

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



Rilevazione di vulnerabilità software in librerie di terze parti

Tesi di laurea

Relatore

Prof. Tullio Vardanega

Laureando

Gionata Legrottaglie

ANNO ACCADEMICO 2022-2023

Sommario

Il presente documento descrive il lavoro che ho svolto durante il periodo di *stage*, della durata di trecentoventi ore, dal laureando Gionata Legrotttaglie presso l'azienda Sanmarco informatica S.p.a.

Gli obiettivi da raggiungere erano diversi.

In primo luogo era richiesto lo sviluppo di un plugin Gradle per l'analisi statica delle dipendenze software di un progetto Gradle o npm; in secondo luogo era richiesto di sviluppare dei servizi REST per il salvataggio dei risultati del plugin e per effettuare la ricerca delle vulnerabilità software note e, infine, una *web application* per la visualizzazione dei risultati.

Questo documento è strutturato in quattro capitoli principali:

1. Il primo capitolo offre una panoramica del contesto aziendale e illustra gli strumenti utilizzati durante lo *stage*.
2. Nel secondo capitolo viene presentata la proposta di *stage*, con un focus sugli obiettivi da raggiungere.
3. Il terzo capitolo descrive in dettaglio le attività svolte durante lo *stage*.
4. Il quarto capitolo contiene riflessioni personali sull'esperienza lavorativa e sulle competenze acquisite.

Durante la scrittura ho utilizzato termini in lingua inglese, in quanto è la lingua più utilizzata nel settore informatico, per riferirmi a concetti tecnici, essi sono stati evidenziati in *corsivo*.

Per mettere in evidenza termini di particolare importanza ho utilizzato il **grassetto**. Ho creato un glossario per chiarire il significato di alcuni termini tecnici di non immediata comprensione, essi sono evidenziati in azzurro.

Ho allegato ad ogni immagine un numero progressivo, in modo da poterla citare nel testo, ed una didascalia per descrivere il contenuto e per citarne la fonte, se non è di mia creazione.

Indice

1	L'azienda	1
1.1	Sanmarco informatica S.p.a.	1
1.1.1	Descrizione dell'azienda	1
1.1.2	Organizzazione e prodotti	1
1.1.3	Strategie e obiettivi	2
1.2	Il <i>team</i> di sviluppo	5
1.3	Strumenti utilizzati	6
1.3.1	Convenzioni	7
1.4	Rapporto con l'innovazione	8
2	Il progetto di <i>stage</i>	9
2.1	Lo <i>stage</i> per Sanmarco informatica S.p.a.	9
2.2	La proposta di <i>stage</i>	10
2.3	Obiettivi e aspettative	11
2.4	Vincoli	12
2.5	La scelta dello <i>stage</i>	16
2.6	Pianificazione e interazioni con il tutor	17
3	Svolgimento del progetto	18
3.1	Analisi dei requisiti	18
3.2	Analisi dei Requisiti Durante lo <i>stage</i>	18
3.2.1	Tracciamento dei requisiti	20
3.3	Progettazione architetturale	21
3.4	Verifica e validazione	32
3.5	I risultati	32
A	Appendice A	33
	Acronimi e abbreviazioni	34
	Glossario	35
	Bibliografia	37

Elenco delle figure

1.1	Le <i>Business Unit</i> di Sanmarco informatica S.p.a. ed i loro Prodotti. Fonte: https://www.sanmarcoinformatica.it/intranet.pag	3
1.2	Strumenti utilizzati	7
2.1	Sequenza temporale delle attività scritte nel piano di lavoro.	11
2.2	Comparazione tra <i>Gradle</i> e <i>Maven</i> . Fonte: https://Gradle.org/maven-vs-Gradle	13
2.3	Caratteristiche di <i>Angular</i>	14
2.4	Differenze tra <i>database</i> a grafo e <i>database</i> relazionali. Fonte: https://aws.amazon.com/it/compare/the-difference-between-graph-and-relational-database	15
2.5	Pro e contro di fare lo <i>stagen</i> nell'azienda per cui lavoro.	16
2.6	Metodologia <i>Agile</i> . Fonte: https://jordanjob.me/blog/scrum-diagram	17
3.1	Grafico a torta che mostra la suddivisione in percentuale dei vari requisiti.	19
3.2	Grafico a torta che mostra la suddivisione dei requisiti in base all'importanza.	19
3.3	Tabella che mostra il tracciamento dei requisiti su <i>Confluence</i> , con il collegamento a <i>Jira</i>	21
3.4	<i>Board</i> di <i>Jira</i> che mostra lo stato di avanzamento del progetto.	21
3.5	Esempio di grafo di dipendenze tra pacchetti.	22
3.6	Struttura finale del grafo di dipendenze.	23
3.7	<i>Mockup</i> grafico dell'interfaccia grafica.	27
3.8	Struttura del progetto <i>frontend</i>	27
3.9	Modulo di ricerca dipendenze per progetto.	28
3.10	Casella di ricerca con lista di suggerimenti.	28
3.11	Risultato tabellare.	28
3.12	Risultato ad albero.	28
3.13	Visualizzazione di tutti gli aggiornamenti disponibili.	29
3.14	Esempio di ricerca per utilizzo.	29
3.15	Modalità <i>query</i> libera.	30

Elenco delle tabelle

2.1	Obiettivi obbligatori	12
2.2	Obiettivi desiderabili	12

Capitolo 1

L'azienda

1.1 Sanmarco informatica S.p.a.

1.1.1 Descrizione dell'azienda

Nata nel 1984, Sanmarco informatica S.p.a. rappresenta oggi una realtà di primo piano nel panorama della consulenza e dello sviluppo *software*. Vanta un portfolio con oltre 2500 clienti, tra cui spiccano nomi di rilievo internazionale.

L'impegno verso la clientela è il pilastro su cui si fonda l'azienda, che si estende con uffici in molteplici regioni italiane: dal Veneto alla Lombardia, passando per Emilia-Romagna, Friuli-Venezia Giulia, Toscana, Puglia e Campania. Un *team* di oltre 600 professionisti, altamente qualificati, costituisce la forza lavoro di Sanmarco informatica S.p.a..

Il fulcro dell'innovazione è localizzato nel Centro di Sviluppo e Ricerca (CSV) a Grignano di Zocco (VI). Qui, più di 200 tra sviluppatori e tecnici lavorano in sinergia per creare soluzioni *software* su misura, efficienti e affidabili, progettate per soddisfare le esigenze specifiche di ogni cliente.

1.1.2 Organizzazione e prodotti

L'organizzazione di Sanmarco informatica S.p.a. è strutturata in diverse *Business Unit* (BU), ciascuna con una visione, *mission*, strategie e obiettivi specifici. Operando in modo autonomo o semi-autonomo, queste unità hanno una struttura organizzativa propria e sono responsabili dei loro bilanci economici, permettendo una focalizzazione mirata su mercati geografici, gruppi di clienti specifici. Questa struttura rende Sanmarco informatica S.p.a. agile e reattiva alle dinamiche di mercato.

Le BU in Sanmarco informatica S.p.a. sono 11, distinte per aree di specializzazione, come illustrato in figura 1.1:

- **JGALILEO:** offre l'*Enterprise Resource Planning (ERP)* Jgalileo, un sistema di gestione integrato che ottimizza i processi aziendali per imprese di varie dimensioni, con un focus sulle normative fiscali internazionali;
- **NEXTBI:** si concentra su *Information Technology* e consulenza strategica, specializzandosi in *marketing*, vendite, *retail*, innovazione cliente, *Business Intelligence* e soluzioni *Internet of Things (IoT)*;

- **4WORDS:** fornisce soluzioni *Business to Business (B2B)*, app e *Customer Relationship Management (CRM)*, per potenziare il *business* attraverso strumenti digitali, inclusi portali B2B e realtà aumentata;
- **TCE:** si dedica alla semplificazione delle fasi di preventivazione e acquisizione ordini con il prodotto *CPQ*, che permette di configurare prodotti e servizi in modo rapido e preciso;
- **DISCOVERY QUALITY:** sviluppa *software* per la *Governance* aziendale, controllo dei processi e misurazione delle performance, con un occhio alle normative e metriche di sostenibilità (*Sustainable Development Goals (SDGs)*, *Benefit Corporation (BCorp)*), per garantire la qualità dei prodotti e servizi;
- **ECM:** propone soluzioni di *Enterprise Content Management (ECM)* per la gestione efficace dei documenti digitali, fornendo strumenti per la gestione dei contenuti, la collaborazione e la condivisione dei documenti;
- **SMITECH:** focalizzata sulla *Cybersecurity* e protezione dei dati, offrendo servizi di consulenza, formazione e soluzioni tecnologiche per la sicurezza informatica;
- **ELEMENT:** è la divisione creativa per lo sviluppo di siti *web* ed *e-commerce*, con un focus sull'esperienza utente e l'interfaccia grafica;
- **JPA:** sviluppa *software* di *Business Process Management (BPM)* per l'automazione e integrazione dei processi aziendali, sviluppa una piattaforma per la gestione dei processi aziendali, fornendo un *designer* grafico per la modellazione dei processi, un motore di esecuzione per l'esecuzione dei processi ed un'interfaccia grafica per l'esecuzione dei *task* assegnati agli utenti;
- **FACTORY:** risponde alle esigenze della *Supply Chain* con soluzioni per la fabbrica del futuro, mirate a ottimizzare il servizio clienti, *asset* e profitti. Offre anche soluzioni per la gestione dei magazzini e per la gestione della produzione;
- **JPM:** offre soluzioni di *Project Management* per la gestione dei progetti, fornendo strumenti per la pianificazione, il monitoraggio e il controllo dei progetti che lavorano a commessa o a preventivo.

1.1.3 Strategie e obiettivi

Il piano strategico triennale di Sanmarco informatica S.p.a., battezzato *Vision 2025*, è un manifesto per la crescita sostenibile, l'innovazione continua e la piena soddisfazione del cliente. Il piano si colloca tra ambizione commerciale e responsabilità sociale, mirando a rafforzare la posizione di Sanmarco informatica S.p.a. come *leader* nel settore informatico, sia a livello nazionale che internazionale.

L'obiettivo principale di *Vision 2025* è di espandere la penetrazione di mercato di Sanmarco informatica S.p.a., non solo consolidando la sua presenza nei mercati italiani, ma anche esplorando nuove opportunità in ambito internazionale. Un'attenzione particolare è rivolta ai mercati emergenti e in rapida espansione, come quelli nordamericani e asiatici, dove la domanda di soluzioni informatiche innovative è in costante crescita. Per realizzare questa visione, Sanmarco informatica S.p.a. si impegna in un investimento strategico nelle tecnologie più avanzate, includendo l'adozione di nuove piattaforme *software*, l'aggiornamento delle infrastrutture esistenti e l'esplorazione di nuove frontiere

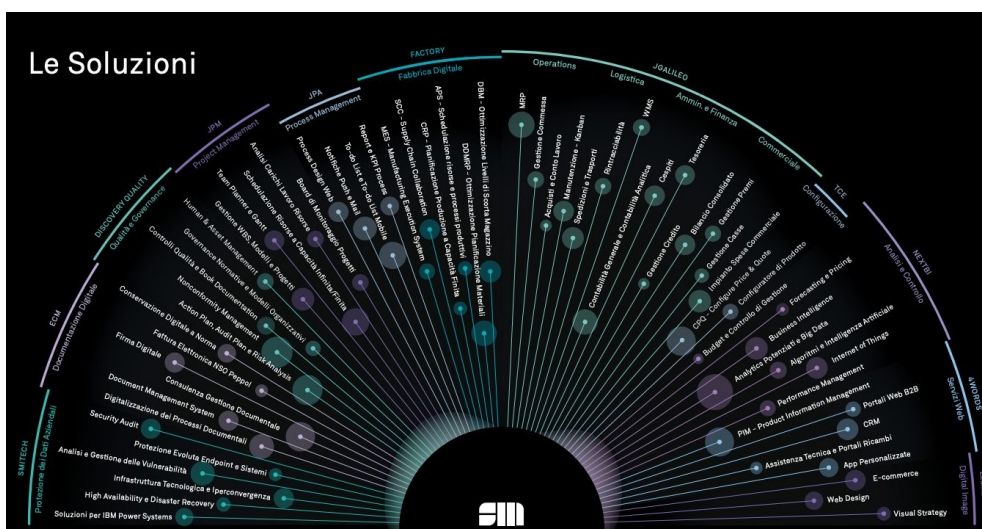


Figura 1.1: Le *Business Unit* di Sanmarco informatica S.p.a. ed i loro Prodotti. Fonte: <https://www.sanmarcoinformatica.it/intranet.pag>

nell'ambito dell'intelligenza artificiale e del big data. Inoltre, un'enfasi particolare è posta sul potenziamento delle competenze dei *team* di Sanmarco informatica S.p.a., attraverso programmi di formazione continua e l'assunzione di talenti nel campo dell'informatica, garantendo che l'azienda rimanga all'avanguardia nell'innovazione tecnologica.

Parallelamente, Sanmarco informatica S.p.a. si impegna a promuovere la responsabilità sociale d'impresa. Questo si traduce nel rispetto dei **SDGs** e nella ricerca della certificazione **BCorp**, che riconosce le aziende per il loro impatto positivo sia sociale che ambientale. L'impegno riflette la consapevolezza di Sanmarco informatica S.p.a. sull'importanza di operare in modo etico e sostenibile, riconoscendo che il successo aziendale che va di pari passo con il benessere della comunità e dell'ambiente.

Infine, Sanmarco informatica S.p.a. prevede di ampliare il proprio portafoglio prodotti, introducendo soluzioni *software* innovative che rispondano alle esigenze in continua evoluzione del mercato. Questo include lo sviluppo di nuove applicazioni, l'aggiornamento e il miglioramento delle soluzioni esistenti e l'espansione in nuovi segmenti di mercato. L'obiettivo è di garantire che ogni cliente riceva soluzioni personalizzate che offrano il massimo valore aggiunto, consolidando così la posizione di Sanmarco informatica S.p.a. come *leader* nel settore delle soluzioni informatiche.

La *Business unit*

Nell'ecosistema di Sanmarco informatica S.p.a., le *Business Unit* (BU) rappresentano, non solo divisioni funzionali, ma veri e propri centri di innovazione e sviluppo agile. Ogni BU, con la sua struttura unica, è progettata per rispondere dinamicamente ai cambiamenti del mercato informatico, enfatizzando la collaborazione e la flessibilità. L'approccio agile si traduce in una consegna incrementale di valore, dove il *feedback* continuo e l'adattamento alle esigenze emergenti sono fondamentali.

La gestione dei progetti all'interno delle BU è un esercizio di equilibrio tra innovazione e efficienza. I responsabili di progetto giocano un ruolo cruciale, bilanciando il *budget*,

le risorse umane e le aspettative dei clienti. La gestione richiede una comprensione profonda delle tecnologie emergenti e delle metodologie di sviluppo *software*, assicurando che ogni progetto, non solo rispetti i tempi e i costi, ma sia anche all'avanguardia in termini di soluzioni tecnologiche.

All'interno di ogni BU, il **Product Owner (PO)** è la figura chiave che assicura l'allineamento del progetto con le visioni e gli obiettivi del cliente. Lavorando a stretto contatto con gli sviluppatori, il PO traduce le esigenze del cliente in requisiti tecnici, garantendo che il prodotto finale soddisfi o superi le aspettative. Lo **Scrum Master**, d'altra parte, si concentra sull'ottimizzazione dei processi di sviluppo, assicurando che il *team* adotti le migliori pratiche agili e mantenga un alto livello di produttività.

Gli sviluppatori, con le loro competenze tecniche, sono il motore che alimenta l'innovazione all'interno di Sanmarco informatica S.p.a.. Lavorando su una vasta gamma di tecnologie, da Java e *Angular* a soluzioni *cloud* e *mobile*, sono in prima linea nel trasformare le idee in realtà tangibili. I *tester*, collaborando strettamente con gli sviluppatori, assicurano che ogni prodotto sia robusto e privo di errori, un aspetto cruciale in un settore dove la qualità del *software* è direttamente correlata alla soddisfazione del cliente.

I consulenti e gli **analisti** svolgono un ruolo fondamentale nell'interpretare le esigenze del cliente e nel tradurle in specifiche tecniche. Il lavoro di analisi è vitale per garantire che i progetti siano allineati con le aspettative del cliente e con le tendenze del mercato. Oltre alle BU, la struttura organizzativa di Sanmarco informatica S.p.a. comprende vari dipartimenti che supportano le operazioni quotidiane. Il dipartimento delle risorse umane si occupa di attrarre e mantenere talenti, essenziale in un settore in rapida evoluzione come l'informatica. Il *marketing*, attraverso strategie digitali e tradizionali, posiziona Sanmarco informatica S.p.a. sul mercato, mentre l'Amministrazione garantisce la solidità finanziaria. Il dipartimento **Information Technology (IT)**, con il suo ruolo di supporto e innovazione, è il cuore tecnologico dell'azienda, garantendo che l'infrastruttura informatica sia sempre efficiente e all'avanguardia.

In conclusione, la struttura di Sanmarco informatica S.p.a. è un tessuto complesso di competenze e funzioni, tutte orientate verso l'obiettivo comune di eccellenza nel settore informatico. Questa sinergia tra le diverse unità e dipartimenti è ciò che permette a Sanmarco informatica S.p.a. di rimanere competitiva e innovativa in un mercato in costante evoluzione.

Monitoraggio delle attività

La struttura lavorativa in Sanmarco informatica S.p.a. è caratterizzata da una flessibilità che si adatta alle esigenze moderne del settore informatico. Con un *team* distribuito che include dipendenti in sede, personale in lavoro remoto e professionisti che operano direttamente presso i clienti, Sanmarco informatica S.p.a. adotta un approccio moderno e versatile alla gestione del lavoro. La diversità nelle modalità lavorative riflette la dinamicità del settore IT e la necessità di un approccio agile e personalizzato per ogni progetto.

Per coordinare efficacemente la forza lavoro distribuita, Sanmarco informatica S.p.a. si affida a un sofisticato *software* interno di *time tracking*. Questo strumento è fondamentale per garantire trasparenza e precisione nella registrazione delle ore lavorative. Ogni dipendente, dotato di un proprio *account* personale, registra le ore dedicate a specifici progetti, fornendo così una visione chiara del tempo impiegato in ogni attività. Il sistema, non solo facilita la gestione amministrativa, ma è anche uno strumento prezioso per la pianificazione e l'allocazione delle risorse.

Il processo di inserimento delle ore lavorate è dettagliato e strutturato per catturare tutte le informazioni rilevanti. Durante la compilazione del "rapportino", il dipendente inserisce una descrizione dettagliata delle attività svolte, specificando la commessa, l'eventuale cliente, la sede di lavoro e gli orari di inizio e fine attività. La procedura, che deve essere completata quotidianamente, permette di collegare ogni ora lavorata a specifici *ticket* o progetti, assicurando una tracciabilità completa e una gestione efficiente del lavoro.

Al termine di ogni mese, il sistema blocca le ore registrate, consentendo ai responsabili di progetto e al dipartimento amministrativo di analizzare i dati raccolti. Questi report mensili sono essenziali per valutare la produttività, pianificare le risorse future e ottimizzare i processi lavorativi. Inoltre, il sistema di *time tracking* gioca un ruolo cruciale nella trasparenza verso i clienti, fornendo una base solida per la fatturazione e la rendicontazione delle attività svolte.

In sintesi, la gestione del lavoro in Sanmarco informatica S.p.a. è un esempio di come le tecnologie moderne possano essere impiegate per ottimizzare la gestione delle risorse umane in un contesto lavorativo complesso e diversificato. Questo approccio, non solo migliora l'efficienza operativa, ma contribuisce anche a una maggiore soddisfazione dei dipendenti e dei clienti, elementi fondamentali per il successo nel settore informatico.

1.2 Il team di sviluppo

Durante il mio periodo in *JPA (Process Management)*, ho avuto l'opportunità di osservare da vicino il lavoro di un *team* di sviluppo particolarmente versatile. Il *team*, a differenza di quelli più tradizionali focalizzati su un singolo prodotto, aveva l'obiettivo di fornire supporto globale a tutti i *team* di sviluppo dell'azienda. Il supporto tecnico e analitico era la principale attività del *team*, affrontando e risolvendo problemi complessi per agevolare il lavoro degli altri *team*. Questo ruolo cruciale implicava l'identificazione e la soluzione di sfide tecniche, assicurando un ambiente di sviluppo efficiente e senza ostacoli.

Un altro aspetto centrale del lavoro del *team* era la formazione. Con l'evolversi continuo delle tecnologie e degli strumenti, era essenziale mantenere i *team* aggiornati. Di conseguenza, il *team* organizzava regolarmente sessioni di formazione per condividere conoscenze e competenze su nuove tecnologie e metodologie di sviluppo.

In parallelo, il *team* era impegnato in attività di ricerca e sviluppo, in particolare nello sviluppo di un *framework* interno. Il framework, un insieme di librerie e strumenti per lo sviluppo di applicazioni *web*, era progettato per rendere la creazione di applicazioni *web* più semplice e veloce, contribuendo significativamente all'efficienza dello sviluppo *software* in azienda.

L'automazione dei processi di sviluppo era un'altra area chiave, includeva l'automazione della compilazione, il rilascio dei prodotti e lo sviluppo di nuove funzionalità, riducendo il tempo e lo sforzo necessari per le operazioni di *routine*.

La gestione dei *repository* di codice sorgente e il supporto all'uso di strumenti di *continuous integration* erano compiti fondamentali del *team*; assicurava una gestione efficace del codice e un'integrazione continua, elementi vitali per mantenere la qualità e l'affidabilità del *software*.

Infine, il *team* era responsabile dello sviluppo di un installatore per i prodotti dell'azienda basati sul *framework* interno. Lo strumento semplificava il processo di installazione, rendendo i prodotti più accessibili agli utenti finali.

Il *team* era composto da tre persone: uno *Scrum Master* e due sviluppatori. In questo

ambiente dinamico, i ruoli erano fluidi: gli sviluppatori svolgevano anche compiti di analisi e test, e lo [Scrum Master](#) partecipava attivamente alle analisi tecniche e funzionali. La struttura flessibile e collaborativa era essenziale per il successo del *team* e per il supporto efficace fornito agli altri *team* di sviluppo.

1.3 Strumenti utilizzati

Durante il mio percorso di sviluppo, ho avuto l'opportunità di utilizzare una serie di strumenti e tecnologie all'avanguardia (figura 1.2), che sono stati essenziali per facilitare vari aspetti del processo di sviluppo, dalla scrittura del codice alla gestione dei progetti. Per la documentazione, l'analisi dei requisiti e la progettazione delle basi di dati, ho utilizzato *Confluence*. Questo *software* di collaborazione si è rivelato estremamente efficace nel creare, organizzare e condividere documenti di progetto.

Una volta completata l'analisi e creato le *issues* su *Jira*, mi sono dedicato allo sviluppo del codice, utilizzando *IntelliJ IDEA* per il *backend* e *WebStorm* per il *frontend*. Questi due [IDE](#) hanno notevolmente semplificato il processo di sviluppo, migliorando sia la produttività che la qualità del codice prodotto.

Per la gestione del codice sorgente, ho scelto *Git*, un sistema di controllo versione distribuito, e *Bitbucket*, un servizio di *hosting* per progetti *Git*, che insieme hanno fornito una soluzione robusta e affidabile. Inoltre, per la gestione dei *database* a grafo, ho impiegato *Neo4j Desktop*, un'applicazione che facilita la creazione, gestione e monitoraggio delle prestazioni dei *database* *Neo4j*.

Il processo di compilazione e *testing* è stato ottimizzato grazie all'uso di *Gradle*, un sistema di automazione che ha ridotto i tempi di rilascio migliorando la coerenza e l'affidabilità delle *build*. Per il *deployment*, ho utilizzato *Docker*, una piattaforma che ha rivoluzionato il nostro approccio, permettendo di impacchettare e distribuire applicazioni in ambienti isolati, assicurando coerenza tra gli ambienti di sviluppo, *testing* e produzione.

Infine, per l'integrazione continua, ho scelto *Jenkins*, uno strumento che ha automatizzato il ciclo di vita del *software*, dalla *build* al *testing* fino al *deployment*, aumentando la velocità di rilascio e riducendo la possibilità di errori.

I linguaggi di programmazione utilizzati hanno incluso:

- **Java:** La mia principale lingua di programmazione, utilizzata per lo sviluppo di applicazioni robuste e ad alte prestazioni, sia *web* che *desktop*;
- **JavaScript:** Fondamentale per lo sviluppo *frontend*, ha permesso di creare interfacce utente interattive e dinamiche;
- **TypeScript:** Utilizzato per aggiungere tipizzazione statica a JavaScript, migliorando la leggibilità e la manutenibilità del codice;
- **Groovy:** Impiegato per script e automazioni, sfruttando la sua compatibilità con la [Java Virtual Machine \(JVM\)](#) e la sua sintassi espressiva;

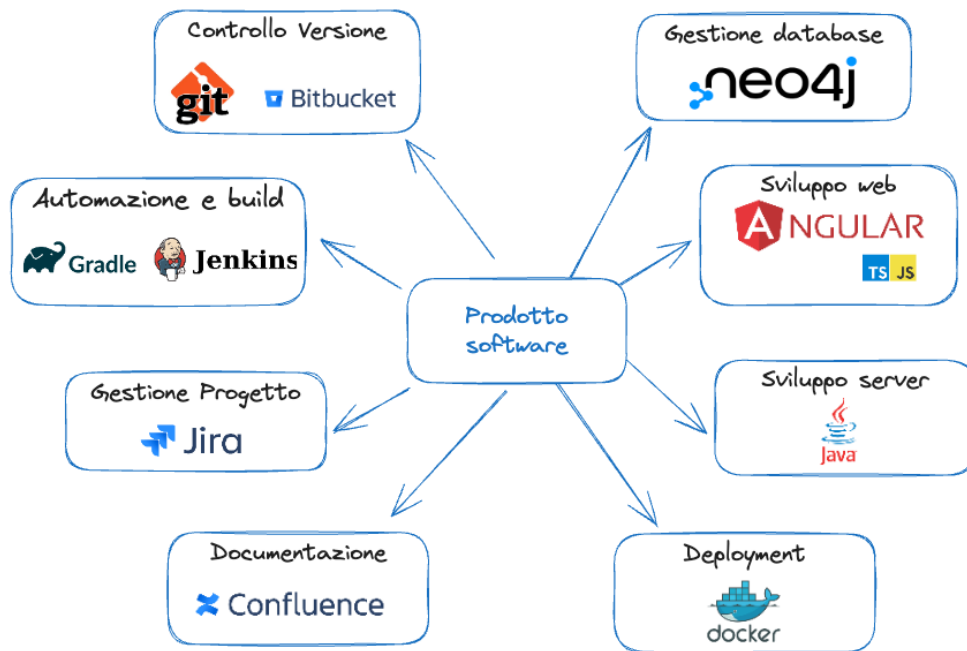


Figura 1.2: Strumenti utilizzati

- **Cypher:** Il linguaggio di *query* per Neo4j, essenziale per interrogare e manipolare i dati nei nostri *database* a grafo.

Questi strumenti e linguaggi hanno formato il nucleo della mia cassetta degli attrezzi di sviluppo, permettendomi di affrontare una vasta gamma di sfide tecniche e di contribuire efficacemente ai progetti dell'azienda.

1.3.1 Convenzioni

Nel corso dello sviluppo dei progetti che impiegano il *framework* interno, vengono adottate una serie di convenzioni standardizzate. Le convenzioni, archiviate in *Confluence* per un facile accesso da parte di tutti i dipendenti, sono state pensate per garantire coerenza, efficienza e qualità nel lavoro. Sono categorizzate come segue:

- **Documentazione:** Regole che stabiliscono come documentare efficacemente il codice sorgente. L'obiettivo è assicurare che ogni segmento di codice sia accompagnato da commenti chiari e concisi, che ne spieghino la funzione e la logica. Questo approccio, non solo facilita la comprensione del codice da parte di altri sviluppatori, ma è anche fondamentale per la manutenzione a lungo termine del *software*;
- **Scrittura analisi:** Linee guida delineano il metodo per redigere analisi dei requisiti e progettazione delle strutture di basi di dati. L'obiettivo è garantire che le analisi siano scritte in modo chiaro e strutturato, facilitando la comprensione e la comunicazione tra i membri del *team* e con i clienti;
- **Progettazione:** Regole forniscono indicazioni su come progettare i componenti *software*. L'enfasi è posta sulla creazione di *design* modulari e riutilizzabili,

che facilitano la manutenzione e l'estensione del codice nel tempo. Questo approccio aiuta a ridurre la complessità del codice e a migliorare la scalabilità delle applicazioni;

- **Codifica:** Convenzioni che mirano a standardizzare lo stile di codifica. L'obiettivo è scrivere codice in modo uniforme, seguendo un insieme di regole che ne migliorino la leggibilità e la comprensione. Questo include convenzioni su nomi di variabili, strutture di controllo, formattazione del codice e commenti;
- **Versionamento:** Regole stabiliscono come gestire il versionamento del codice sorgente. L'obiettivo è facilitare la tracciabilità delle modifiche e la gestione delle diverse versioni del *software*. Questo è cruciale per il controllo qualità, la risoluzione dei *bug* e la collaborazione tra i membri del *team*.

Adottare convenzioni ha migliorato significativamente la qualità e l'efficienza del processo di sviluppo. Hanno fornito una base solida per la collaborazione e la standardizzazione, elementi chiave per il successo dei nostri progetti *software*.

1.4 Rapporto con l'innovazione

Sanmarco informatica S.p.a. si impegna costantemente nell'innovazione delle aziende clienti, giocando un ruolo cruciale nella loro trasformazione digitale. Specializzata nella progettazione e realizzazione di soluzioni integrate, l'azienda si dedica alla riorganizzazione dei processi aziendali e professionali, mirando a un impatto significativo e misurabile.

Per perseguire questo obiettivo, Sanmarco informatica S.p.a. investe una quota sostanziale del proprio fatturato, tra il 15 e il 20%, in Ricerca e Sviluppo ogni anno. L'investimento testimonia l'impegno dell'azienda nel rimanere all'avanguardia nel settore tecnologico, garantendo l'innovazione continua dei suoi prodotti e servizi.

Un aspetto distintivo di Sanmarco informatica S.p.a. è la sua capacità di ascoltare e valorizzare le idee provenienti da clienti, dipendenti e collaboratori. Questo approccio collaborativo è fondamentale per l'ispirazione e lo sviluppo di nuovi prodotti e soluzioni innovative. Attualmente, quasi tutti i prodotti installati presso i clienti sono in fase di aggiornamento, dimostrando l'impegno dell'azienda nel fornire soluzioni sempre aggiornate e allineate con le ultime tendenze tecnologiche.

L'investimento in cultura e formazione è un altro pilastro fondamentale per Sanmarco informatica S.p.a.. Ogni anno, l'azienda organizza una serie di corsi di formazione per i propri dipendenti, consentendo loro di acquisire competenze su nuove tecnologie e strumenti. I corsi sono tenuti sia da membri esperti del team interno, sia da consulenti esterni, e spesso si avvalgono di piattaforme di *e-learning* come *Udemy Business*, fornite gratuitamente dall'azienda.

In aggiunta, Sanmarco informatica S.p.a. promuove attivamente l'innovazione attraverso l'organizzazione di eventi, come il *Choose Innovation* in collaborazione con *IBM*. Gli eventi rappresentano un'opportunità per discutere di innovazione e di come le aziende possano adottare nuove strategie per rimanere competitive nel mercato in rapida evoluzione. Attraverso questi sforzi, Sanmarco informatica S.p.a., non solo rafforza la propria posizione come *leader* nell'innovazione, ma contribuisce anche attivamente all'avanzamento del settore.

Capitolo 2

Il progetto di *stage*

In questo capitolo descrivo il progetto di *stage*, e le motivazioni che mi hanno spinto a scegliere questo progetto.

Ho scelto Sanmarco informatica S.p.a. perchè è l'azienda per cui lavoro da più di 5 anni, e che mi ha dato la possibilità di crescere professionalmente. Ho scelto questo progetto perchè mi ha permesso di lavorare con tecnologie nuove, e di imparare nuovi linguaggi di programmazione.

Lo *stage* prevedeva lo sviluppo di un prototipo per il monitoraggio, la raccolta e l'analisi delle dipendenze *software* degli applicativi di Sanmarco informatica S.p.a..

Il nome del progetto è *Dependency Analyzer* ed è stato scelto dai membri del team di cui ho fatto parte.

2.1 Lo *stage* per Sanmarco informatica S.p.a.

Gli *stage* rappresentano un pilastro fondamentale nella strategia di Sanmarco informatica S.p.a., svolgendo molteplici funzioni cruciali. Primo fra tutti, offrono l'opportunità di dedicarsi a progetti innovativi che, a causa di limitazioni di *budget* o di tempo, non troverebbero spazio nelle attività lavorative ordinarie. I *team* di sviluppo di Sanmarco informatica S.p.a. sono primariamente impegnati nel mantenimento e nello sviluppo dei prodotti esistenti, lasciando poco margine per la ricerca e lo sviluppo di nuove idee. Questo compito solitamente è affidato al *team* di ricerca e sviluppo, che, nonostante la sua competenza, è spesso limitato dalla scarsità di risorse umane per affrontare tutte le richieste innovative. In questo contesto, lo *stage* diventa una soluzione efficace, consentendo lo sviluppo di prototipi e la conduzione di ricerche senza gravare eccessivamente sulle risorse aziendali.

L'investimento principale per l'azienda risiede nel tempo dedicato dal tutor aziendale alla guida e al supporto dello stagista. Pertanto, una selezione accurata sia del tutor sia dello stagista è essenziale per il successo del progetto e il raggiungimento degli obiettivi entro i tempi stabiliti.

Da anni, Sanmarco informatica S.p.a. ha instaurato una proficua collaborazione con l'Università di Padova, accogliendo studenti per lo svolgimento di tesi di laurea e *stage*. Questi periodi di formazione in azienda sono spesso orientati verso l'assunzione: negli ultimi anni, più di 140 stagisti sono entrati a far parte del team di Sanmarco informatica S.p.a. a seguito del loro *stage*.

Per lo studente, lo *stagesi* configura come un'esperienza formativa di grande valore. Consente di applicare concretamente le conoscenze teoriche acquisite durante il percorso di studi in un contesto lavorativo reale, favorendo al contempo l'acquisizione di nuove competenze e conoscenze. Questo periodo rappresenta un momento di verifica importante per lo studente, che ha l'opportunità di confrontarsi con il mondo del lavoro e di valutare se l'attività svolta corrisponde alle proprie aspettative e aspirazioni professionali future.

2.2 La proposta di *stage*

Un obiettivo strategico futuro per Sanmarco informatica S.p.a. consiste nella modularizzazione dei propri prodotti *software*. L'intento è quello di realizzare moduli che possano essere utilizzati in maniera autonoma e che, al contempo, siano facilmente integrabili con altre soluzioni. Questa direzione strategica implica un significativo aumento del numero di progetti e delle interdipendenze tra essi, rendendo essenziale un'efficace gestione e monitoraggio di tali dipendenze.

In risposta a questa esigenza, il dipartimento di Ricerca e Sviluppo di Sanmarco informatica S.p.a. ha intrapreso l'iniziativa di sviluppare un sistema avanzato per la raccolta, il monitoraggio e l'analisi delle dipendenze *software*. L'obiettivo è quello di garantire un controllo accurato sulle versioni delle dipendenze collegate alle diverse release dei prodotti, contribuendo così a soddisfare i requisiti non funzionali in termini di sicurezza e affidabilità delle soluzioni offerte.

Nell'ambito di questa iniziativa, la proposta di *stage* riguarda lo sviluppo di un *prototipo* per tale strumento. Il focus del progetto di *stage* è stato lo sviluppo di un sistema in grado di raccogliere informazioni dettagliate sulle dipendenze direttamente dai sistemi di *build* utilizzati per la compilazione dei prodotti, specificatamente *Gradle* e *npm*. Questo prototipo rappresenta un passo fondamentale verso l'implementazione di una soluzione completa per la gestione delle dipendenze.

Il plugin *Gradle*

Il primo prodotto che l'azienda intendeva realizzare tramite lo *stage* è un *plugin* per *Gradle* che raccoglie informazioni sulle dipendenze dei progetti java, analizzando l'albero delle dipendenze generato da *Gradle*, e dei progetti *npm* analizzandone il file *package-lock.json* generato durante l'installazione dei pacchetti. Il plugin doveva poter essere pubblicato su un *repository* interno aziendale, e doveva essere utilizzabile da tutti i progetti di Sanmarco informatica S.p.a.. Per essere più flessibile, doveva poter essere configurato specificando i nomi dei progetti *npm* da analizzare ed i riferimenti al *backend* per l'invio delle informazioni raccolte.

Il backend

Il secondo prodotto atteso era un *backend* per il salvataggio e l'analisi delle informazioni raccolte. Il *backend* doveva esporre dei servizi *REST* per l'interrogazione del sistema, e doveva essere sviluppato utilizzando il linguaggio di programmazione *Java*. Non è stato richiesto l'utilizzo di un *framework* specifico, ma è stata lasciata la libertà di scegliere il *framework* più adatto al progetto.

Il *plugin* dopo aver raccolto le informazioni le invia al *backend* che, dopo averle elaborate, le salva in un *database* a grafo. Anche in questo caso è stato richiesto un file

di configurazione per specificare i riferimenti al *database* a grafo, e le credenziali per l'autenticazione ai servizi *REST*.

Il frontend

Infine, un'interfaccia grafica realizzata tramite una *web-app* per la visualizzazione delle informazioni raccolte. Le specifiche per il *frontend* erano molto generiche, ma per essere in linea con le tecnologie utilizzate in azienda, è stato richiesto di utilizzare il *framework Angular* per lo sviluppo.

2.3 Obiettivi e aspettative

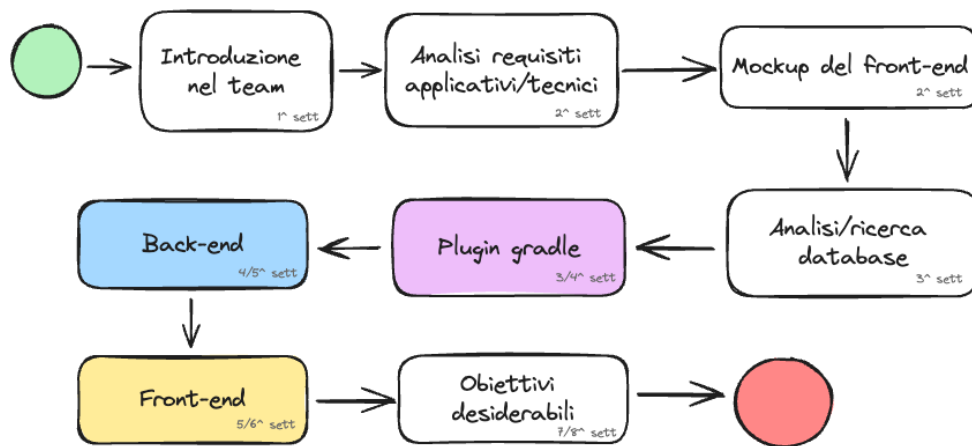


Figura 2.1: Sequenza temporale delle attività scritte nel piano di lavoro.

L'identificazione degli obiettivi e delle aspettative è un passo fondamentale per la definizione di un progetto di *stage*. In questo contesto, gli obiettivi rappresentano le funzionalità che il sistema deve implementare, mentre le aspettative sono le caratteristiche che il sistema deve possedere.

Gli obiettivi e le aspettative del progetto di *stage* sono stati definiti in collaborazione con il tutor aziendale, tramite il piano di lavoro. Esso è stato redatto prima dell'inizio dello *stage*, e ha rappresentato un punto di riferimento per la valutazione delle attività svolte durante il periodo di *stage*.

All'interno del piano di lavoro erano presenti obiettivi obbligatori e obiettivi desiderabili. Gli obiettivi avevano un codice identificativo, e una descrizione.

Il codice identificativo era composto da una lettera che rappresentava la tipologia di obiettivo, e da un numero progressivo. Il progressivo ha permesso l'ordinamento, non casuale, degli obiettivi.

La pianificazione settimanale scritta nel piano di lavoro, e riportata in figura 2.1, è stata una vera e propria guida per lo svolgimento delle attività, ed ha permesso di rispettare le scadenze, raggiungendo gli obiettivi prefissati in ordine di priorità, portando quelli desiderabili a termine solo se il tempo lo permetteva.

Nel piano di lavoro gli obiettivi non erano molto dettagliati (tabella 2.1 e tabella 2.2), questo ci ha permesso di avere una certa flessibilità durante lo svolgimento delle attività, e di poter cambiare gli obiettivi in corso d'opera, se necessario.

Tabella 2.1: Obiettivi obbligatori

Identificativo	Descrizione
O01	Analisi dell'attuale infrastruttura
O02	Analisi requisiti applicativi e tecnici e <i>mockup</i> del <i>front end</i>
O03	Identificazione del db a grafo più adeguato al progetto
O04	Implementazione della struttura di <i>database</i>
O05	Implementazione di un <i>plugin Gradle</i> per la raccolta delle informazioni
O06	Invocazione da <i>jenkins</i>
O07	Implementazione dei servizi <i>backend</i> per l'interrogazione del sistema
O08	Implementazione del <i>front end</i> per rendere possibile l'interrogazione

Tabella 2.2: Obiettivi desiderabili

Identificativo	Descrizione
D01	Possibilità di visualizzare i risultati in forma grafica (grafo) e non solo tabellari
D02	Integrazione con <i>repository</i> remoti al fine di verificare nuove versioni delle dipendenze
D03	Integrazione con <i>repository</i> remoti al fine di identificare vulnerabilità sulle dipendenze utilizzate
D04	Login con <i>LDAP</i> aziendale

2.4 Vincoli

Vincoli tecnologici

Il piano di lavoro tramite gli obiettivi, intrinsecamente, ha definito i vincoli tecnologici del progetto.

Il primo vincolo è stato quello di utilizzare un *database* a grafo per il salvataggio dei dati raccolti dal *plugin*, questo perchè permette di rappresentare le dipendenze in maniera naturale, e di eseguire *query* molto complesse in tempi molto brevi.

Con i membri del team abbiamo deciso di utilizzare *Neo4j* come *database* a grafo, perchè è il più utilizzato, ha una comunità molto attiva ed è molto ben documentato.

Il secondo vincolo tecnologico è stato quello di utilizzare *Gradle* per il *plugin* che raccoglie le informazioni sulle dipendenze, questo perchè quasi tutti i progetti di San-marco informatica S.p.a. lo utilizzano come sistema di build.

Gradle

Gradle è uno strumento di automazione della *build open source* che si è guadagnato una notevole popolarità nel mondo dello sviluppo *software*, grazie alla sua flessibilità e potenza. Utilizzato principalmente per progetti *Java*, è anche ampiamente adottato per applicazioni scritte in altri linguaggi di programmazione come *Kotlin*, *C++*, e *Python*. La sua caratteristica principale è la capacità di supportare configurazioni di *build* altamente personalizzabili, rendendolo adatto sia per piccoli progetti sia per grandi imprese con esigenze complesse.

Gradle si distingue per l'uso di un *DSL* basato sui linguaggi di programmazione *Groovy* o *Kotlin*, che fornisce un modo intuitivo e dichiarativo di definire le *build*. Questo approccio, combinato con la sua potente capacità di gestione delle dipendenze e il suo modello di esecuzione incrementale, consente agli sviluppatori di costruire *software* in modo più efficiente e affidabile. Inoltre, *Gradle* è noto per la sua velocità, superando spesso altri strumenti di *build* come *Maven* e *Ant*, specialmente in progetti di grandi dimensioni, la figura 2.2 mostra alcune differenze in termini di velocità tra *Gradle* e *Maven*.

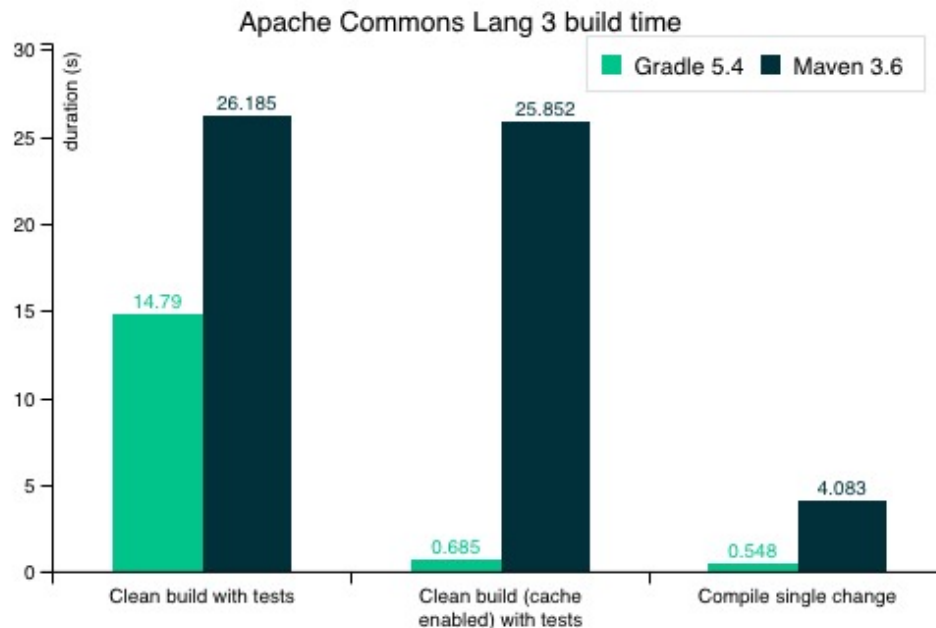


Figura 2.2: Comparazione tra *Gradle* e *Maven*. Fonte: <https://Gradle.org/maven-vs-Gradle>

Un altro aspetto fondamentale di *Gradle* è la sua estensibilità. Gli sviluppatori possono estendere le funzionalità di *Gradle* scrivendo script personalizzati o integrando *plugin* esistenti. Questa flessibilità lo rende uno strumento ideale per adattarsi a flussi di lavoro specifici e requisiti di *build* unici. *Gradle* è anche il sistema di *build* ufficiale per *Android*, ulteriormente consolidando la sua posizione come uno strumento chiave nello sviluppo di applicazioni moderne.

Angular

Un altro vincolo tecnologico è stato quello di utilizzare *Angular* per lo sviluppo del *frontend*, questo perché, come scritto precedentemente, è il *framework* più utilizzato in azienda per lo sviluppo delle *web-app*.

Angular è un *framework front-end open source* sviluppato e mantenuto da *Google*. È ampiamente riconosciuto per la sua capacità di creare applicazioni *web* dinamiche e reattive. *Angular* utilizza *TypeScript*, una *super-set* di *JavaScript*, che fornisce funzionalità di tipizzazione statica e orientamento agli oggetti, migliorando la manutenibilità e la qualità del codice.

Una delle caratteristiche distintive di *Angular* è il suo approccio basato su componenti, che aiuta gli sviluppatori a costruire applicazioni *web* complesse in modo più modulare e mantenibile. Ogni componente in *Angular* è una combinazione di *HTML*, *CSS*, e *TypeScript*, che gestisce una parte specifica dell'interfaccia utente. Questo approccio promuove la riusabilità del codice e una migliore separazione delle preoccupazioni.

Angular è anche noto per il suo potente sistema di *dependency injection*, che semplifica lo sviluppo e il *testing* fornendo un modo per iniettare dipendenze in classi in modo pulito e flessibile. Inoltre, *Angular* offre un'ampia gamma di funzionalità integrate come il *routing*, la gestione delle *form*, e l'accesso HTTP, che accelerano lo sviluppo di applicazioni *web*.

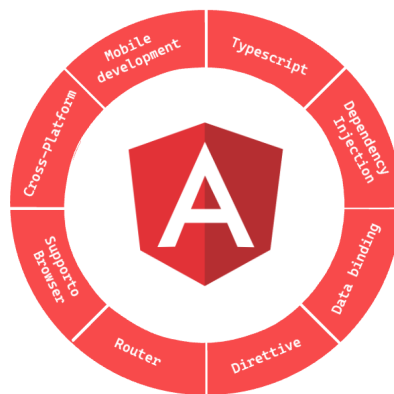


Figura 2.3: Caratteristiche di *Angular*.

Con il suo ecosistema ricco (figura 2.3) e una forte comunità di sviluppatori, *Angular* è diventato uno dei *framework front-end* più popolari e affidabili per lo sviluppo di applicazioni *web* moderne, sia per progetti di piccole dimensioni sia per applicazioni aziendali di grande scala.

I database a grafo e *Neo4j*

I *database* a grafo sono una categoria di *database* NoSQL progettati per gestire e rappresentare dati complessi e le loro relazioni in modo più efficiente rispetto ai tradizionali *database* relazionali, la figura 2.4 mostra alcune differenze tra i *database* relazionali e quelli a grafo.

Questi *database* utilizzano strutture di grafo per la memorizzazione semantica, con nodi,

bordi e proprietà per rappresentare e memorizzare dati. La flessibilità dei *database* a grafo li rende particolarmente adatti per applicazioni che richiedono l'analisi di relazioni complesse e interconnesse, come i *social network*, i sistemi di raccomandazione, e la gestione delle reti.

Neo4j è uno dei più popolari *database* a grafo, noto per la sua alta *performance* e flessibilità. È un *database open source*, ma offre anche una versione *enterprise* con funzionalità aggiuntive. *Neo4j* utilizza il linguaggio di *query Cypher*, che è specificamente progettato per lavorare con grafi. *Cypher* consente agli sviluppatori di esprimere facilmente *query* complesse sulle relazioni tra i dati, rendendo *Neo4j* particolarmente potente per analisi di dati relazionali complessi.

	Database relazionali	Database grafici
Modello	Tabellare con righe e colonne.	Nodi interconnessi con dati rappresentati come documenti JSON.
Operatività	Operazioni SQL come creazione, lettura, aggiornamento ed eliminazione (CRUD).	Le operazioni includono CRUD e operazioni di attraversamento di grafi basate sulla teoria matematica dei grafi.
Scalabilità	I database relazionali tradizionali possono dimensionare verticalmente ma faticano a dimensionare orizzontalmente.	Un database a grafo eccelle nel dimensionamento orizzontale. Può utilizzare il partizionamento per distribuire i dati su molti nodi.
Prestazioni	I database relazionali spesso devono gestire query complesse che coinvolgono attraversamenti di relazioni, il che può avere un impatto negativo sulle prestazioni.	Un database a grafo eccelle nel rappresentare e interrogare le relazioni tra i dati.
Facilità d'uso	I database relazionali funzionano bene con set di dati di grandi dimensioni e dati strutturati. Hanno difficoltà con le query multi-hop.	Quando si lavora con dati basati sulle relazioni, l'utilizzo di un database a grafo risulta semplice e intuitivo. Utilizzando un linguaggio di query a grafo, è possibile eseguire query su più nodi di dati in modo rapido ed efficiente.

Figura 2.4: Differenze tra *database* a grafo e *database* relazionali.

Fonte: <https://aws.amazon.com/it/compare/the-difference-between-graph-and-relational-database>.

Vincoli di dominio

Il prodotto finale dovrà essere utilizzato, non solo dagli sviluppatori, ma anche dai *project manager*, i *team leader*, e i *product owner* ed i consulenti; Per questo è stato richiesto, come obiettivo desiderabile, di implementare l'autenticazione tramite *LDAP* aziendale, per permettere l'accesso a tutti i dipendenti di Sanmarco informatica S.p.a.. L'*LDAP* è un protocollo di rete che permette la gestione centralizzata delle autenticazioni e delle autorizzazioni. Questo è fondamentale per garantire la sicurezza dei dati, e per permettere l'accesso solo a chi ne ha i permessi, senza dover implementare un sistema di autenticazione ad hoc.

2.5 La scelta dello *stage*

La scelta dello *stage* non è stata molto semplice. La decisione principale da prendere era, se scegliere un progetto interno all'azienda, o se momentaneamente abbandonare l'azienda per fare l'esperienza di *stage* in un'altra.

All'inizio della mia carriera lavorativa, ho avuto l'opportunità di fare un'altro *stage* di sei mesi e questo per me è stato di fondamentale importanza per la mia crescita professionale.

Mi ha permesso di imparare tantissime cose, e di mettermi alla prova in un ambiente lavorativo diverso da quello universitario.

Lo *stage* per me rappresenta un'opportunità per imparare nuove tecnologie, e questa volta, come per la precedente, poteva essere l'occasione giusta. Dopo aver valutato i pro e i contro, riportati in figura 2.5, ho deciso di rimanere in azienda, perchè siamo riusciti a trovare un progetto interessante e stimolante, che mi ha permesso di imparare nuove tecnologie, e di mettermi alla prova.

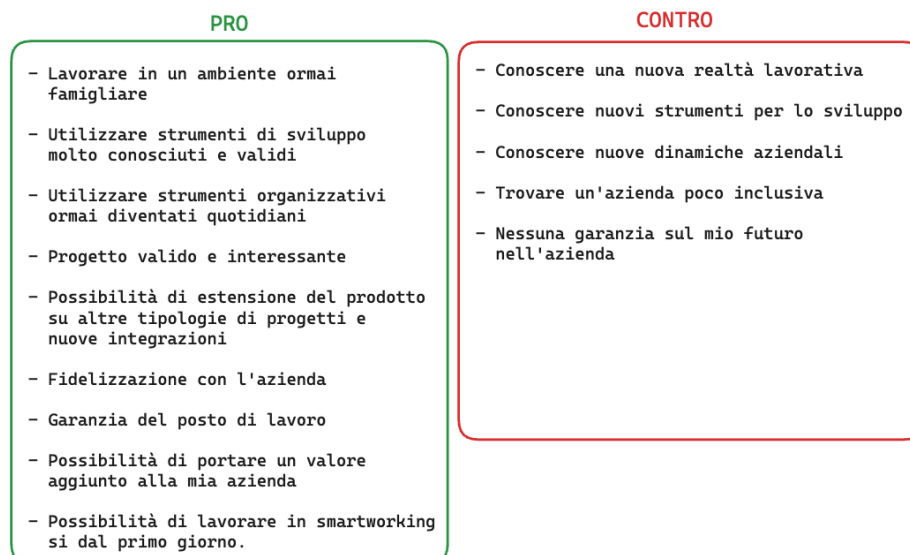


Figura 2.5: Pro e contro di fare lo *stage* nell'azienda per cui lavoro.

Questo progetto mi avrebbe permesso di affrontare problemi riguardanti lo sviluppo *software*, problemi di gestione e organizzazione del lavoro ma anche i problemi riguardanti i sistemi di *build* e di *continuous integration*, un argomento che mi ha sempre affascinato ma che non ho mai avuto modo di approfondire.

Per questo progetto mi ero imposto degli obiettivi personali:

1. Prendere dimestichezza con *Gradle* ed i sistemi di *build* in generale;
2. Imparare a scrivere un plugin per *Gradle*. Questo mi servirà in un futuro prossimo, per auto configurare alcuni progetto personali;
3. Conoscere i *database* a grafo, imparare ad utilizzarli e riuscire a comprendere quando è meglio utilizzarli rispetto ai *database* relazionali;

4. Provare le ultime versioni di *Angular* e delle librerie che solitamente uso per lo sviluppo delle *web-app*; In questo modo potrò valutare se è il caso di aggiornare i progetti che ho sviluppato in passato, e se è il caso di utilizzarle per i progetti futuri;

Successivamente farò un'analisi dei risultati ottenuti, e valuterò se gli obiettivi sono stati raggiunti.

2.6 Pianificazione e interazioni con il tutor

All'inizio dello *stage* ho avuto un incontro con il tutor aziendale, per discutere del piano di lavoro, e per definire la modalità di svolgimento e di interazione durante lo *stage*. Abbiamo deciso di fare un incontro settimanale, per discutere delle attività svolte durante la settimana, e per pianificare le attività della settimana successiva. Quando avevo dei dubbi o dei problemi, potevo contattarlo tramite *Google Chat* o tramite *email*.

Ogni mattina alle 9:00 avevamo un incontro con il *team* di sviluppo, per discutere delle attività svolte il giorno precedente, e per pianificare le attività della giornata, come previsto dalla metodologia *Agile* e illustrato in figura 2.6).

A questo incontro non partecipava il tutor aziendale. Capitava spesso comunque di avere degli incontri al di fuori di quelli pianificati, per discutere di problemi, principalmente analitici, e per prendere decisioni riguardanti l'implementazione.

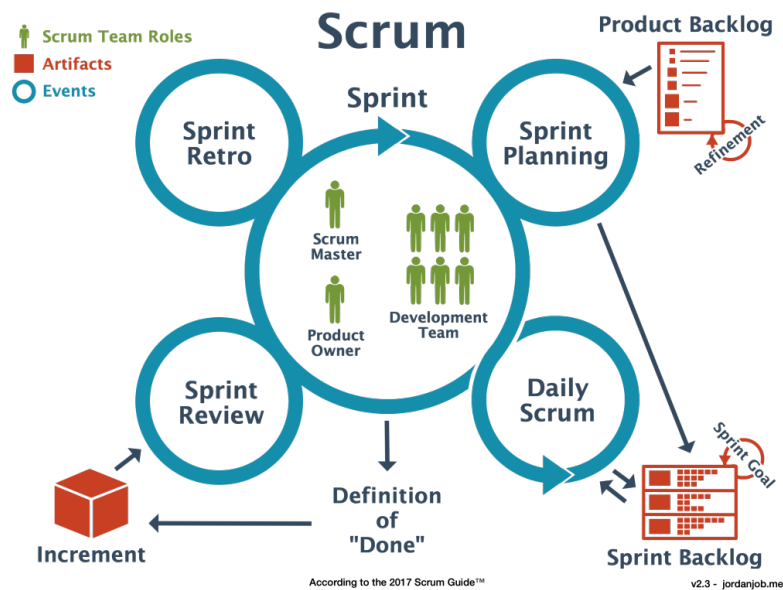


Figura 2.6: Metodologia Agile. Fonte: <https://jordanjob.me/blog/scrum-diagram>.

Capitolo 3

Svolgimento del progetto

In questo capitolo descrivo nel dettaglio le attività svolte durante lo *stage*, illustrando le scelte progettuali e le soluzioni adottate per la realizzazione del progetto. Per i concetti più importanti ho allegato immagini e *snippet* di codice, per rendere più chiara e comprensibile la spiegazione. Elenco i principali *framework*, tecnologie ed i *tool* utilizzati per la verifica e la validazione del progetto.

3.1 Analisi dei requisiti

Individuazione dei requisiti

3.2 Analisi dei Requisiti Durante lo *stage*

La seconda settimana dello *stage* l'ho stata dedicata all'analisi dei requisiti del progetto, un'attività cruciale che ho condotto insieme ai membri del *team*. Con uesto processo ho potuto identificare i requisiti funzionali, di vincolo e definire i casi d'uso del sistema.

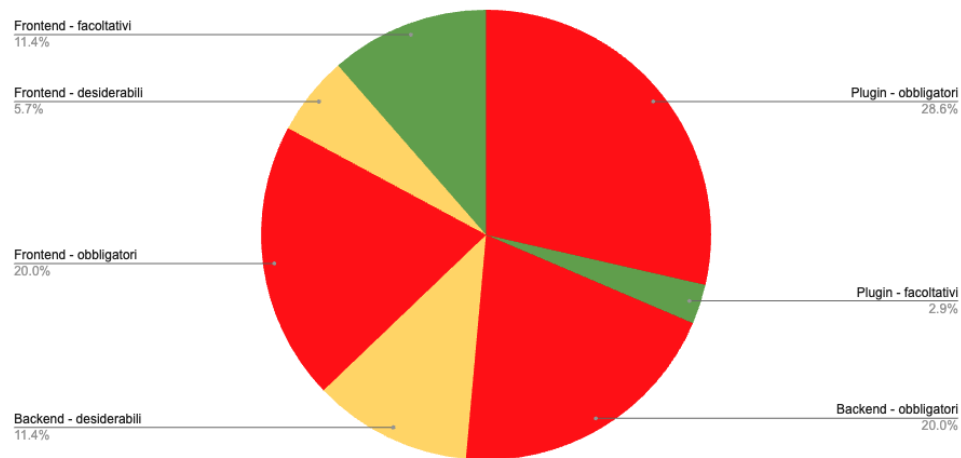
Adottando un approccio in linea con la metodologia agile, ho optato per una raccolta requisiti di tipo incrementale. Questo metodo ha permesso una maggiore flessibilità, consentendo aggiornamenti e modifiche dei requisiti in base alle mutevoli esigenze del *team* e dell'evoluzione del progetto.

Il primo passo è stato la creazione di un nuovo documento su *Confluence*. Ho iniziato inserendo le informazioni generali del progetto, quali il nome, il *team* coinvolto e una descrizione sintetica. Successivamente, ho definito e documentato le convenzioni adottate per la stesura del documento, come la nomenclatura specifica per i requisiti e i casi d'uso.

Nella fase successiva, ho proceduto con la stesura effettiva dei requisiti. Inizialmente, ho optato per una suddivisione basata sul tipo di requisito, differenziando tra funzionali e di vincolo attraverso due tabelle distinte. Tuttavia, nel corso dell'analisi, abbiamo introdotto un ulteriore livello di suddivisione. Questo ci ha permesso di classificare i requisiti anche in base al componente del sistema a cui si riferiscono, ovvero *backend*, *frontend* e *plugin* *Gradle*.

Conforme alle aspettative e alla natura agile del progetto, i requisiti sono stati soggetti a continui aggiornamenti e modifiche nel corso dello sviluppo. Un esempio significativo è stato l'inserimento dei requisiti facoltativi, che inizialmente non erano stati presi in considerazione, ma che si sono rivelati fondamentali per rispondere in modo più completo alle esigenze del progetto.

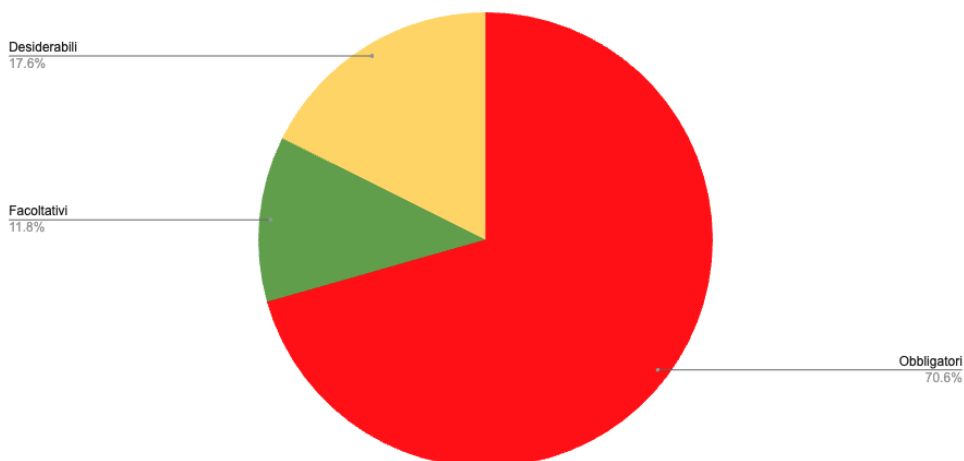
Plugin - Requisiti funzionali

**Figura 3.1:** Grafico a torta che mostra la suddivisione in percentuale dei vari requisiti.

Nell'immagine 3.1 è possibile vedere la suddivisione in percentuale dei requisiti, suddivisi come descritto in precedenza.

Nell'immagine invece, 3.2 è possibile vedere la suddivisione dei requisiti in base all'importanza.

Requisiti per importanza

**Figura 3.2:** Grafico a torta che mostra la suddivisione dei requisiti in base all'importanza.

3.2.1 Tracciamento dei requisiti

Ad ogni requisito scritto in *Confluence* ho associato un codice univoco, che mi ha permesso di tracciarlo. Il codice che ho utilizzato per identificarli è il seguente:

R[F/V]-[O/D]-[B/F/P]-[001]

Nello specifico:

- **RF**: requisito funzionale, **RV**: requisito di vincolo;
- **O**: obbligatorio, **D**: desiderabile, **F**: facoltativo;
- **P**: componente del sistema a cui si riferisce, **B**: *backend*, **F**: *frontend*, **P**: *plugin Gradle*;
- **001**: numero progressivo del requisito.

Per il dei requisiti, ho utilizzato un metodo efficace utilizzando *Confluence* e *Jira*. Nella tabella dei requisiti, che ho creato su *Confluence* e identificabile tramite la figura 3.3, ho inserito una nuova colonna. In questa colonna, ho inserito il riferimento alla *issue* di *Jira* corrispondente a ciascun requisito.

Questa integrazione tra *Confluence* e *Jira* ha permesso di collegare direttamente ogni requisito a una specifica *issue* di *Jira*, che ne dettaglia l'implementazione. Grazie a questo collegamento, ho ottenuto una visione più completa e dettagliata del progetto, migliorando notevolmente la precisione nel tracciamento dei requisiti.

Inoltre, ho sfruttato la funzionalità di visualizzazione automatica in modalità *read-only* su *Confluence*. Questo ha permesso di consultare direttamente nella tabella dei requisiti le informazioni essenziali relative alle *issues* di *Jira*, come lo stato di avanzamento e le descrizioni dettagliate, quando necessario.

Ogni *issue* l'ho collegata ad un'epica, che rappresenta un insieme di *issues* correlate tra loro. In tutto ho creato 4 epiche, una per ogni componente del progetto (*backend*, *frontend*, *plugin Gradle* e analisi).

Tramite *Jira* ho creato un *board* per il progetto, che mi ha permesso di visualizzare in modo chiaro e semplice lo stato di avanzamento del progetto, vedi la figura 3.4. All'interno di questa *board* è possibile visualizzare, per ogni *record* della tabella, il tipo di *issue*, il codice, la priorità, lo stato, la persona assegnata, la stima in giorni/ore e, per l'intero *sprint*, la stima totale in giorni/ore.

Jira offre una gamma di funzionalità avanzate, tra cui spicca la capacità di creare dei *plan*. Questa funzionalità si rivela particolarmente utile nella pianificazione delle *issues*, consentendo di organizzarle efficacemente in base alle loro date di inizio e fine, permettendoti di avere una visualizzazione grafica del progetto attraverso un grafico di *Gantt*.

Questo grafico rappresenta visivamente le *issues* in relazione alle loro date di inizio e fine, offrendo una panoramica chiara e immediata dello stato di avanzamento del progetto, mettendo in risalto anche le *milestone* prefissate nel piano di lavoro.

Inoltre, la funzionalità di *Jira* permette di modificare e riallocare le *issues* con facilità, adattandosi dinamicamente alle esigenze del progetto in corso.

Target release	
Epic	SYN-12817 - DONE
Document status	DRAFT
Document owner	@Gionata Legrottaglie
Designer	@Gionata Legrottaglie @Leo Molinaro
Developers	@Gionata Legrottaglie
QA	@Leo Molinaro

Requirements

ID	Issue Jira	Descrizione	Importanza
RF-PO-001	SYN-12821 - DONE	È possibile avviare il plugin tramite gradle	Obbligatorio
RF-PO-002	SYN-12824 - DONE	È possibile avviare il plugin da jenkins	Obbligatorio
RF-PO-003	SYN-12822 - DONE	È possibile analizzare un progetto gradle	Obbligatorio
RF-PO-004	SYN-12825 - DONE	È possibile analizzare un progetto npm	Obbligatorio
RF-PO-005	SYN-12829 - DONE	I dati sono salvati in un database a grafo	Obbligatorio
RF-PO-006	SYN-12831 - DONE	Ogni applicativo deve impostare il proprio identificativo nella configurazione del plugin	Obbligatorio
RF-PO-007	SYN-12832 - DONE	I dati inviati dal plugin sono salvati riferendosi all'identificativo dell'applicativo	Obbligatorio
RF-PO-008	SYN-12833 - DONE	È possibile configurare il plugin per impostarne il server di destinazione	Obbligatorio
RF-PO-009	SYN-12834 - DONE	È possibile configurare il plugin per impostare i progetti npm da analizzare	Obbligatorio
RF-PO-010	SYN-12835 - DONE	Il plugin deve essere compatibile con la versione 7 di gradle	Obbligatorio
RF-PF-011	SYN-12836 - DONE	Il plugin deve essere compatibile con la versione 8 di gradle	Facoltativo

Figura 3.3: Tabella che mostra il tracciamento dei requisiti su *Confluence*, con il collegamento a *Jira*.

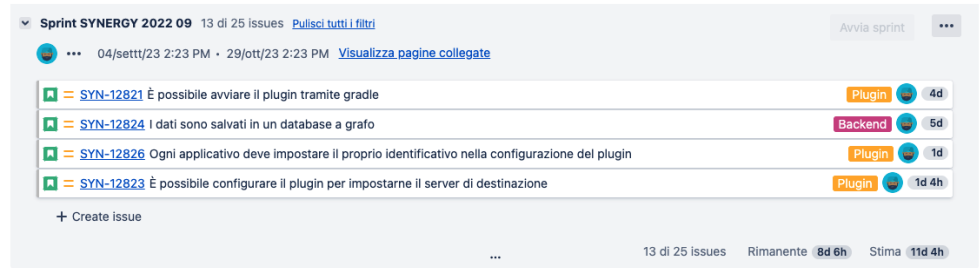


Figura 3.4: *Board* di *Jira* che mostra lo stato di avanzamento del progetto.

3.3 Progettazione architetturale

Progettazione delle strutture dati

La prima fase della progettazione architetturale è stata la progettazione delle strutture dati. Era molto importante progettare le strutture dati in modo da poterle utilizzare in modo efficiente e veloce, in quanto il progetto si basa su di esse.

L'obiettivo era quello di creare una struttura dati che permettesse di rappresentare un grafo di dipendenze tra i vari pacchetti di un progetto.

Ogni nodo del grafo rappresenta un pacchetto, ed aveva bisogno di avere le seguenti informazioni:

- **id**: identificativo univoco del nodo;
- **group**: nome del gruppo del pacchetto;

- **name**: nome del pacchetto;
- **version**: versione del pacchetto;
- **tipo**: tipo del pacchetto, può essere *Java* o *JavaScript*;

Ogni arco del grafo, come mostrato in figura 3.5 rappresenta una dipendenza tra due pacchetti, ed ha bisogno di avere le seguenti informazioni:

- **id**: identificativo univoco dell'arco;
- **from**: identificativo del nodo di partenza;
- **to**: identificativo del nodo di arrivo;
- **require**: numero di versione richiesto dal pacchetto di partenza;
- **variants**: rappresenta la lista di utilizzi del pacchetto, es: *compileOnly*, *implementation*, *testImplementation*;

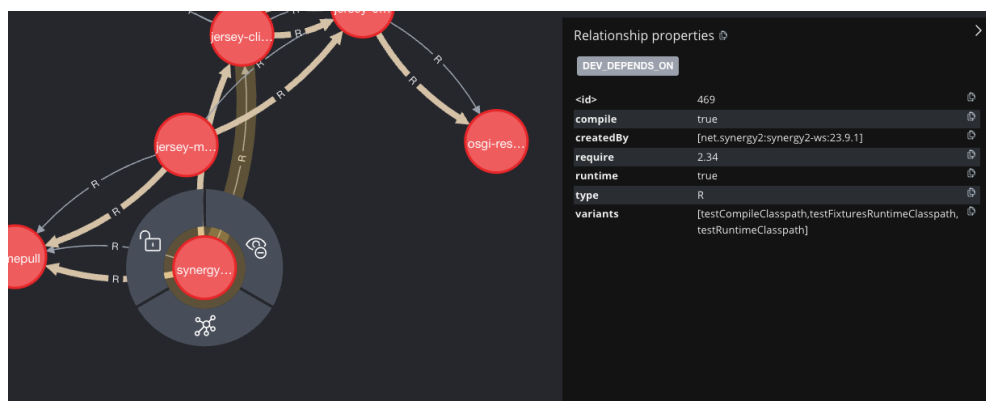


Figura 3.5: Esempio di grafo di dipendenze tra pacchetti.

Prima di arrivare alla struttura finale, abbiamo fatto diverse prove, cercando di trovare la struttura più adatta alle nostre esigenze.

La versione finale, come mostrato in figura 3.6 ha saputo soddisfare tutte le nostre esigenze.

Da ogni nodo possono partire tre tipi di archi, ognuno con una funzione diversa; Il primo tipo di arco è quello che rappresenta una dipendenza diretta tra due pacchetti, quindi ad esempio se il pacchetto **A 1.0** dipende dal pacchetto **B 1.0**, allora dal nodo A parte un arco, nella figura è rappresentato con una linea continua, che punta al nodo B.

Il secondo tipo di arco è quello che rappresenta una dipendenza transitiva tra due pacchetti, quindi ad esempio se il pacchetto **A 1.0** dipende dal pacchetto **C 1.0** e il pacchetto **C 1.0** dipende dal pacchetto **D 3.0**, allora dal nodo A parte un arco, nella figura è rappresentato con una linea tratteggiata, che punta al nodo **D 3.0**.

Il terzo ed ultimo tipo di arco è quello che rappresenta una dipendenza transitiva tra due pacchetti, ma con una versione diversa, quindi ad esempio se il pacchetto **A 1.0** dipende dal pacchetto **B 1.0** e il pacchetto **B 1.0** dipende dal pacchetto **D 2.0**, ma il pacchetto **A 1.0** dipende anche, in modo diretto o transitivo, dal pacchetto **D**

3.0, allora dal nodo **B 1.0** parte un arco, nella figura è rappresentato con una linea tratteggiata, a tratto più piccolo, che punta al nodo **D 3.0**.

Questo ultimo tipo di arco l'abbiamo introdotto per poter gestire le dipendenze transitive con versioni diverse, dato che al rilascio di un pacchetto finale da installare, la versione che verrà installata sarà quella più recente, quindi in questo caso la versione **D 3.0**.

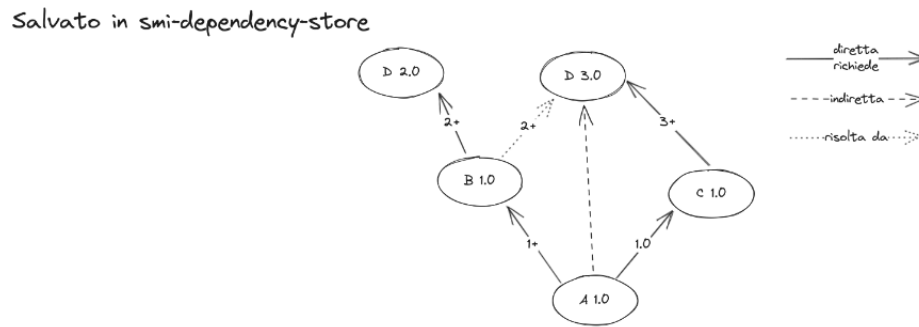


Figura 3.6: Struttura finale del grafo di dipendenze.

Progettazione plugin di analisi delle dipendenze

Il progetto *Java* per la creazione del *plugin Gradle*, *smi-dependency-analyzer*, è stato sviluppato utilizzando il *framework Gradle*.

Gradle mette a disposizione delle *API* per la creazione di *plugin* personalizzati, che permettono di estendere le funzionalità del *build tool*.

Il progetto contiene tre *package* principali:

- **model**: contiene le classi che rappresentano le strutture dati utilizzate dal *plugin*;
- **logic**: contiene le classi che rappresentano la logica del *plugin*, dove vengono eseguite le operazioni di analisi, trasformazione e trasferimento dei dati;
- **task**: contiene le classi che rappresentano i *task* del *plugin*, ovvero le operazioni che possono essere eseguite dal *plugin* e la classe di registrazione del *plugin*.

Ho definito anche una classe che rappresenta l'oggetto di configurazione del *plugin*, che contiene le informazioni necessarie per l'esecuzione del *plugin* e per il salvataggio dei dati, un esempio lo si può visualizzare nello *snippet 3.1*

```

1 smi_dependency_analyzer {
2     username = "nome_utente"
3     password = "private_key"
4     url = "http://localhost:8080/smi-dependency-store"
5     npmProject {
6         packageJson = "/projects/esempio1/client/package.json"
7         packageLockJson = "/projects/esempio1/client/package-lock.json"
8     }
  
```

9 }

Snippet 3.1: Esempio di configurazione del *plugin Gradle*, utilizzando il linguaggio *Groovy*.

L'idea di base del *plugin* è quella di creare un *task* che avvii il task standard messo a disposizione da *Gradle* per la creazione del *dependency tree*, ovvero *dependencyInsight*, e con l'output di questo task, creare un grafo di dipendenze, che verrà inviato al *server* per essere salvato.

Per i progetti *Npm* invece, il *plugin* legge il file *package-lock.json*. Questo file viene generato automaticamente da *Npm* e contiene tutte le informazioni necessarie per la creazione del grafo di dipendenze.

Un prerequisito per il funzionamento del *plugin* è che sia stato eseguito il comando ***npm install***, in modo da avere tutte le dipendenze installate nella cartella *node_modules* e quindi avere il file *package-lock.json*.

Una volta raccolte tutte le informazioni necessario, effettuo una chiamata *REST* di tipo *PUT* al *server*, per salvare il grafo di dipendenze.

Progettazione *backend*

Il progetto del *backend*, denominato **smi-dependency-store**, l'ho suddiviso in quattro *package* principali:

- ***model***: contiene le classi che rappresentano le strutture dati utilizzate dal *server* e le classi di configurazione;
- ***dao***: contiene le classi utilizzate per la comunicazione con il *database*, ed in questo caso conteneva la classe ***PackageNodeDao*** che gestiva l'inserimento/aggiornamento/cancellazione dell'entità nodo ***PackageNode*** e la classe ***DependencyLinkDao*** che gestiva l'inserimento/aggiornamento/cancellazione dell'entità arco ***DependencyLink***;
- ***logic***: contiene le classi che rappresentano la logica del *server*, dove vengono eseguite le operazioni di analisi, trasformazione e chiamate alle classi *dao*;
- ***rest***: contiene le classi che rappresentano i *controller* del *server*, ovvero tutti i servizi *REST* messi a disposizione dal *server*.

Per la configurazione del *server*, ho previsto quattro diversi file di configurazione, uno per ogni funzionalità del *server*:

- **neo4j.yml**: contiene le informazioni per la connessione al *database Neo4j*, come l'indirizzo, l'username e la password. Vedi snippet 3.2;

```
1 url: "bolt://localhost:7689"
2 username: "neo4j"
3 password: "neo4jlocal"
```

Snippet 3.2: Esempio di configurazione del *neo4j.yml* captionpos

- **token.yml** contiene le informazioni per la generazione del *token*, come la chiave segreta, il *clientId*, il *clientSecret*, il nome del *server* e la durata del *token*. Vedi snippet 3.3. Questo vuol dire che se non configurato l'*LDAP*, ci sarà un solo

utente che potrà accedere al *server*, utilizzando come *username* e *password* quelle configurate come *clientId* e *clientSecret*.

Il token *JWT* generato avrà una durata di 30 minuti e sarà firmato con la chiave segreta *SECRET_KEY*.

```
1  hmacSha512Key: "SECRET_KEY"
2  clientId: "admin"
3  clientSecret: "admin"
4  issuer: "dependency_store"
5  validity: 1800000
```

Snippet 3.3: Esempio di configurazione del *token.yml*.

- **logger.yml:** contiene le informazioni per la configurazione del *logger*, come il livello di log, il formato del *log* e il file di output. Vedi snippet 3.4. Questa configurazione viene passata ad una libreria sviluppata da Sanmarco informatica S.p.a., che è un *wrapper* della libreria *slf4j*, nota libreria per la gestione dei *log* in *Java*. Il campo *params* contiene le informazioni per la configurazione del livello di *log* per ogni *package* del *server*. Questo mi permette di avere un livello di *log* diverso per ogni *package* del *server*, molto utile per il *debug*.

```
1  defaultLogLevel: "info"
2  showDateTime: true
3  dateTimeFormat: "yyyy-MM-dd HH:mm:ss"
4  showThreadName: false
5  showShortLogName: true
6  logFile: "../logs/out.log"
7  params:
8    com.smi: "debug"
```

Snippet 3.4: Esempio di configurazione del *logger.yml*.

- **ldap.yml** contiene le informazioni per la configurazione dell'*LDAP*, come l'indirizzo del *server*, il dominio ed un *flag* che indica se utilizzare una connessione sicura o meno. Vedi snippet 3.5.

```
1  url: "ldap://10.10.99.1"
2  domain: "DOMINIO_AZIENDA"
3  ssl: false
```

Snippet 3.5: Esempio di configurazione dell'*ldap.yml*.

Nel corso dello sviluppo, ho adottato il *design pattern Singleton* per l'implementazione delle classi *Java* dedicate alla logica, *dao* e configurazione. Questo approccio assicura che esista una sola istanza di queste classi all'interno del *server*, utilizzata condivisamente da tutte le altre classi.

Per realizzare il *Singleton* in *Java*, ho sfruttato una libreria sviluppata da Sanmarco informatica S.p.a., che facilita l'implementazione di questo pattern in modo efficiente e rapido. L'*snippet* 3.6 mostra un esempio di questa implementazione per la classe *PackageNodeDao*. Abbiamo optato per l'uso di una *inner class* statica, dotata di un

campo statico *SingletonHolder*. Quest'ultimo contiene un metodo *get* che accetta una *lambda expression*, restituendo un oggetto di tipo *PackageNodeDao*. Questa strategia garantisce la creazione dell'istanza solo all'occorrenza, evitando la creazione anticipata al momento del caricamento della classe e assicurando l'unicità dell'istanza anche in contesti di *multithreading*.

La classe *SingletonHolder* non solo gestisce la creazione dell'istanza, ma offre anche la possibilità di personalizzare i metodi di *PackageNodeDao*. Se si desidera modificare i metodi di questa classe, è possibile definire una sottoclasse di *PackageNodeDao*, nominata *PackageNodeDaoPers*. In questo caso, grazie all'uso della *reflection* di Java, il *SingletonHolder* restituirà automaticamente un'istanza di *PackageNodeDaoPers*, permettendo una personalizzazione avanzata.

```

1 public class PackageNodeDao {
2     protected PackageNodeDao () { }
3     public static PackageNodeDao get() {
4         return Singleton.INSTANCE.get(PackageNodeDao::new);
5     }
6     private static class Singleton {
7         private static final SingletonHolder<PackageNodeDao> INSTANCE
8             = new SingletonHolder<> (PackageNodeDao.class);
9     }

```

Snippet 3.6: Esempio di implementazione del *design pattern Singleton* in Java.

Progettazione *frontend*

Per la progettazione del *frontend* siamo dal *mockup* grafico realizzato in fase di analisi. Questo ci ha permesso di avere una visione chiara e dettagliata dell'interfaccia grafica, semplificando notevolmente la fase di sviluppo.

Il *mockup* è stato realizzato utilizzando il tool gratuito online *excalidraw*, che permette di creare grafiche semplici come quella mostrata in figura 3.7.

Da qui ho iniziato a sviluppare il *frontend*, utilizzando il framework *Angular*. Per prima cosa creato le due cartelle principali del progetto, la prima **,commons**, contiene i componenti, i servizi e i modelli che possono essere utilizzati in più componenti; la seconda, **features**, contiene a sua volta una cartella per ogni funzionalità del *frontend*. Ho creato quindi le cartelle *home*, *login*, *query* e *find-by-project*. Ogni cartella al suo interno contiene i componenti, i servizi e i modelli relativi alla funzionalità. Nella figura 3.8 è possibile vedere la struttura del progetto.

Per la scrittura del codice ho seguito le linee guida di *Angular*, rilasciate nel loro sito ufficiale *Angular Style Guide*. URL: <https://angular.io/guide/styleguide>.

Come mostrato in figura 3.9, l'interfaccia grafica è formata da una barra di navigazione, che contiene il nome del progetto, il pulsante per il *logout* ed i pulsanti per cambiare la lingua dell'interfaccia grafica.

Sotto la barra di navigazione è presente una *sidebar*, che contiene i pulsanti per accedere alle varie funzionalità del *frontend* ed il contenitore principale che mostra il contenuto della funzionalità selezionata.

Le voci di menu presenti nella *sidebar*, oltre alla funzionalità *home*, sono tre:



Figura 3.7: Mockup grafico dell'interfaccia grafica.

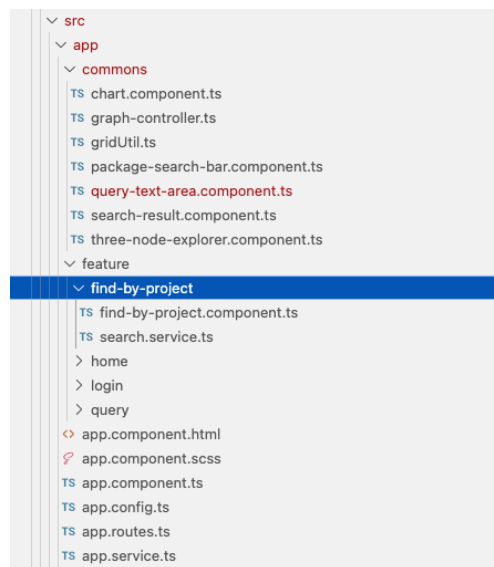


Figura 3.8: Struttura del progetto *frontend*.

- **Ricerca per progetto:** Come mostrato in figura 3.9, in questo modulo, tramite la barra di ricerca è possibile inserire il nome completo, formato da *group*, *name* e *version*, di un pacchetto e visualizzare le informazioni relative a quel pacchetto. Tuttavia per rendere più semplice la ricerca, ho aggiunto una funzionalità di suggerimento, che mostra la lista di pacchetti conosciuti dal *server*, che iniziano con la stringa inserita nella barra di ricerca e raggruppati per *group*, come rap-



Figura 3.9: Modulo di ricerca dipendenze per progetto.

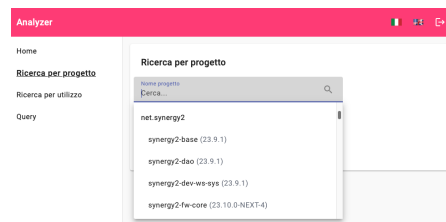


Figura 3.10: Casella di ricerca con lista di suggerimenti.

presentato in figura 3.10.

Sotto alla casella di testo sono presenti tre *checkbox*, che permette di scegliere se includere i pacchetti targati come *dev*, se includere i pacchetti utilizzati solo durante la compilazione dei progetti o se includere tutti i pacchetti utilizzati a runtime da un progetto.

Una volta avviata la ricerca, viene mostrato il risultato sotto di esso, come mostrato in figura 3.11.

Qui troviamo tre possibili visualizzazioni del risultato: a grafo, ad albero e a lista.

Nella visualizzazione tabellare abbiamo una lista piatta di tutti i pacchetti, con le informazioni principali, trascurando i collegamenti tra i pacchetti.

Questa visualizzazione può tornare utile quando si vuole avere una visione d'insieme di tutti i pacchetti utilizzati da un progetto.

Nella visualizzazione ad albero invece, abbiamo una visualizzazione gerarchica dei pacchetti, dove ogni nodo rappresenta un pacchetto ed espandendo un nodo è possibile vedere i pacchetti da cui dipende.

In questo modo riusciamo a capire a come mai un pacchetto è stato incluso nel progetto, e quali altri pacchetti sono stati inclusi a causa di esso.

Nella visualizzazione a grafo invece, abbiamo una visualizzazione grafica dei pacchetti, dove ogni nodo rappresenta un pacchetto e gli archi rappresentano le dipendenze tra i pacchetti.

Inoltre, in visualizzazione è possibile fare un controllo sulla presenza di ag-

165 risultati

net_synergy2_synergy2-ws:23.9.1

Grafo

Griglia

Albero

```

Query(text="MATCH (p)-[:DEPENDS_ON]->(n) WHERE p.id = $id AND (L.compile = false OR L.runtime = withRuntime) AND (L.runtime = true OR L.compile = withCompile) RETURN type (l), labels (n), l.type, l.required, l.variants, l.createdBy, L.compile, l.runtime, n.id, n.group, n.name, n.version ORDER BY n.id", parameters={id: "net_synergy2_synergy2-ws:23.9.1", withCompile: FALSE, withRuntime: TRUE})

```

Id ↑

Gruppo

Nome

com.fasternxml.jackson.core.jackson-annotations:2.15.1

com.fasternxml.jackson.core:2.15.1

com.fasternxml.jackson.core.jackson-databind:2.15.1

com.fasternxml.jackson.dataformat.jackson-dataformat-yaml:2.15.1

com.fasternxml.jackson.datatype.jackson-datatype-jr310:2.15.1

com.fasternxml.jackson.jaxrs.jackson-jaxrs-base:2.15.1

com.fasternxml.jackson.jaxrs.jackson-jaxrs-jsp-provider:2.15.1

com.fasternxml.jackson.module.jackson-module-jaa-annotations:2.15.1

com.fasternxml.jackson.module-jackson-bom:2.15.1

github.oshi.oshi-core:5.7.2

google.api-client:google-api-client:2.2.0

google.apis.oauth2api-services-calendar-v3-rv20230317.0.0

com.fasternxml.jackson.core

com.fasternxml.jackson.core

com.fasternxml.jackson.core

com.fasternxml.jackson.dataformat

com.fasternxml.jackson.datatype

com.fasternxml.jackson.jaxrs

com.fasternxml.jackson.jaxrs

com.fasternxml.jackson.module

com.fasternxml.jackson

github.oshi

google.api-client

google.apis

jackson annotations

jackson core

jackson databind

jackson dataformat-yaml

jackson datatype-jr310

jackson-jaxrs-base

jackson-jaxrs-jsp-provider

jackson-module-jaa-annot

jackson-bom

oshi-core

google-api-client

google-api-services-calendar

Figura 3.11: Risultato tabellare.

giornamenti e vulnerabilità, ad esempio, come mostrato in figura 3.13, vengono

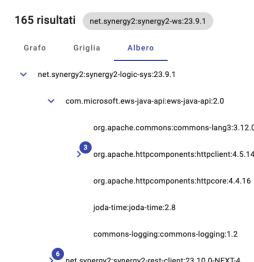


Figura 3.12: Risultato ad albero.

mostrati tutti i pacchetti che hanno una versione più recente di quella utilizzata dal progetto.

```
0: "[com.fasterxml.jackson.core:jackson-annotations:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
1: "[com.fasterxml.jackson.core:jackson-annotations:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
2: "[com.fasterxml.jackson.core:jackson-core:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
3: "[com.fasterxml.jackson.core:jackson-core:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
4: "[com.fasterxml.jackson.core:jackson-databind:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
5: "[com.fasterxml.jackson.core:jackson-databind:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
6: "[com.fasterxml.jackson.dataformat:jackson-dataformat-yaml:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
7: "[com.fasterxml.jackson.dataformat:jackson-dataformat-yaml:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
8: "[com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
9: "[com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
10: "[com.fasterxml.jackson.jaxrs:jackson-jaxrs-base:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
11: "[com.fasterxml.jackson.jaxrs:jackson-jaxrs-base:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
12: "[com.fasterxml.jackson.jaxrs:jackson-jaxrs-json-provider:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
13: "[com.fasterxml.jackson.jaxrs:jackson-jaxrs-json-provider:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
14: "[com.fasterxml.jackson.module:jackson-module-jaxb-annotations:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
15: "[com.fasterxml.jackson.module:jackson-module-jaxb-annotations:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
16: "[com.fasterxml.jackson.jakarta.rs:jackson-jakarta-rs-base:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
17: "[com.fasterxml.jackson.jakarta.rs:jackson-jakarta-rs-base:2.15.1] Minor version update available: 2.15.1 -> 2.16.0"
18: "[com.github.oshi:oshi-core:5.7.2] Major version update available: 5.7.2 -> 6.4.7"
19: "[com.github.oshi:oshi-core:5.7.2] Major version update available: 5.7.2 -> 6.4.7"
20: "Failed to check updates for com.google.apis:google-api-services-calendar - For input string: \"v3-rev20230317-2\""
21: "Failed to check updates for com.google.apis:google-api-services-calendar - For input string: \"v3-rev20230317-2\""
22: "[com.google.code.gson:gson:2.10] Patch version update available: 2.10 -> 2.10.1"
23: "[com.google.code.gson:gson:2.10] Patch version update available: 2.10 -> 2.10.1"
24: "[com.google.errorprone:error_prone_annotations:2.16] Minor version update available: 2.16 -> 2.23.0"
25: "[com.google.errorprone:error_prone_annotations:2.16] Minor version update available: 2.16 -> 2.23.0"
26: "[com.google.guava:failureaccess:1.0.1] Patch version update available: 1.0.1 -> 1.0.2"
27: "[com.google.guava:failureaccess:1.0.1] Patch version update available: 1.0.1 -> 1.0.2"
28: "[com.google.guava:guava:31.1-jre] Major version update available: 31.1-jre -> 32.1.3-jre"
29: "[com.google.guava:guava:31.1-jre] Major version update available: 31.1-jre -> 32.1.3-jre"
30: "[com.google.http-client:google-http-client-apache-v2:1.42.3] Minor version update available: 1.42.3 -> 1.43.3"
31: "[com.google.http-client:google-http-client-apache-v2:1.42.3] Minor version update available: 1.42.3 -> 1.43.3"
```

Figura 3.13: Visualizzazione di tutti gli aggiornamenti disponibili.

- Ricerca per utilizzo:** La ricerca avviene in modo simile a quella per progetto, ma in questo caso, il risultato mostra solo una griglia con i pacchetti che utilizzano il pacchetto cercato. Nell'esempio in figura 3.14, ho cercato il pacchetto *net.synergy2:synergy2-base:23.9.1*, e il risultato mostra tutti i pacchetti che lo utilizzano. Oltre alla griglia, è presente anche un *box* che mostra la *query*, con la sintassi *Cypher*, utilizzata per la ricerca.

12 risultati net.synergy2:synergy2-base:23.9.1

```
Query(text='MATCH (p)-[l:DEPENDS_ON]->(n) WHERE n.id = $id AND (l.compile = false OR l.runtime = $withRuntime) AND (l.runtime = true OR l.compile = $withCompile) RETURN type (l), labels (n), l.type, l.require, l.variants, l.createdBy, l.compile, l.runtime, p.id, p.group, p.name, p.version ORDER BY p.id', parameters={id: "net.synergy2:synergy2-base:23.9.1", withCompile: FALSE, withRuntime: TRUE})
```

Id ↑	Gruppo	Nome	Versione	Tipo	Dev
net.synergy2:synergy2-dao:23.9.1	net.synergy2	synergy2-dao	23.9.1	java	false
net.synergy2:synergy2-dev-ws-sys:23.9.1	net.synergy2	synergy2-dev-ws-sys	23.9.1	java	false
net.synergy2:synergy2-logic-dvz:23.9.1	net.synergy2	synergy2-logic-dvz	23.9.1	java	false
net.synergy2:synergy2-logic-sys:23.9.1	net.synergy2	synergy2-logic-sys	23.9.1	java	false
net.synergy2:synergy2-model:23.9.1	net.synergy2	synergy2-model	23.9.1	java	false
net.synergy2:synergy2-schedule-sys:23.9.1	net.synergy2	synergy2-schedule-sys	23.9.1	java	false
net.synergy2:synergy2-virtual-sys:23.9.1	net.synergy2	synergy2-virtual-sys	23.9.1	java	false
net.synergy2:synergy2-ws-base:23.9.1	net.synergy2	synergy2-ws-base	23.9.1	java	false
net.synergy2:synergy2-ws-dvz:23.9.1	net.synergy2	synergy2-ws-dvz	23.9.1	java	false
net.synergy2:synergy2-ws-sys:23.9.1	net.synergy2	synergy2-ws-sys	23.9.1	java	false
net.synergy2:synergy2-ws:23.9.1	net.synergy2	synergy2-ws	23.9.1	java	false
net.synergy2:webpush-java:23.9.1	net.synergy2	webpush-java	23.9.1	java	false

Figura 3.14: Esempio di ricerca per utilizzo.

- **Query:** in quest'ultima funzionalità, come mostrato in figura 3.14, è possibile inserire una *query Cypher* libera e visualizzare il risultato in formato *JSON*. Questo può tornare utile quando si vuole effettuare una ricerca più complessa, che non è possibile effettuare con le altre funzionalità.



Figura 3.15: Modalità *query* libera.

Codifica

Librerie e *framework* utilizzati

Jersey

Per la creazione servizi *REST*, ho utilizzato *Jersey*, che tramite annotazioni su metodi e classi, mi ha permesso di creare facilmente i vari servizi.

Lo *snippet* 3.7 mostra un esempio di *servizio REST* che restituisce tutte le dipendenze di un pacchetto.

```

1 @GET ()
2 @Path ("dependency/byPackage/{id}")
3 @Produces (MediaType.APPLICATION_JSON) @Consumes
  (MediaType.APPLICATION_JSON)
4 public Response byPackage (@PathParam ("id") String id,
5   @QueryParam ("withDev") boolean withDev,
6   @QueryParam ("withCompile") boolean withCompile,
7   @QueryParam ("withRuntime") boolean withRuntime
8 ) {

```

```
9      return Response.ok (DependencyLogic.get  
        ().getDependenciesByPackage (id, withDev, withCompile,  
        withRuntime)).build ();  
10    }
```

Snippet 3.7: Esempio di *servizio REST* utilizzando *Jersey*.

Neo4j

Per la gestione del *database Neo4j*, ho utilizzato la libreria *Neo4j Java Driver*, che mette a disposizione una serie di classi e metodi per l'apertura e la gestione delle connessioni al *database* e per l'esecuzione di *query Cypher*.

smi-commons

Per la gestione delle operazioni comuni, come la gestione del *logger*, la gestione delle eccezioni, la lettura dei file di configurazione e la gestione del *token*, ho utilizzato una libreria sviluppata da Sanmarco informatica S.p.a., denominata *smi-commons*.

Nx

Per la creazione e gestione del progetto *Angular*, ho utilizzato il *tool Nx*, che permette di creare progetti *Angular* senza dover pensare alla struttura del progetto.

Nx mette a disposizione una serie di comandi per la creazione di progetti, moduli, componenti, servizi e librerie, e permette di eseguire i *textitests* e il *build* del progetto. *Nx* viene utilizzato principalmente per la gestione di progetti *monorepo*, ovvero progetti che contengono più applicazioni, in modo da poter condividere le librerie tra le varie applicazioni.

Nel mio caso però avevo solo un'applicazione, ma ho deciso di utilizzarlo comunque per sfruttare le semplificazioni sulla gestione dei *test* e del *build* del progetto.

Interrogazioni su *database* a grafo

Per la gestione delle interrogazioni su *database* a grafo, ho utilizzato il linguaggio *Cypher*, che è un linguaggio dichiarativo per la manipolazione di grafi.

Questa è stata la parte più complessa del progetto, in quanto non avevo mai utilizzato un *database* a grafo, e quindi non conoscevo il linguaggio *Cypher*.

Per imparare il linguaggio, ho seguito un corso sulla piattaforma *Udemy*, messa a disposizione da Sanmarco informatica S.p.a., che mi ha permesso di apprendere le basi del linguaggio.

Ricerca di aggiornamenti e vulnerabilità

3.4 Verifica e validazione

Analisi statica

Analisi dinamica

Test di unità

Test di integrazione

Test *End-to-End*

3.5 I risultati

Appendice A

Appendice A

Citazione

Autore della citazione

Acronimi e abbreviazioni

- B2B** [Business to Business](#). 2, 35
- BCorp** [Benefit Corporation](#). 2, 3, 34
- BPM** [Business Process Management](#). 2, 35
- CPQ** [Configure Price Quote](#). 35
- CRM** [Customer Relationship Management](#). 2, 35
- DSL** [Domain Specific Language](#). 35
- ECM** [Enterprise Content Management](#). 2, 35
- ERP** [Enterprise Resource Planning](#). 1, 36
- IDE** [Integrated Development Environment](#). 36
- IoT** [Internet of Things](#). 1, 36
- IT** [Information Technology](#). 4, 34
- JVM** [Framework](#). 6
- PO** [Product Owner](#). 4, 36
- SDGs** [Sustainable Development Goals](#). 2, 3, 34

Glossario

Analisti L'analista è una figura professionale che si occupa di analizzare i requisiti del cliente e di redigere la documentazione. Molte volte gli analisti sono anche sviluppatori e tester.. [4](#)

B2B Business-to-business (B2B) è un modello di business che si riferisce alle transazioni commerciali tra due aziende, come quelle tra un produttore e un grossista o tra un grossista e un dettagliante.. [34](#)

BPM Business process management (BPM) è un approccio alla gestione delle operazioni aziendali che si concentra su allineamento tutti i processi con i desideri e le esigenze dei clienti.. [34](#)

Continuous integration in ingegneria del software, l'integrazione continua (*continuous integration*) è una pratica che si applica in contesti in cui lo sviluppo del software avviene attraverso un sistema di versionamento. Consiste nell'allineamento frequente dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso, al fine di rilevare tempestivamente eventuali errori di integrazione. [5](#)

CPQ Configure Price Quote (CPQ) è un software che consente alle aziende di automatizzare alcuni dei processi più complessi e propensi agli errori nella vendita di prodotti e servizi.. [2](#), [34](#)

CRM Customer relationship management (CRM) è un approccio per gestire l'interazione di un'azienda con i clienti attuali e potenziali. Utilizza l'analisi dei dati sui clienti per migliorare le relazioni con i clienti, concentrarsi sulla customer retention e guidare le vendite.. [34](#)

Cybersecurity La cybersecurity è la pratica di proteggere i sistemi, le reti e i programmi da attacchi digitali. Questi attacchi sono generalmente mirati a accedere, modificare o distruggere informazioni sensibili; estorcere denaro ai utenti; o interrompere normali operazioni aziendali.. [2](#)

DSL Un Domain Specific Language (DSL) è un linguaggio di programmazione o un linguaggio di specifica dedicato a un particolare dominio di problemi, tecnica e/o settore di applicazione.. [13](#), [34](#)

ECM Enterprise content management (ECM) è un insieme di strumenti e strategie che consentono a un'organizzazione di acquisire, organizzare, archiviare e distribuire informazioni critiche per l'organizzazione.. [34](#)

ERP Un sistema di pianificazione delle risorse aziendali (ERP) è un sistema di gestione che consente a un'organizzazione di utilizzare un sistema di applicazioni integrate per gestire l'attività aziendale e automatizzare molte funzioni back office relative alla tecnologia, ai servizi e ai processi umani.. [34](#)

Framework Un *framework* è una struttura concettuale e tecnologica predefinita che fornisce un modello standard su cui gli sviluppatori possono costruire applicazioni. Include librerie di codice, strumenti e linee guida che facilitano lo sviluppo, consentendo agli sviluppatori di concentrarsi sulla logica specifica dell'applicazione piuttosto che su dettagli di basso livello. Un framework può anche promuovere le buone pratiche di programmazione e ridurre la probabilità di errori.. [5](#), [34](#)

Gradle Gradle è un sistema di automazione open source che gestisce le dipendenze e permette di automatizzare il processo di compilazione, testing, pubblicazione e deployment di un software.. [10](#)

IDE L'acronimo IDE sta per "Integrated Development Environment" che in italiano si traduce come "Ambiente di Sviluppo Integrato". Un IDE è un software che fornisce strumenti e servizi integrati per facilitare ai programmatori lo sviluppo di software. Include spesso un editor di codice, strumenti per il debugging, e funzionalità per la gestione di progetti, tra gli altri.. [6](#), [34](#)

IoT L'Internet of Things (IoT) è un sistema di dispositivi interconnessi digitalmente, macchine, oggetti, animali o persone che sono forniti di identificatori univoci e la capacità di trasferire dati su una rete senza richiedere interazioni uomo-uomo o uomo-computer.. [34](#)

Npm Npm è un gestore di pacchetti per il linguaggio di programmazione JavaScript. È il gestore di pacchetti predefinito per l'ambiente di runtime JavaScript Node.js.. [10](#)

PO Il Product Owner (PO) è una figura professionale che si occupa di gestire il progetto e di interfacciarsi con il cliente.. [34](#)

Project Management Il project management è l'insieme di attività di pianificazione, organizzazione, gestione e controllo di un progetto.. [2](#)

Prototipo Un prototipo è un esemplare o un modello di un prodotto o di un sistema che viene realizzato prima del prodotto finale.. [10](#)

Reflection La reflection è una funzionalità di un linguaggio di programmazione che consente di ispezionare, analizzare e modificare il proprio codice sorgente o oggetti in esecuzione.. [26](#), [36](#)

Repository Un repository è un archivio di dati digitali.. [5](#)

Scrum Master Lo Scrum Master è una figura professionale che si occupa di gestire il team di sviluppo e di facilitare il processo di sviluppo.. [4-6](#), [36](#)

Supply Chain La supply chain è la rete globale di tutte le organizzazioni coinvolte nella creazione e nella distribuzione di un prodotto o servizio.. [2](#)

Bibliografia

Riferimenti bibliografici

James P. Womack, Daniel T. Jones. *Lean Thinking, Second Editon*. Simon & Schuster, Inc., 2010.

Siti web consultati

Angular Style Guide. URL: <https://angular.io/guide/styleguide> (cit. a p. 26).

Documentazione Angular. URL: <https://angular.io/docs>.

Documentazione Neo4j. URL: <https://neo4j.com/docs/>.

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