32         not (item_evaluation in similar_item_evaluations)
    and # the 'similar_item_evaluations' list,
33         not (item_evaluation in similar_user_evaluations)
    and # the 'similar_user_evaluations' list or
34         not (item_evaluation in target_user_evaluations)):
    # the 'target_user_evaluations' list
35         similar_item_evaluations.append(item_evaluation)
36
37 # Putting together the evaluations recommended in
    similar_user_evaluations list and similar_item_evaluations
    list
38 similar_evaluations = []
39 # Adding to similar_evaluations list the evaluation in the
    similar_user_evaluations list
40 for s_user_evaluation in similar_user_evaluations:
41     similar_evaluations.append(s_user_evaluation)
42 # Adding to similar_evaluations list the evaluation in the
    similar_item_evaluations list
43 for item_evaluation in similar_item_evaluations:
44     # Taking only the evaluations that weren't added to \
45     if (not (item_evaluation in similar_evaluations) and #
    the 'similar_evaluations' list or
46         not (item_evaluation in target_user_evaluations)): #
    the 'target_user_evaluations' list
47         similar_evaluations.append(item_evaluation)
48 similar_evaluations = sorted(similar_evaluations, key=
    lambda i: i['other_id'])
49
50 return JsonResponse(similar_evaluations, safe=False)

```

Listing 3.3: Implementazione di un sistema raccomandazione ibrido che metta insieme le Raccomandazioni generate dagli algoritmi Item-based e User-based per le Evaluation presenti in Moon Cloud.

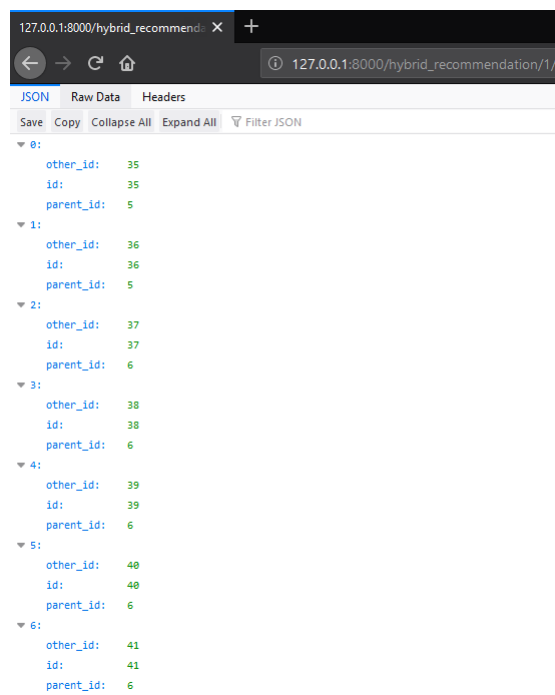


Figura 3.5: Esempio di esempio di una risposta in JSON a una chiamata Rest all'algoritmo di raccomandazione Ibrido.

# Capitolo 4

## Descrizione della soluzione

In questo capitolo verrà approfondito l'aspetto puramente pratico e le fasi che hanno portato alla realizzazione della soluzione; inoltre verranno mostrate le applicazioni pratiche degli aspetti teorici enunciati nei capitoli precedenti, unitamente alle difficoltà principali incontrate.

Prima di poter costruire il sistema di raccomandazione proposto in questa tesi, sono state eseguite delle operazioni preliminari per poter impostare il progetto di Django e la relativa applicazione che implementerà effettivamente la soluzione.

Come annunciato nei capitoli precedenti per procedere alla costruzione di un sistema di raccomandazione bisogna avere a disposizione una base di dati solida da cui attingere tutte le informazioni; ed è proprio questo il primo passo che è stato seguito, disegnare e progettare una base di dati da cui partire per la realizzazione degli algoritmi proposti. In generale Moon Cloud possiede una struttura delle Evaluation ad albero, quindi anche di conseguenza anche le tabelle del database rispecchiano questa struttura, sulla base delle considerazioni sulle tecniche adottate sono state fatte nei capitoli precedenti.

Per implementare un modified pre-order trasversal tree in Django, si è fatto uso del package MPTT, come detto in precedenza, questa tecnica è usata per memorizzare dati gerarchici in un database, puntando all'efficienza nelle operazioni di recupero dei dati e scendendo a compromessi per quanto riguarda le operazioni di inserimento e spostamento dei nodi all'interno della struttura. Grazie all'usilio di questa utility la costruzione dei Model del progetto sono stati semplificati e qui di seguito 4 è possibile trovare le porzioni principali del codice costituente i Model, i quali poi vengono utilizzati da

Django per la generazione della base di dati.

```

1  # Target supported by Moon Cloud
2  class TargetType(models.Model):
3      TYPES = (
4          ('host', 'host'),
5          ('windows', 'windows'),
6          ('url', 'url'),
7          ('azure', 'azure'),
8          ('aws', 'aws')
9      )
10     name = models.CharField(max_length=150, choices=TYPES,
11                             default="host")
12     descr = models.TextField(max_length=1000, default="none")
13     # Description of a target
14
15 # Control that can be part of evaluations
16 class Control(MPTTModel):
17     other_id = models.IntegerField(default=-1, unique=True)
18     parent = TreeForeignKey('self', on_delete=models.CASCADE,
19                             null=True, blank=True, related_name='children')
20     name = models.CharField(max_length=150, unique=True)
21     descr = models.TextField(max_length=1000, default="none")
22     # Description of a node in the taxonomy
23     TYPES = (
24         ('cat', 'category'),
25         ('con', 'control')
26     )
27     # Possible node type of the taxonomy (category node or
28     # control node)
29     node_type = models.CharField(max_length=3, choices=TYPES,
30                                  default='cat')
31     target_type = models.ForeignKey(TargetType, null=True,
32                                     blank=True, on_delete=models.CASCADE) # It's null for the
33                                     # root node and category nodes
34
35 # Evaluation used by users (group of controls)
36 class Evaluation(MPTTModel):
37     other_id = models.IntegerField(default=-1, unique=True)
38     parent = TreeForeignKey('self', on_delete=models.CASCADE,
39                             null=True, blank=True, related_name='children')
40     name = models.CharField(max_length=150, unique=True)
41     descr = models.TextField(max_length=1000, default="none")
42     # Description of a node in the taxonomy
43     TYPES = (
44         ('cat', 'category'),
45         ('eva', 'evaluation')
46     )

```

```

39 # Possible node types of the taxonomy (category node or
    evaluation node)
40 node_type = models.CharField(max_length=3, choices=TYPES,
    default='cat')
41 controls = models.ManyToManyField(Control) # Evaluation
    can be composed of one or more controls
42
43
44 # Users with a Moon Cloud account who can use evaluations
45 class User(models.Model):
46     other_id = models.IntegerField(default=-1, unique=True)
47     email = models.EmailField(max_length=50, unique=True)
48     evaluations = models.ManyToManyField(Evaluation, blank=True)
    # Evaluations chosen by user
49
50
51 # Target table to save the targets type (more than one) that
    a user can have
52 class Target(models.Model):
53     user = models.ForeignKey(User, blank=True, on_delete=models
    .CASCADE) # User has chosen some target_type
54     other_id = models.IntegerField(default=-1, unique=True)
55     target_type = models.ForeignKey(TargetType, blank=True,
    on_delete=models.CASCADE) # TargetType Id

```

Listing 4.1: Parti principali del codice dei Models della soluzione

A partire da questo Model vennero introdotte nel database le seguenti tabelle, le quali è possibile visionare nella Figura 4.1:

**Control** : contiene l'insieme dei software che vengono poi effettivamente eseguiti all'interno di una Evaluation, i campi `other_id` (identificativo che fa riferimento al database effettivo di Moon Cloud), `descr` (una descrizione del funzionamento del controllo), `node_type` (definisce se il nodo è un Evaluation o un nodo Categoria) definiscono le caratteristiche del controllo mentre `lft`, `right`, `tree_id`, `level` e `parent` sono introdotti automaticamente dal package MPTT per poter rappresentare i dati in modo gerarchico, infine `target_type_id` rappresenta, quel controllo a quale Target viene associato.

**Evaluation** : contiene l'insieme di Evaluation che un utente può eseguire per un certo Target, e allo stesso modo i campi contenuti nella tabella Control. La tabella intermedia `evaluation_controls` permette di memorizzare quali controlli sono associati a quali Evaluation.

**User** : contiene gli utenti registrati alla piattaforma Moon Cloud, e sono anche loro, come con le tabelle precedenti, identificati con un campo

other\_id, e distinti da un email. La tabella intermedia user\_evaluations permette di memorizzare quali Evaluation un utente ha selezionato e usato.

**Target** : è utilizzata per memorizzare quali Target un utente ha inserito e sui quali vuole effettuare dei processi di Evaluation.

**TargetType** : specifica quali sono i tipi di Target supportati da Moon Cloud.



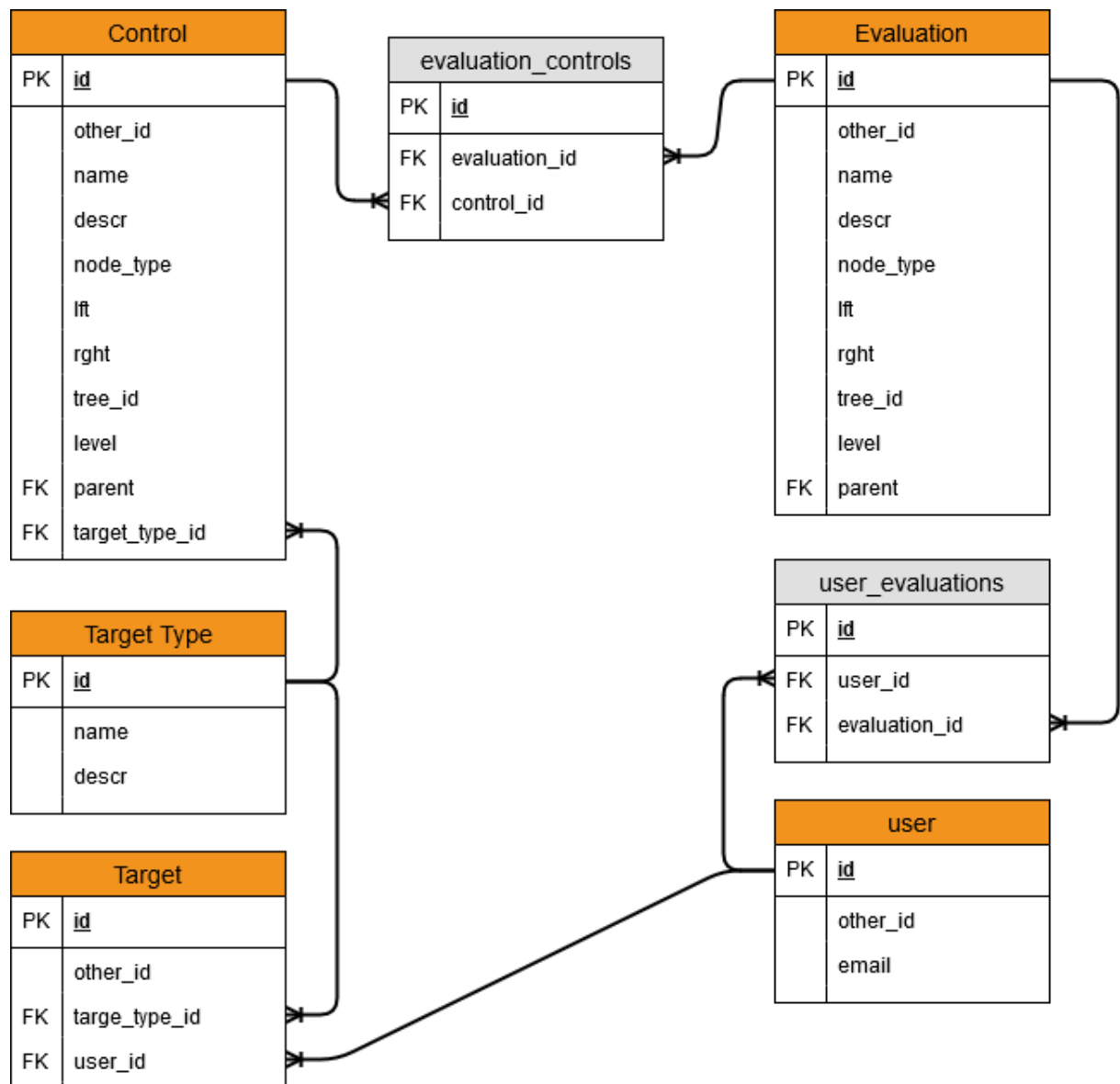


Figura 4.1: Struttura del database

Successivamente per poter testare che la tassonomia creata per le Evaluation e i Controlli fosse corretta e funzionante si è implementata un'interfaccia Web a scopo didattico. Avviando il server, la home page che viene prooposta è mostrata nella Figura 4.2, dalla quale è possibile accedere alle pagine specifiche per la navigazione della tassonomia delle Evaluation piuttosto che dei Controlli; inoltre tramite la barra di navigazione è possibile tornare a questa home page o accedere alla admin page generata da Django, e successivamente personalizzata, per poter manipolare la base di dati.

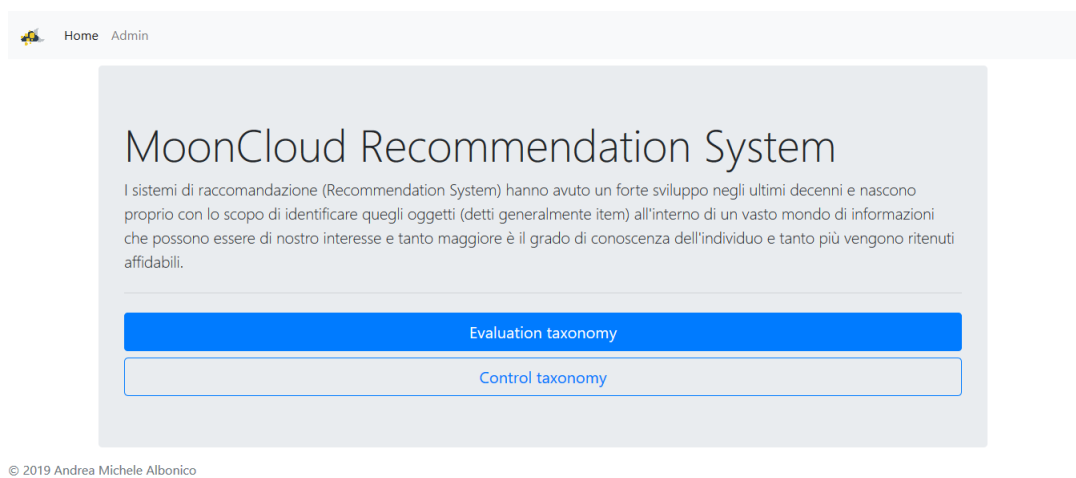
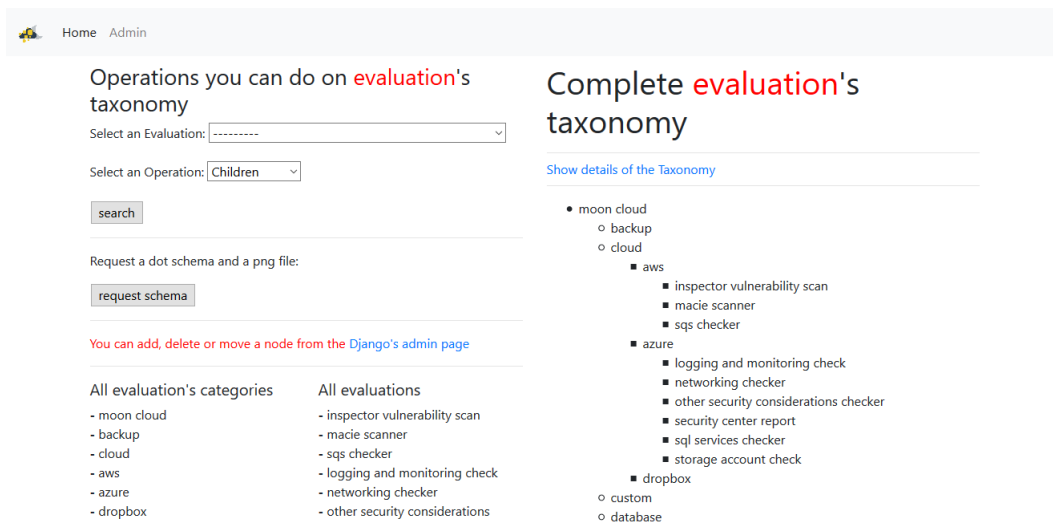


Figura 4.2: Home page

```
1 # Index page where you can choose to navigate the evaluation
   taxonomy or the control taxonomy
2 def index(request):
3     return render(request, "recommendation_app/index.html")
```

Listing 4.2: Parte principale del codice delle View della soluzione per gestire l'accesso alla home page

Una volta scelta la tassonomia su cui si vuole navigare, è possibile, per ogni singolo nodo, recuperare: i discendenti, la famiglia, i fratelli, e gli antenati; ed è anche possibile scaricare un file scritto in linguaggio DOT e un'immagine in formato png rappresentante la gerarchia dei dati contenuti nella base dati. Inoltre è possibile osservare in maniera più approfondita le informazioni rilevanti sulla tassonomia contenute nelle tabelle del database.



Home Admin

## Operations you can do on evaluation's taxonomy

Select an Evaluation:

Select an Operation:

Request a dot schema and a png file:

You can add, delete or move a node from the [Django's admin page](#)

**All evaluation's categories**

- moon cloud
- backup
- cloud
- aws
- azure
- dropbox

**All evaluations**

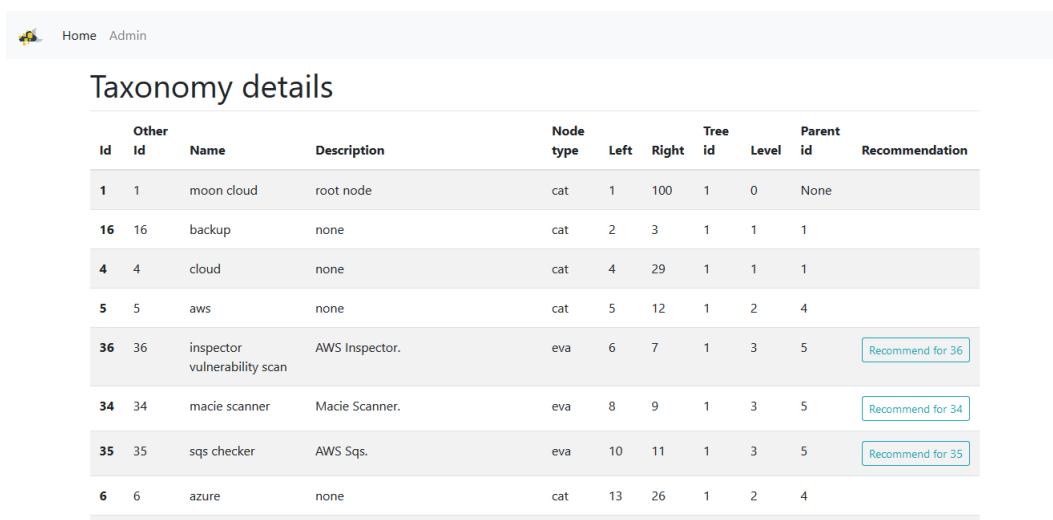
- inspector vulnerability scan
- macie scanner
- sqs checker
- logging and monitoring check
- networking checker
- other security considerations

## Complete evaluation's taxonomy

[Show details of the Taxonomy](#)

- moon cloud
  - backup
  - cloud
    - aws
      - inspector vulnerability scan
      - macie scanner
      - sqs checker
    - azure
      - logging and monitoring check
      - networking checker
      - other security considerations checker
      - security center report
      - sql services checker
      - storage account check
    - dropbox
      - custom
      - database

Figura 4.3: Home page per la navigazione della tassonomia

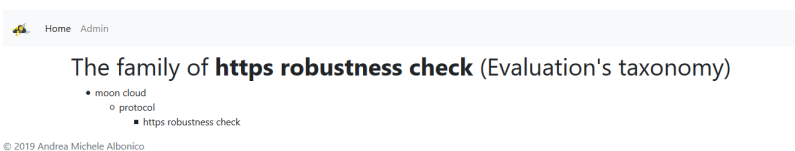


Home Admin

## Taxonomy details

	Other Id	Id	Name	Description	Node type	Left	Right	Tree id	Level	Parent id	Recommendation
1	1	moon cloud	root node		cat	1	100	1	0	None	
16	16	backup	none		cat	2	3	1	1	1	
4	4	cloud	none		cat	4	29	1	1	1	
5	5	aws	none		cat	5	12	1	2	4	
36	36	inspector vulnerability scan	AWS Inspector.		eva	6	7	1	3	5	<a href="#">Recommend for 36</a>
34	34	macie scanner	Macie Scanner.		eva	8	9	1	3	5	<a href="#">Recommend for 34</a>
35	35	sqs checker	AWS Sqs.		eva	10	11	1	3	5	<a href="#">Recommend for 35</a>
6	6	azure	none		cat	13	26	1	2	4	

Figura 4.4: Dettagli della tassonomia sottoforma di tabella come nella base di dati



Home Admin

## The family of **https robustness check** (Evaluation's taxonomy)

- moon cloud
  - protocol
    - https robustness check

© 2019 Andrea Michele Albonico

Figura 4.5: Risultato dell'operazione selezionata sul nodo in questione

Qui di seguito è possibile trovare il codice scritto all'interno delle View per poter eseguire tutte le operazioni descritte sopra.

```

1
2 # The home page shows all taxonomy and a form to make
   operations on it
3 def tax_index(request, parameter):
4     # If this is a POST request we need to process the form
       data
5     if request.method == 'POST':
6         # Create a form instance and populate it with data from
           depending on the parameter
7         if (parameter == "evaluation"):
8             form = EvaluationOperationForm(request.POST)
9         else:
10            form = ControlEvaluationForm(request.POST)
11        # Check whether it's valid:
12        if form.is_valid():
13            # Process the data in form.cleaned_data as required
14            nodename_form = form.cleaned_data['nodeName']
15            taxonomy_operation_form = form.cleaned_data['actionTax'
16        ]
17        # Redirect to a new URL (page that show a part of the
           taxonomy, depending on the action user has chosen):
18        return redirect(
19            reverse('rec:tax_index', args=[parameter]) + str(
20                nodename_form) + '_' + taxonomy_operation_form)
21    # If it's a GET method we'll create a blank form
22    else:
23        if (parameter == "evaluation"):
24            form = EvaluationOperationForm()
25        else:
26            form = ControlEvaluationForm()
27
28    # Depending on the parameter, I'm getting all the
       categories of Evaluations or Controls taxonomy and save it
       in a
29    # list called "categories_list"
30    if (parameter == "evaluation"):
31        q_categories = Evaluation.objects.filter(node_type='cat')
32    else:
33        q_categories = Control.objects.filter(node_type='cat')
34    categories_list = []
35    for node in q_categories:
36        categories_list.append(node.name)
37
38    # Depending on the parameter, I'm getting all the
       categories of Evaluations or Controls node in the taxonomy
       # and save it in a list called "node_list"

```

```

38 if (parameter == "evaluation"):
39     q_nodes = Evaluation.objects.filter(node_type='eva')
40 else:
41     q_nodes = Control.objects.filter(node_type='con')
42 node_list = []
43 for node in q_nodes:
44     node_list.append(node.name)
45
46 # Depending on the parameter, I'm getting all the
47 # Evaluations or Controls taxonomy
48 if (parameter == "evaluation"):
49     tax = Evaluation.objects.all()
50 else:
51     tax = Control.objects.all()
52
53 # Passing the complete taxonomy and data to fill the form
54 # so you can operate on the taxonomy
55 args = {'tax': tax,
56         'categories': categories_list,
57         'nodes': node_list,
58         'form': form,
59         'request_path': parameter}
60
61 return render(request, "recommendation_app/tax_index.html",
62              args)
63
64 # Show the taxonomy's details page showing an overview of the
65 # taxonomy
66 def tax_details(request, parameter):
67     if (parameter == 'evaluation'):
68         tax_details_obj = Evaluation.objects.all()
69     else:
70         tax_details_obj = Control.objects.all()
71
72     return render(request, "recommendation_app/tax_details.html",
73                  {'tax_details': tax_details_obj,
74                   'parameter': parameter})
75
76 # Create the .dot file (based on the Dot language) and the
77 # graph showing the taxonomy in .png format
78 def dot_graph(request, parameter):
79     # Create a graph object
80     taxonomy_dot_object = Graph(comment='Taxonomy', format='png',
81                                ')
82     # Fill the graph with every node in the database (
83     # evaluations/controls node and categories nodes),

```

```

79 # and create a link with the parent node
80 if (parameter == 'evaluation'):
81     taxonomy_nodes = Evaluation.objects.all()
82 else:
83     taxonomy_nodes = Control.objects.all()
84 i = 0
85 for node in taxonomy_nodes.order_by('level'):
86     # This If construct will prevent the adding of an empty
87     # node to the root node in the graph
88     if (i == 0):
89         # Insert the root node
90         taxonomy_dot_object.node(str(node.id), label=str(node.
91         name))
92     else:
93         # Insert the other nodes
94         taxonomy_dot_object.node(str(node.id), label=str(node.
95         name))
96         taxonomy_dot_object.edge(str(node.parent_id), str(node.
97         id))
98     i += 1
99 # Specify where I want to save the .png image and the .dot
100 # file
101 taxonomy_dot_object.render('taxonomy_output/taxonomy.dot')
102
103 # This function is used to zip a directory
104 def make_zipdir(path, ziph):
105     # Ziph is zipfile handle
106     for root, dirs, files in os.walk(path):
107         for file in files:
108             ziph.write(os.path.join(root, file))
109
110 # Making the zip file
111 zip_file = zipfile.ZipFile('taxonomy_output.zip', 'w',
112     compression=zipfile.ZIP_DEFLATED)
113 make_zipdir('taxonomy_output/', zip_file)
114 zip_file.close()
115
116 # Remove the directory which was zipped and all files
117 # inside
118 shutil.rmtree("taxonomy_output/")
119
120 return redirect(reverse('rec:index'))
121
122 # Method to navigate the taxonomy
123
124 # Based on the MPTT's method 'get descendants' that return
125 # the descendants of a model instance, in tree order

```

```

120 def show_descendants(request, nodename, parameter):
121     if (parameter == 'evaluation'):
122         q_result = Evaluation.objects.get(name=nodename).
123         get_descendants(include_self=False)
124         # Get the count of descendants of the model instance
125         q_result_num = Evaluation.objects.get(name=nodename).
126         get_descendant_count()
127     else:
128         q_result = Control.objects.get(name=nodename).
129         get_descendants(include_self=False)
130         # Get the count of descendants of the model instance
131         q_result_num = Control.objects.get(name=nodename).
132         get_descendant_count()
133
134     return render(request, "recommendation_app/tax_node_details
135     .html",
136                 {'tax_type': (str(parameter)).capitalize(),
137                  'descendants': q_result,
138                  'node_exe': nodename,
139                  'method': 'descendants',
140                  'num_descendants': q_result_num})
141
142 # Based on the MPTT's method 'get children' that return the
143 # immediate children of a model instance, in tree order
144 def show_children(request, nodename, parameter):
145     if (parameter == 'evaluation'):
146         q_result = Evaluation.objects.get(name=nodename).
147         get_children()
148     else:
149         q_result = Control.objects.get(name=nodename).
150         get_children()
151
152     return render(request, "recommendation_app/tax_node_details
153     .html",
154                 {'tax_type': (str(parameter)).capitalize(),
155                  'children': q_result,
156                  'node_exe': nodename,
157                  'method': 'children'})
158
159 # Based on the MPTT's method 'get family' that return the
160 # ancestors, the model instance itself and the descendants,
161 # in tree order
162 def show_family(request, nodename, parameter):
163     if (parameter == 'evaluation'):
164         q_result = Evaluation.objects.get(name=nodename).
165         get_family()
166     else:

```

```

158     q_result = Control.objects.get(name=nodename).get_family
159     ()
160     return render(request, "recommendation_app/tax_node_details
161         .html",
162         {'tax_type': (str(parameter)).capitalize(),
163         'family': q_result,
164         'node_exe': nodename,
165         'method': 'family'})
166
167 # Based on the MPTT's method 'get siblings' that return
168 # siblings of the model instance (root nodes are considered
169 # to be siblings of other root nodes)
170 def show_siblings(request, nodename, parameter):
171     if (parameter == 'evaluation'):
172         q_result = Evaluation.objects.get(name=nodename).
173         get_siblings()
174     else:
175         q_result = Control.objects.get(name=nodename).
176         get_siblings()
177
178     return render(request, "recommendation_app/tax_node_details
179         .html",
180         {'tax_type': (str(parameter)).capitalize(),
181         'siblings': q_result,
182         'node_exe': nodename,
183         'method': 'siblings'})

```

Listing 4.3: Parti principali del codice delle View della soluzione per gestire la navigazione delle tassonomie, quella delle Evaluation e quella dei Controlli

Per poter agilmente manipolare la base di dati, Django mette a disposizione la cosiddetta Admin Page mostrata in Figura 4.6, che è stata personalizzata per mostrare le tabelle su cui è possibile effettuare modifiche, e per ognuna vengono mostrate le informazioni più rilevanti, come mostrato dalla Figura 4.7 nel caso della tabella Evaluation, e dalla quale è possibile effettuare ricerche, eliminare direttamente i dati contenuti nel database e aggiungere nuovi dati, come mostrato in Figura 4.8.



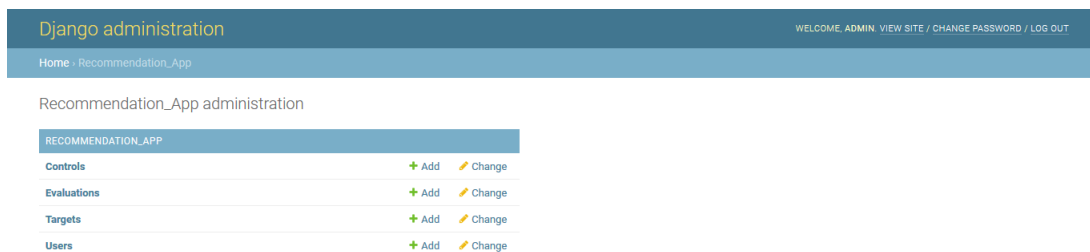


Figura 4.6: Admin page

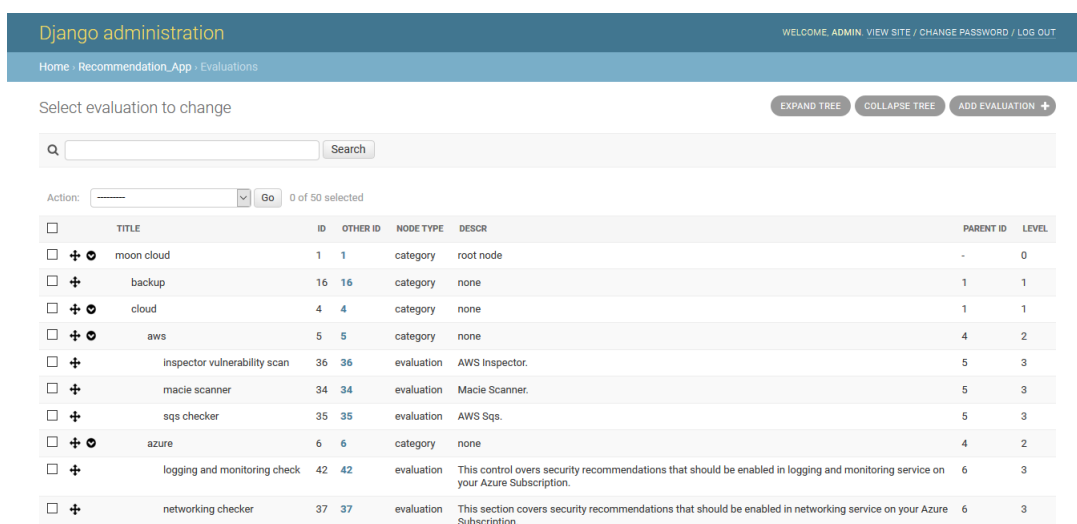


Figura 4.7: Esempio di Admin page per le Evaluation

The screenshot shows the Django administration interface for the 'Recommendation\_App' app, specifically the 'Add evaluation' page. The page has a dark blue header with 'Django administration' on the left and 'WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT' on the right. Below the header is a breadcrumb trail: 'Home > Recommendation\_App > Evaluations > Add evaluation'. The main content area is titled 'Add evaluation' and contains several form fields: 'Other id:' with a value of '-1' and a dropdown arrow; 'Parent:' with a dropdown menu and a plus icon; 'Name:' with an empty text input; 'Descr:' with a large text area containing the word 'none'; 'Node type:' with a dropdown menu set to 'category'; and 'Controls:' with a list box containing several items: 'wordpress vulnerability assessment check', 'web vuln scan', 'wappalizer', 'portscan', 'observatory', 'joomla scan', 'ping checker', 'lightweight vuln scan', and 'find file'. A plus icon is next to the list box. At the bottom right, there are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'. A small note at the bottom of the 'Controls' list box says: 'Hold down "Control", or "Command" on a Mac, to select more than one.'

Django administration

WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Recommendation\_App > Evaluations > Add evaluation

Add evaluation

Other id: -1

Parent:

Name:

Descr: none

Node type: category

Controls: wordpress vulnerability assessment check, web vuln scan, wappalizer, portscan, observatory, joomla scan, ping checker, lightweight vuln scan, find file

Hold down "Control", or "Command" on a Mac, to select more than one.

Save and add another Save and continue editing SAVE

Figura 4.8: Esempio di Admin page per il caso in cui si vuole aggiungere una nuova Evaluation

## Capitolo 5

### Conclusioni

Presentazione dei risultati e conclusioni: la presentazione dei risultati dovrebbe consistere in una descrizione tecnica dei risultati raggiunti, unitamente ad un commento critico e ad un'analisi della rispondenza agli obiettivi iniziali (si consiglia pertanto di motivare la rilevanza dei risultati e l'eventuale scostamento dagli obiettivi iniziali). La sezione relativa ai risultati dovrebbe infine contenere una sintesi critica e un giudizio sull'esperienza effettuata, che renda conto di aspetti positivi e negativi per il tirocinante e per l'ente ospitante, del valore formativo, professionale e umano, così via.



# Bibliografia

- [1] M. Anisetti et al. «A semi-automatic and trustworthy scheme for continuous cloud service certification». In: *IEEE TRANSACTIONS ON SERVICES COMPUTING* (2017). DOI: 10.1109/TSC.2017.2657505.
- [2] M. Anisetti et al. «Moon Cloud: A Cloud Platform for ICT Security Governance». In: (dic. 2018), pp. 1–7. DOI: 10.1109/GLOCOM.2018.8647247.
- [3] *Django documentation*. <https://docs.djangoproject.com/en/2.2/>. 2019.
- [4] Minh-Phung Do, Dung Nguyen e Academic Network of Loc Nguyen. «Model-based approach for Collaborative Filtering». In: ago. 2010.
- [5] *MDN Django documentation*. <https://developer.mozilla.org/it/docs/Learn/Server-side/Django/>. 2019.
- [6] Miquel Montaner, Beatriz López e Josep Lluís de la Rosa. «A Taxonomy of Recommender Agents on the Internet». In: *Artificial Intelligence Review* 19.4 (giu. 2003), pp. 285–330. ISSN: 1573-7462. DOI: 10.1023/A:1022850703159. URL: <https://doi.org/10.1023/A:1022850703159>.
- [7] *Python 3.7 documentation*. <https://docs.python.org/3.7/>. 2019.
- [8] Badrul Sarwar et al. «Item-based Collaborative Filtering Recommendation Algorithms». In: WWW '01 (2001), pp. 285–295. DOI: 10.1145/371920.372071. URL: <http://doi.acm.org/10.1145/371920.372071>.