



POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

**Integrazione sistema
documentale a microservizi:
Realizzazione di due
applicazioni Web**

Relatore

Prof.ssa Tania Cerquitelli

Candidato

Stefano Cafieri

Tutore aziendale

dott. Pierantonio Manco

ANNO ACCADEMICO 2019-2020

*Alla mia famiglia,
per l'infinito supporto anche
nei momenti più difficili
di questo percorso universitario.
A voi dedico questo mio lavoro.
Grazie*

Sommario

La presente tesi tratterà del lavoro svolto sul sistema documentale dell'azienda AlmavivA, il quale permette di gestire un elevato numero di documenti, adattandosi a varie tecnologie che è possibile scegliere in fase di configurazione.

Il lavoro di tesi si concentra maggiormente sulla realizzazione di due Application, necessarie per l'interfacciamento degli utenti con il sistema documentale.

Indice

Elenco delle tabelle	7
Elenco delle figure	8
Introduzione	9
1 Stato dell'arte	15
1.1 Servizi REST	15
1.1.1 Client-Server	17
1.1.2 Stateless	17
1.1.3 Cacheable	17
1.1.4 Interfaccia uniforme	18
1.1.5 Sistema stratificato	18
1.1.6 Code on demand	18
1.2 JSON	19
1.3 Web Application	20
1.3.1 Web-App vs App native	22
1.3.2 Progressive Web Application	25
1.4 Angular	25
1.4.1 Componente	28
1.4.2 Funzionalità in Angular	29
1.4.3 Moduli	31
1.4.4 Pipe e Direttive	32
1.4.5 Servizi e Routing	33
1.4.6 Typescript	34
2 Sistema Documentale	37

3 Sviluppo, Test e Messa in Produzione delle due Web Application	43
3.1 Sviluppo	43
3.1.1 Promise	45
3.1.2 Grafica	46
3.1.3 Single Page Application	47
3.1.4 Pipe	47
3.1.5 Angular Material e lo stile adottato	47
3.1.6 Comunicazione tra componenti	49
3.1.7 Elementi delle due console	51
3.1.8 Resizing	52
3.2 Console Utente	54
3.3 Console Amministrativa	61
3.4 Testing	69
3.5 Messa in produzione	72
4 Conclusioni e Sviluppi Futuri	75
Bibliografia	79

Elenco delle tabelle

1.1	Confronto tra Web Application e Applicazione nativa	24
3.1	Confronto tra Console utente e Console amministrativa	44

Elenco delle figure

1.1	Esempio di componenti in un'applicazione	28
2.1	Composizione del sistema documentale	39
3.1	Finestra estesa del browser	53
3.2	Finestra ridotta del browser	54
3.3	Pagina iniziale della console utente	55
3.4	Menù di ricerca	56
3.5	Menù di ricerca esteso per tipologia	56
3.6	Creazione di un nuovo documento	57
3.7	Nuovo documento salvato	58
3.8	Contenuto del documento con aggiunta di un tipo	59
3.9	Lista dei documenti	63
3.10	Lista dei tipi	63
3.11	Dettaglio di un documento	64
3.12	Visualizzazione dettaglio di una tipologia	65
3.13	Visualizzazione statistiche della console	67

Introduzione

Nel corso degli anni il web ha assunto sempre più importanza per la popolazione mondiale, portando ad un'evoluzione rapidissima della tecnologia. Di giorno in giorno aumentano gli utenti connessi che utilizzano dispositivi quanto più diversi.

Un'immagine dell'evoluzione tecnologica è mostrata dal cambiamento radicale della navigazione sul web.

Inizialmente le pagine web non erano altro che documenti web statici legati tra loro da collegamenti ipertestuali. Essi erano dei file HTML realizzati a priori e inviati dal server ai vari client solo al momento in cui venivano richiesti. Questi file erano modificabili manualmente solo al lato server, mentre al client era concesso solamente di visualizzare le pagine e di navigare tra esse. L'esigenza di evitare queste operazioni manuali di modifica ha portato allo sviluppo di soluzioni applicative automatizzate che svolgano le stesse funzionalità in maniera rapida, sicura e soprattutto autonoma. Grazie a tali esigenze sono nati i siti web dinamici che, appoggiandosi ad esempio a un database per prelevare i dati, creano in maniera dinamica le varie pagine HTML da inviare al client. Quindi le pagine vengono prodotte in base all'interazione dell'utente con il sito web.

La navigazione è divenuta più interattiva tramite l'introduzione di un linguaggio di scripting lato client chiamato Javascript, che ha avuto il merito di rendere le pagine web ancor più dinamiche. Il codice in Javascript non viene eseguito sul server ma direttamente sul client, così che non tutte le modifiche effettuate dall'utente debbano coinvolgere il server.

Ad esempio, grazie a questo linguaggio si può validare un form fin da subito direttamente al lato client, così da evitare di inviare al server dei dati sicuramente non corretti. Una volta superata la prima validazione, per ragioni di sicurezza ne verrà effettuata una seconda anche lato server, dove però la maggior parte delle volte arriveranno dei dati correttamente validati. Si può quindi capire da questo esempio che il server verrà coinvolto solo quando i

dati immessi nel form sono corretti, evitando così ad esso di rispondere ad un client che compila un form in maniera errata.

Grazie a queste pagine web dinamiche e alla potenza sempre maggiore dei browser si è arrivati alla nascita di applicazioni web, chiamate più comunemente Web Application.

Il termine Web Application indica un'applicazione basata su un'architettura client-server che fornisce determinati servizi ai suoi fruitori tramite l'accesso al browser.

Esse sono nate per superare i problemi delle classiche applicazioni client-server in cui l'applicazione doveva essere installata su ogni macchina del client, portando una certa onerosità agli sviluppatori nei casi in cui fosse richiesto un aggiornamento. Diversamente dalle precedenti applicazioni appena illustrate, le Web Application vengono eseguite direttamente sul browser, prevenendo il problema legato agli aggiornamenti sulle singole macchine e offrendo altri numerosi vantaggi.

Le Web-App, abbreviazione con il quale vengono spesso chiamate, permettono di avere una user experience simile a quella di una classica applicazione desktop, ma con il vantaggio della portabilità e dell'immediatezza di utilizzo. Esse funzionano in maniera piuttosto semplice: il browser che esegue la Web-App invia tramite il protocollo HTTP delle richieste al server che le elabora e, facendo delle richieste al database per ottenere dei dati, invia la propria risposta al client. In base alla risposta ottenuta, sul browser si avrà l'aggiornamento della pagina visualizzata.

Molti dei siti più famosi su cui oggi si naviga sono delle Web-App: il servizio di posta elettronica Gmail, il servizio di storage Google Drive e anche Google Maps, il social network Facebook e l'enciclopedia Wikipedia.

Questi sono solo alcuni esempi di Web Application poichè ormai la direzione presa dalle grandi aziende, e più in generale dagli sviluppatori, è quella di realizzare applicazioni di questo tipo sia per avere una gamma più vasta di utenti che riescono ad usufruirne sia per avere vantaggi legati ai suoi minori costi di manutenzione.

L'obiettivo della seguente tesi è l'analisi e la realizzazione di due Web Application per il sistema documentale dell'azienda AlmavivA.

Si spiegheranno i motivi che hanno portato alla realizzazione di questa specifica soluzione, analizzandone i vari requisiti e mostrandone i suoi aspetti principali.

Si tratteranno inoltre concetti relativi al testing sulle applicazioni, per evidenziare la loro importanza nel contesto dello sviluppo di un prodotto.

L'azienda AlmagivA è un'azienda italiana leader nel campo dell'Information & Communication Technology (ICT) con sedi in tutto il mondo. Opera in molti specifici mercati: cybersecurity, banche, sanità, pubblica amministrazione, trasporti, energia e tanti altri. [1]

Il sistema documentale di cui si parlerà è stato sviluppato dall'ufficio Architetture di Torino il quale si occupa di fornire soluzioni distribuite su richiesta dei clienti e, come in questo caso, di realizzare nuovi prodotti aziendali. Oltre a questo prodotto ne sono presenti anche degli altri, alcuni dei quali già installati presso i vari clienti mentre altri ancora in fase di sviluppo.

L'idea dell'azienda è di fornire sempre nuove soluzioni per rispondere alle varie esigenze di mercato, ponendosi come obiettivo la continua ricerca di innovazione digitale.

La necessità di creare le due Web Application deriva dal fatto di voler dare un'interfaccia grafica al sistema documentale sviluppato fino a quel momento dall'azienda, in quanto il lavoro effettuato precedentemente su di esso è stato focalizzato sullo sviluppo del vero e proprio cervello del sistema, cioè il back-end.

Vista la richiesta del mercato è sorta, quindi, la necessità di sviluppare il front-end del sistema per un uso migliore e ottimale di esso. Infatti il cliente stesso potrebbe implementare la sua applicazione grafica richiedendo solo le API del sistema documentale, ma spesso si preferisce avere già l'applicazione sviluppata dall'azienda fornitrice del servizio poichè essa ne conosce già tutti gli aspetti e può sfruttarne tutte le funzionalità anche a livello grafico.

Vengono quindi soddisfatte le richieste dell'ideatore del progetto, facendo però anche riferimento alle richieste provenienti dal mercato. Infatti un'applicazione che funziona correttamente ma che rispecchia le esigenze solo di chi lo sviluppa, diventa poco utile a livello commerciale per essere venduto a clienti che potrebbero avere esigenze diverse.

L'idea è stata, dunque, quella di avere un'applicazione con una base solida e comune a tutti, a cui si potranno apportare delle modifiche in base alle esigenze dei propri clienti.

Ognuna delle due Web-App, chiamate anche console utente e console amministrativa, dovrà implementare le funzionalità necessarie per fornire un'esperienza d'uso ottimale in base al ruolo con cui queste applicazioni vengono usate, poichè ognuna delle due console sarà rivolta ad un pubblico diverso: la prima sarà rivolta ai semplici utenti che utilizzano il sistema, mentre la

seconda agli amministratori del sistema.

L'elaborato che si sta presentando è strutturato nel seguente modo:

- La seguente introduzione ha presentato un sunto generale sulla storia della navigazione sul web, introducendo anche le Web Application e il sistema documentale nel quale esse verranno integrate.
- Il primo capitolo presenta uno studio dello stato dell'arte nel quale si illustrano alcuni concetti e framework utilizzati per la realizzazione delle Web Application. Più in generale si descrivono i servizi REST, il formato JSON, le Web Application e il framework Angular. Per i servizi REST si illustra brevemente cosa sia lo stile architetturale REST e le principali linee guida che lo definiscono. Dopo di che si illustra brevemente il formato per lo scambio di dati chiamato JSON. Nel punto successivo riguardante le Web Application si mostrano le sue caratteristiche principali e i vantaggi di questa soluzione; inoltre si fa un confronto con le app native e si illustra un'evoluzione di esse. In merito ad Angular, invece, si descrivono le caratteristiche principali e le novità introdotte rispetto alle versioni precedenti; si illustrano inoltre gli elementi utilizzati per la scrittura del codice in Typescript, di cui si fornisce un'introduzione che evidenzia anche le differenze con Javascript.
- Il secondo capitolo presenta una spiegazione generica del sistema documentale, nella quale si illustrano i concetti principali che lo identificano e le tecnologie utilizzate. Si spiegano i motivi che hanno portato alla nascita di questo sistema, i requisiti sul quale è stato costruito e le caratteristiche principali. Inoltre si spiega come le due Web Application sviluppate vengano inserite all'interno del sistema documentale.
- Il terzo capitolo affronta i temi riguardanti le Web Application nel contesto del sistema documentale, illustrandone il loro sviluppo, la loro fase di testing e la loro fase di messa in produzione. Nella sezione dedicata allo sviluppo si descrivono le linee guida utilizzate nella stesura del codice e le differenze tra le due console create; inoltre si spiegano quali scelte sono state fatte per entrambe le soluzioni e quali elementi di Angular sono stati sfruttati maggiormente. Di seguito, tramite delle immagini catturate direttamente dalle console, si mostrano e si spiegano le funzionalità di ognuna di esse, illustrando anche gli elementi in Angular che le definiscono. Dopo la parte sullo sviluppo, si approfondiscono i concetti riguardanti la fase di testing. Si spiega inizialmente come questa fase

possa essere eseguita, mostrando successivamente degli esempi di test sul back-end e sul front-end del sistema documentale. Infine si mostra in cosa consiste la fase di messa in produzione, con relativo approfondimento sulla manutenzione del prodotto.

- Il quarto e ultimo capitolo affronta le tematiche finali del seguente lavoro di tesi, illustrando le conclusioni ottenute e i possibili sviluppi futuri sul sistema documentale e nello specifico sulle Web Application.

Capitolo 1

Stato dell'arte

Il sistema documentale, sia per la parte di back-end sia per la parte di front-end, è stato sviluppato utilizzando tecnologie moderne in modo da sfruttarne a pieno le loro potenzialità.

Di seguito si spiegheranno le tecnologie usate per lo sviluppo delle due Web Application, facendo anche riferimento a ciò che è stato implementato nel back-end e quindi direttamente riconducibile al successivo sviluppo front-end.

Si spiegheranno lo stile architetturale REST che sta alla base di tutto il progetto, il formato dei dati JSON utilizzato, gli aspetti legati ad una Web Application e infine la piattaforma Angular utilizzata per lo sviluppo delle due console.

1.1 Servizi REST

REST, acronimo di REpresentational State Transfer, è un insieme di linee guida che descrivono lo stile con il quale i dati possono essere trasmessi in un sistema distribuito.

Non si tratta dunque né di una tecnologia né di un protocollo, ma bensì di uno stile architetturale, poiché esso non va a definire ad esempio il formato esatto dei messaggi da scambiare (cioè la loro sintassi) o il protocollo di trasferimento da utilizzare. È uno stile che pone dei requisiti architetturali che i messaggi o il sistema devono soddisfare, utilizzando degli standard come XML o JSON per la rappresentazione dei dati, HTTP per il trasferimento di questi dati o altri.

REST è uno dei due approcci principali per la realizzazione di Web Service.

Un Web Service è un sistema software creato per supportare l'interoperabilità tra macchine che interagiscono tra loro su una stessa rete. Ognuno di questi servizi, comunicando tramite il protocollo HTTP, si mette a servizio delle varie applicazioni.

Un Web Service dichiara all'esterno dei servizi, identificabili come web API¹. Le web API espongono su Internet dati e funzionalità di un'applicazione; queste funzionalità vengono richieste tramite metodi HTTP e ad esse si replicherà con risposte in formati come JSON o XML.

Quindi un'implementazione si dice RESTful quando i servizi web seguono i principi REST, soddisfacendo tutti i vincoli architetturali indicati da esso. Questi servizi web si baseranno sul concetto di risorsa, elemento fondamentale in REST indicante una qualsiasi entità rappresentabile e di cui si parlerà successivamente nel principio REST di interfaccia uniforme.

Le API RESTful svolgono le operazioni CRUD² (creazione, lettura, aggiornamento e cancellazione di una risorsa) tramite i metodi HTTP:

- POST: crea una nuova risorsa;
- GET: restituisce una o più risorse esistenti;
- PUT: aggiorna una risorsa o, se non esiste, la crea;
- DELETE: elimina una risorsa.

L'Implementazione basata su REST presenta numerosi vantaggi: separazione tra client e server che consente di sviluppare i diversi componenti in maniera indipendente tra loro, alta scalabilità dovuta al fatto che server e client possono risiedere su server diversi, indipendenza delle API REST dal linguaggio o dalle tecnologie utilizzate per implementare i componenti. Riguardo quest'ultimo punto bisogna, per ogni linguaggio, mantenere solamente la coerenza sul formato delle richieste e delle risposte che ci si aspetta dal client o dal server in base a come esse siano state definite.

Di seguito si elencano i principi base da seguire per costruire un'architettura REST:

¹API: Application Programming Interface

²CRUD: Create, Read, Update, Delete

1. Client-Server
2. Stateless
3. Cacheable
4. Interfaccia uniforme
5. Sistema stratificato
6. Code on demand

1.1.1 Client-Server

Basato sul paradigma SoC³, questo principio stabilisce la separazione dei compiti tra client e server.

Infatti in un'architettura distribuita bisogna andare a definire le figure di client e server che avranno il compito di svolgere una determinata funzionalità. Avremo che il client invocherà, tramite un messaggio di richiesta, un servizio esposto dal server, che a sua volta si occuperà di elaborare la richiesta e di fornire una risposta al client.

1.1.2 Stateless

La comunicazione tra il client e il server deve essere "senza stato", cioè la richiesta del client verso il server deve contenere tutte le informazioni utili in modo tale che essa possa essere soddisfatta sul servizio chiamato. Viene quindi eliminata la correlazione tra le varie richieste, facendo così apparire ogni richiesta come se fosse la prima verso quel server.

Inoltre questo tipo di comunicazione dà la possibilità di poter scalare il sistema molto più facilmente, poichè non vi è più il problema di sincronizzare le informazioni sulla sessione degli utenti tra i vari nodi del server; questa informazione viene conservata lato client oppure demandata a un database.

1.1.3 Cacheable

Le risposte fornite dal server possono essere etichettate come cachabili o non cachabili.

³SoC: separation of concerns

Nel caso di risposte cachabili, le informazioni contenute al suo interno possono essere riutilizzate nelle richieste successive evitando l'interazione con il server poichè esse sono informazioni ancora valide ed attuali. Questo porta ad un conseguente miglioramento della scalabilità e delle performance del server.

Contrariamente, le risposte possono essere definite non cachabili se contengono informazioni su stati che in futuro potrebbero non essere più validi.

1.1.4 Interfaccia uniforme

La comunicazione tra client e server deve avvenire tramite un'interfaccia uniforme che preveda la definizione delle risorse e del formato nella quale esse devono essere rappresentate.

Una risorsa è l'elemento fondamentale del servizio REST: essa rappresenta un oggetto identificante un qualcosa di inerente al dominio applicativo al quale si accede tramite un identificatore globale (URI) e di cui client e server si scambiano rappresentazioni tramite un'interfaccia standard come HTTP. La rappresentazione più usata per le risorse è il formato JSON, ma vi sono altri formati utilizzabili come HTML, XML o altri ancora.

1.1.5 Sistema stratificato

Il sistema stratificato permette ad un'architettura di essere composta da più livelli intermedi posti tra client e server, indipendenti tra loro ed ognuno dei quali svolge un determinato compito. Ad esempio, strati intermedi possono essere composti da sistemi che svolgono azioni di caching, di sicurezza o di load-balancing.

Il vantaggio di avere un sistema stratificato con livelli indipendenti sta nella possibilità di poter aggiungere, cancellare o spostare i livelli intermedi in base al comportamento che si vuole ottenere nell'architettura, senza influenzare gli altri livelli già presenti.

1.1.6 Code on demand

Il seguente è l'unico principio opzionale. Indica la possibilità di estendere le funzionalità del client scaricando ed eseguendo del codice inviato dal server nella forma di Applet Java o script, semplificando e riducendo la complessità del client.

1.2 JSON

JSON, acronimo di JavaScript Object Notation, è un formato per lo scambio di dati usato nella programmazione web.

Tramite esso si rappresentano i dati che vengono inviati o ricevuti dal client o dal server in un contesto come il web, in cui possono essere coinvolte applicazioni con tecnologie e linguaggi differenti.

Nonostante sia basato su Javascript, questo formato è completamente indipendente dal linguaggio di programmazione in cui è usato.

La maggior parte dei linguaggi forniscono delle funzioni per manipolare gli oggetti JSON. Ad esempio in Javascript, tramite la funzione `JSON.parse()` si converte una stringa (rappresentante un oggetto JSON) in un vero e proprio oggetto JSON.

All'interno dell'intestazione di ogni richiesta o risposta verso/da un server viene introdotta l'informazione sul formato dei dati che è stato usato per il body del messaggio; in questo caso sarà presente "Content-Type: application/json".

Il formato JSON supporta molti tipi di dati: null, booleani, numeri, stringhe, array e array associativi.

Di seguito si mostra la rappresentazione di un oggetto JSON caratterizzato da un'ampia varietà di tipi di dato. Si può ad esempio notare come un oggetto JSON o un array vengano immessi come valori di campo dello stesso oggetto JSON.

```
{
  "nome": "Stefano",
  "cognome": "Cafieri",
  "data_nascita": {
    "giorno": 13,
    "mese": 07,
    "anno": 1993
  },
  "lingue": [ "it", "en" ],
  "residente": true
}
```

Spesso JSON è paragonato ad un altro linguaggio per la rappresentazione di dati, chiamato XML. Il JSON è molto più semplice da processare per le macchine poichè presenta una struttura più snella rispetto alla struttura più articolata di XML, caratteristica che però risulta più svantaggiosa per l'umano. Tramite XML infatti si possono creare documenti strutturati, in modo

che siano facilmente leggibili per l'umano, a meno che questi non crescano troppo in dimensione.

Entrambe queste due soluzioni, però, sono state studiate per il trasporto in maniera strutturata di differenti tipi di dato. [6]

1.3 Web Application

Una Web Application, cioè un applicazione web chiamata più comunemente Web-App, è un programma applicativo distribuito che permette di usufruire, tramite accesso da browser, di determinati servizi residenti su un server remoto.

Essa quindi comprende sia la parte relativa all'interfaccia visibile all'utente con la quale esso interagisce, sia la logica lato client nella quale si vanno a definire le funzionalità degli elementi visibili e le loro interazioni con il back-end; quest'ultimo contiene tutte le funzioni che svolge il server, come ad esempio il salvataggio e l'elaborazione delle informazioni.

Queste applicazioni possono essere sviluppate per molti usi differenti, quali ad esempio siti di e-commerce, webmail o altro.

Le Web Application hanno avuto sempre più importanza con l'avvento del modello SaaS⁴ (acronimo di "software come servizio"). Questo è un modello di distribuzione del software nella quale un provider di terze parti ospita un'applicazione web e la rende disponibile ai suoi client tramite Internet, sottoscrivendo un abbonamento che non ne dà il possesso ma il diritto di utilizzo. Il codice sorgente dell'applicazione è uguale per tutti i client e quando vengono fornite nuove funzionalità oppure vengono corretti alcuni bug, le nuove modifiche vengono rese disponibili a tutti i client.

Il modello SaaS è uno dei servizi principali del cloud computing al cui software si accede ad esempio tramite API RESTful.

Queste applicazioni nascono per superare i problemi legati alla generazione di pagine web statiche da parte del server. Le pagine web sono dei file statici in HTML che vengono presi dal server e visualizzati interamente nel browser. Con questa nuova soluzione invece le pagine web vengono generate

⁴SaaS: Software as a service

dinamicamente al momento della richiesta del client, in base alle operazioni che egli compie e grazie a degli algoritmi che girano sul server che sono in grado di risolvere problemi complessi.

Quindi si ha che il browser scarica inizialmente dal server tutte le risorse rappresentanti l'intera pagina e ne mostra in maniera dinamica solo le parti che in quel momento sono utili all'utente.

Ad esempio, se una pagina presenta una lista di oggetti corredata da un certo numero di componenti, al momento in cui viene richiesto l'aggiornamento della lista (ad esempio perchè si è effettuata una ricerca) non verrà scaricata dal server l'intera pagina, ma verranno scaricati, oltre alla lista, solamente i componenti che si saranno aggiornati. Gli altri componenti che rimarranno immutati saranno già presenti nel browser e quindi non sarà necessario scaricarli nuovamente.

Una Web Application è un'applicazione Multi-Tier⁵ poichè si sviluppa su più livelli logico funzionali (in tre livelli nella maggior parte dei casi):

- Livello di Presentazione: associabile al front-end, rappresenta l'interfaccia dell'applicazione con cui l'utente interagisce per immettere i dati e leggere i risultati (WebServer e pagine HTML);
- Livello Intermedio: associabile al back-end, rappresenta la logica di elaborazione poichè elabora i dati e in base alle richieste dell'utente genera dei risultati (moduli di un application server);
- Livello Dati: rappresenta un DBMS, indipendente dalle applicazioni server, che viene usato per la persistenza dei dati. [9]

Ognuno di questi livelli riesce a comunicare solo con i livelli adiacenti assumendo nella comunicazione il ruolo di client o di server. Inoltre essi sono moduli indipendenti, per cui una modifica a uno dei livelli non si ripercuote direttamente su un altro livello.

Parlando più specificamente del front-end, esso è costruito usando linguaggi come HTML, CSS e JavaScript, i quali sono supportati da gran parte dei browser.

Uno dei vantaggi principali di una Web-App è la completa indipendenza dal sistema nel quale viene eseguita. Infatti non ha importanza che venga eseguita su una macchina Windows o su una macchina Linux poichè, in

⁵Multi-Tier: Multi livello

entrambi i casi, è necessario solamente un browser web. L'unico vincolo presente è che il browser sia compatibile con questo tipo di applicazioni, ma questo è un problema relativo poichè tutti i browser più diffusi ne risultano compatibili.

Questo vantaggio quindi semplifica di molto il lavoro degli sviluppatori che non devono sviluppare un client per uno specifico computer o sistema operativo; chiunque abbia accesso ad Internet e disponga di un browser compatibile, può utilizzare la Web-App senza bisogno di alcuna installazione.

Il loro obiettivo è quello di adattarsi al meglio con la piattaforma sulla quale vengono lanciate.

Ulteriori vantaggi di questa soluzione sono:

- aggiornamento automatico e indipendente dalle scelte degli utenti, che quindi eseguiranno tutti la medesima, e anche ultima, versione;
- bassi costi di realizzazione e manutenzione;
- alta scalabilità;
- modularità;
- fase di testing poco complessa;
- facilità di manutenzione;
- rari problemi di compatibilità.

1.3.1 Web-App vs App native

Le app native sono applicazioni installabili che sono state sviluppate specificamente per una particolare piattaforma o device. Per determinate funzionalità sono utilizzabili anche offline.

Queste applicazioni possono sfruttare i componenti hardware del dispositivo nel quale vengono installate, come ad esempio la fotocamera o il GPS, per ampliare le loro funzionalità, a differenza delle Web-App nelle quali ciò non è sempre possibile, poichè esse possono essere eseguite su qualunque dispositivo non necessariamente provvisto di quello specifico hardware.

Una combinazione delle due soluzioni porta alla nascita di applicazioni ibride: applicazioni simili a Web-App, per funzionalità e navigabilità, ma installate sul dispositivo. Permettono di far funzionare l'applicazione su diversi dispositivi, programmando una sola volta e immettendo il codice sorgente in dei

contenitori specifici per ogni piattaforma, perdendo però in termini di integrabilità e prestazioni. [10]

Concludendo, dal confronto delle tre soluzioni si evince che:

- le applicazioni native hanno bisogno di maggiori sviluppi poichè devono essere progettate per i singoli sistemi nei quali devono essere installate (bassa portabilità). Hanno inoltre bisogno di una fase iniziale di installazione e configurazione. Tutto ciò però garantisce una maggiore integrazione con i device, performance migliori e consumi più bassi.
- le Web-App non devono essere installate sul dispositivo e il loro sviluppo è più rapido grazie al fatto che esse funzionano su tutti i sistemi grazie alla presenza dei browser. Un aspetto negativo è rappresentato dal fatto che il loro utilizzo potrebbe portare a performance più basse e limitate rispetto alle applicazioni native.
- le soluzioni ibride presentano alcuni vantaggi di entrambe le soluzioni, come la singola programmazione e il costo ridotto, ma portano anche svantaggi in senso di performance.

Nella tabella 1.1 si è fatto un confronto più dettagliato tra le due soluzioni, prendendo in considerazione gli aspetti più importanti riguardanti questi tipi di applicazione.

	Web-Application	App nativa
Portabilità	Alta (necessario solo un browser)	Bassa (sviluppo diverso per ogni sistema/device)
Integrazione	Bassa (sviluppata per funzionare su tutti i sistemi)	Alta (sviluppata per un singolo sistema)
Performance	Medie	Alte (se ottimizzata)
Costi di sviluppo	Bassi	Più alti
Funzionamento offline	No	Si
Installazione	No	Si
Aggiornamento	Automatico	Manuale

Tabella 1.1. Confronto tra Web Application e Applicazione nativa.

Osservazione 1 Dalla tabella si può vedere che entrambe le soluzioni hanno vantaggi e svantaggi. La soluzione migliore quindi si basa sulle proprie esigenze in termini economici e di sviluppo, in considerazione anche dell'uso che si deve fare del prodotto e agli utenti a cui esso è rivolto.

1.3.2 Progressive Web Application

Un'evoluzione delle Web-App è rappresentata dalle Progressive Web Application (PWA), il cui scopo è stato quello di andare a ridurre sempre più il gap con le app native a livello di user experience e funzionalità.

Basandosi quindi sull'idea delle Web-App di poter essere eseguite su qualsiasi device che abbia un browser, le PWA vogliono innalzare il livello di queste ultime adattandosi al meglio alla piattaforma corrente, in modo del tutto simile alle applicazioni native.

Il termine Progressive, aggiunto alla precedente dicitura di Web-App, indica la possibilità da parte dell'utente di poter abilitare nuove funzioni in base a ciò che offre il device a livello di funzionalità proprie e di hardware, in aggiunta a quelle già presenti nell'applicazione base.

Molte di queste novità vengono fornite grazie all'esistenza di nuovi componenti, chiamati Service Worker, posti tra la connessione di rete e il nodo che eroga il contenuto. Ad esempio questi nuovi componenti forniscono il meccanismo di notifica push oppure, tramite un meccanismo di cache, permettono l'utilizzo offline della Web-App.

Una delle più grandi novità delle PWA è la possibilità data all'utente di poter salvare la Web-App nella schermata principale del dispositivo mobile così che essa appaia come un'applicazione nativa a tutti gli effetti, soprattutto in termini di navigazione e di interazione.

Risultano molto affidabili, con un caricamento delle pagine più veloce e con minore esposizione a connessioni instabili; non vengono scaricate da un app store e quando si avviano caricano sempre la loro versione più recente.

A differenza delle applicazioni native, sono emersi dei problemi sulla quantità di energia che esse consumano. Inoltre si è evidenziata la necessità di una costante connessione al server per ottenere il massimo dalle sue funzionalità e la mancanza di supporto da parte di alcuni sistemi ospitanti nel far fungere alcune funzionalità, quali ad esempio, il sistema di notifiche push su iOS.[11]

1.4 Angular

Angular è un framework Javascript open source, sviluppato da Google, che viene utilizzato per la realizzazione di applicazioni dinamiche web e mobile. Rilasciato abbastanza recentemente, è nato per andare a sopperire ai problemi di AngularJS, framework ritenuto troppo complesso e difficile da far

coesistere con le altre librerie Javascript. Nonostante ciò, Angular JS ha avuto un grande successo tra gli sviluppatori. Le versioni successive ad esso si discostano completamente da JS in quanto la versione 2 di Angular, da cui derivano tutte le successive versioni, è stata scritta quasi completamente da zero, non mantenendo la compatibilità con Angular JS.

Questo framework offre una serie di strumenti e funzionalità che garantiscono elevate performance e al contempo una semplificazione della fase di sviluppo.

Parte del lavoro degli sviluppatori si è concentrato nel creare un framework che permettesse un utilizzo ottimale dell'applicazione web, non solo sui browser dei computer, ma anche sui browser dei dispositivi mobili. Per fare ciò, l'applicazione si deve adattare alle dimensioni ridotte degli schermi di smartphone e tablet, oltre che alle loro prestazioni limitate.

Per migliorarne le prestazioni, si è deciso di snellire il framework tramite l'ottimizzazione del codice, consentendo così di avere tempi di avvio, di caricamento e di download sempre più brevi; inoltre il compilatore AoT⁶ riduce ulteriormente la dimensione dell'applicazione e ne velocizza ancor di più il tempo di avvio.

Alcune funzionalità introdotte permettono di scaricare solamente i file che contengono il codice sorgente delle parti dell'applicazione che servono in quel momento per il suo funzionamento, portando così alla realizzazione di applicazioni di tipo Single-Page, cioè consultabili su una singola pagina web. Inizialmente tutte le risorse presenti sul server vengono scaricate sul browser, per poi essere caricate dinamicamente nelle pagine di interesse solo quando sono necessarie per l'utente. Non si avrà più la necessità di ricaricare ogni pagina dal server. Oltre alla fase iniziale, si dialogherà con il server solo tramite API per la richiesta di determinati servizi.

Angular permette inoltre di creare Progressive Web App e si integra con altri strumenti per lo sviluppo di applicazioni Desktop e mobile sia per Android sia per iOS.

Un'importante semplificazione introdotta è stata lo sviluppo di un tool chiamato Angular CLI⁷. Tramite una serie di comandi da console, questo tool aiuta lo sviluppatore a creare la struttura già configurata dell'applicazione

⁶AoT: Ahead-of-Time

⁷CLI: command line interface

o anche elementi più specifici, anch'essi già configurati, come componenti, servizi, moduli e molto altro.

Angular CLI a quindi il compito, tramite semplici comandi, di semplificare allo sviluppatore la fase di sviluppo e di test delle applicazioni.

I comandi più importanti sono:

- "ng new <nome_applicazione>" per creare la struttura dell'applicazione
- "ng generate <schematic>" per creare un nuovo modello come ad esempio un componente, un direttivo, un modulo o altro
- "ng build" per compilare un'applicazione
- "ng serve" per visualizzare l'applicazione sviluppata sul browser

Visto il grande successo avuto, Angular vanta una grande community di programmatori in grado di dare supporto per tutti i problemi derivanti dallo sviluppo di un'applicazione. Questo può essere sicuramente considerato come uno dei motivi del grande uso dell'applicazione da parte degli sviluppatori web. [12]

Novità introdotte:

- eliminazione di concetti che rendevano Angular JS di una complessità troppo elevata, dando spazio all'elemento fondamentale, cioè il componente;
- maggiore modularità delle applicazioni;
- supporto per la scrittura del codice in nuovi linguaggi come il TypeScript, il quale permette di identificare gli errori già durante la fase di compilazione;
- supporto di eventi come il touch e le gesture che lo rendono ottimo anche per sviluppi mobile;
- riduzione dei tempi di avvio e caricamento dell'applicazione;
- tempi di rendering e di rilevamento delle modifiche nei dati molto efficienti e veloci grazie al nuovo compilatore AoT, il quale fornisce al browser una versione già pre-compilata dell'applicazione rendendo il rendering immediato.

Di seguito si descrivono gli elementi più importanti per lo sviluppo in Angular. In conclusione si vedrà brevemente il linguaggio Typescript.

1.4.1 Componente

Il componente è l'elemento fondamentale di un'applicazione Angular. Rappresenta un'entità configurabile e personalizzabile che visualizza delle informazioni e con il quale l'utente può interagire.

Le applicazioni Angular non sono altro che un insieme di componenti, dove ognuno di essi in determinate pagine va ad occupare a livello grafico una parte dell'applicazione. Per favorire la modularità e la comprensione futura del codice, ogni componente può inglobare al suo interno altri componenti, andando così a creare una struttura gerarchica. In questo modo si evita anche di aumentare di troppo le dimensioni del codice di ogni componente.

La creazione dei componenti dipende da come si vuole strutturare l'applicazione e dai compiti che si vogliono dare ad ognuno di essi; ciò sottolinea il fatto che non esistono scelte corrette o sbagliate nella creazione di un componente poiché molto dipende dalle funzionalità e dalla lunghezza del codice che lo sviluppatore vuole dare ad esso.

Un esempio di scelta dei componenti, a livello grafico, è raffigurata nella figura 1.1.



Figura 1.1. Esempio di componenti in un'applicazione

[13]

Osservazione 2 Nella figura si vede come la pagina dell'applicazione sia divisa in tanti blocchi, dove ognuno di essi rappresenta un componente: vi è un componente per la barra superiore (rappresentata dalla scritta "Lista spesa"), uno per la barra di ricerca e il suo bottone, uno per la lista di cibi, e così via. Per ognuno di questi verrà definito il suo file HTML, il suo file CSS e il suo file Typescript. Ad esempio, per evidenziare la non unicità delle soluzioni, si poteva anche scegliere di suddividere in due il componente comprendente la barra di ricerca e il bottone; questa scelta poteva essere adottata, ad esempio, per definire uno stile o un comportamento speciale al bottone di ricerca.

Creare manualmente un componente può risultare non molto semplice. Per questo motivo ci viene in aiuto Angular CLI, il quale tramite il comando `ng generate component <nome_componente>` genera un nuovo componente; in assenza di altri flag opzionali, esso creerà una cartella con all'interno i tre file già configurati (un file Typescript, uno HTML e uno CSS) e pronti per essere implementati.

Il file Typescript conterrà la vera e propria definizione del componente che non è altro che una classe a cui è applicato il decoratore `@Component` e contenente proprietà e metodi del componente. Il file HTML conterrà un template che rappresenta la struttura grafica del componente, mentre il file CSS conterrà lo stile e l'aspetto del componente.

I componenti hanno un ciclo di vita ben definito. Vengono creati dinamicamente in base all'interazione con l'utente, che quindi ne determina la creazione, l'aggiornamento e la distruzione in base alle scelte che egli compie. Si parte con la fase di inizializzazione del componente, successivamente ne vengono inizializzati i contenuti sui quali vengono anche fatti dei controlli, e infine vi è la fase di distruzione del componente. Per ognuna di queste fasi e nelle sotto-fasi più specifiche, si scrive il codice indicante il comportamento che il componente deve avere in quella determinata fase del suo ciclo di vita.

1.4.2 Funzionalità in Angular

In Angular vi sono alcune funzionalità importanti che permettono la comunicazione tra DOM⁸ e il modello dati del componente (il file Typescript).

⁸DOM: Document Object Model, cioè una rappresentazione ad oggetti di una pagina web generata dal browser nell'interpretare il file HTML, cioè il template del componente

Questa comunicazione avviene tramite un meccanismo chiamato Data Binding, che può essere di due tipi:

- Unidirezionale (one-way data binding)
- Bidirezionale (two-way data binding)

Il **binding unidirezionale** viene utilizzato nel caso in cui si vogliono passare delle informazioni dal modello dati al template o viceversa.

Nel primo caso, chiamato interpolazione, al fine di visualizzare nel template una proprietà del componente definita in Typescript, si utilizzano le doppie parentesi graffe con all'interno l'espressione

$$\{\{proprietàComponente\}\}$$

mentre per assegnare a un elemento del DOM una proprietà del componente si utilizzano le parentesi quadre.

$$[elementoDOM] = "proprietàComponente"$$

Nel secondo caso invece, chiamato event binding ed utilizzato per il binding da template a modello dati, si utilizzano le parentesi tonde con all'interno l'evento che è stato compiuto sull'elemento del DOM, in modo da mappare l'azione che è stata svolta e un metodo nel modello dati, come ad esempio il click su un bottone.

$$(click) = "eseguiAzione()"$$

Il **binding bidirezionale** viene utilizzato per aggiornare automaticamente la proprietà nel modello dati e la sua rappresentazione nel template, in modo che la modifica su una delle due parti si ripercuota anche sull'altra parte. Si utilizza solitamente nei form quando si vuole far sì che una proprietà venga allo stesso tempo visualizzata nel template e modificata nel modello dati; ciò è fattibile tramite l'espressione

$$[(ngModel)] = "proprietà"$$

dove ngModel è una direttiva di Angular.

Questo permette di abbreviare notevolmente il codice; infatti l'espressione precedente si può scrivere in forma estesa come

$$[value] = "proprietà"$$

(input) = "proprietà = \$event.target.value"

dove la prima espressione permette di visualizzare la proprietà nel template, mentre la seconda permette di modificare la proprietà nel modello dati ogni qual volta questa viene modificata nel template. Da questo breve esempio si può notare tutta l'efficacia e l'importanza del binding bidirezionale. [14]

1.4.3 Moduli

Per avere una migliore organizzazione dell'applicazione, Angular mette a disposizione i moduli.

Questi moduli sono delle classi dove è possibile raggruppare e definire tutti gli elementi collegati tra loro che hanno lo scopo di conseguire una determinata funzionalità comune. I moduli quindi non fanno altro che seguire l'approccio modulare di Angular. Essi sono contraddistinti dalla presenza del decoratore @NgModule che permette di definirne le caratteristiche.

Tramite la definizione di questi moduli, si avrà una strutturazione dell'applicazione in entità collegate tra loro, come ad esempio componenti, pipe e direttive.

Le applicazioni in Angular presentano già un modulo iniziale chiamato AppModule che è il contenitore iniziale nel quale possono essere inclusi tutti i componenti, gli altri moduli, le pipe e altri elementi; questo modulo può bastare nel caso di piccoli progetti.

In caso contrario, cioè per progetti abbastanza grandi, è più opportuno organizzare gli elementi in moduli separati così che ognuno di essi comprenda elementi che svolgono un compito a loro comune.

Al decoratore @NgModule si passa un oggetto con diversi campi da valorizzare (non tutti obbligatori), tra cui i principali sono:

- declarations, proprietà che indica tutti i componenti, pipe e direttive del modulo;
- imports, proprietà che elenca tutti gli altri moduli esterni che sono utilizzati nei componenti del seguente modulo;
- bootstrap, proprietà presente solo in AppModule che indica qual è il componente di avvio dell'applicazione (solitamente AppComponent).

1.4.4 Pipe e Direttive

Le pipe sono uno strumento molto utile in Angular in quanto, chiamate in un template HTML, permettono di trasformare (o formattare) un qualsiasi valore secondo una funzione definita precedentemente. Esempi di pipe sono la conversione della stringa in maiuscolo o in minuscolo, la restituzione di una sotto stringa o la restituzione dell'ora in un determinato formato.

Ogni pipe può o meno richiedere anche dei parametri aggiuntivi. Più pipe possono essere combinate tra loro in modo tale da subire più trasformazioni consecutive, eseguite sempre nell'ordine in cui esse sono scritte.

Lo sviluppatore può definire delle nuove pipe personalizzate oppure utilizzare le pipe di sistema. Degli esempi di pipe di sistema sono: uppercase, lowercase, json (trasforma un oggetto in un oggetto JSON), date (formatta una data), number (formatta valori numerici) e altre.

Di seguito si mostra un esempio di utilizzo di pipe in un file HTML, il cui risultato sarà la stringa "ANGULAR" :

```
< span > {{ "Angular" | uppercase }} < /span >
```

La definizione di una pipe avviene tramite una classe a cui viene aggiunto il decoratore @Pipe, usato per definire al suo interno il nome della stessa pipe. Tramite questa classe si implementa l'interfaccia PipeTransform, contenente il metodo transform() dentro il quale vi è la logica di trasformazione della pipe. [15]

Un altro strumento molto importante nello sviluppo in Angular sono le direttive. Ne sono state definite tre: [16]

- i componenti, sono una particolare direttiva con template
- le direttive strutturali, che permettono di modificare la struttura del DOM tramite l'inserimento o la rimozione degli elementi; un esempio sono:
 - *ngIf: per decidere se inserire o meno l'elemento
 - *ngFor: per iterare su un gruppo di elementi
- le direttive di attributo, che permettono di cambiare il comportamento o l'aspetto di un elemento, tramite l'assegnazione ad esse di un'espressione indicante il comportamento voluto; un esempio sono:

- `ngClass`: per aggiungere dinamicamente classi CSS ad un elemento
- `ngStyle`: per impostare lo stile di un elemento in base al risultato di un'espressione

1.4.5 Servizi e Routing

I servizi in Angular sono delle classi che svolgono determinati compiti riguardanti la gestione dei dati, utili sia per i vari componenti sia per gli altri servizi. Il fine di questi compiti spesso è rappresentato da interazioni verso un database per la richiesta o la manipolazione di dati. Il front-end chiamerà i servizi del back-end per la gestione di dati memorizzati in un database.

Secondo il principio di Single Responsibility, queste funzionalità vengono introdotte all'interno di un servizio, e non di un componente, poichè si vuole mantenere la separazione dei compiti tra i vari elementi. In questo caso il componente si deve occupare solo dell'interazione con l'utente, mentre il servizio si occupa di gestire il modello dei dati.

Inoltre sempre secondo questo principio, in un'applicazione Angular dove sono definiti più servizi, ognuno di essi deve svolgere un insieme di funzioni correlate tra loro e diverse da quelle di altri servizi.

A livello implementativo, quando all'interno di un componente si vuole utilizzare un metodo del servizio, si "inietta" un'istanza del servizio in esso, cioè si passa un'istanza del servizio come parametro del costruttore. Ciò funziona grazie all'implementazione del pattern di Dependency Injection all'interno del framework. Questo pattern permette di iniettare le classi tramite un oggetto Injector che, previa registrazione in esso delle classi da iniettare, permette di risolvere il problema delle dipendenze.

Angular CLI semplifica la creazione di un servizio. Tramite il comando `ng generate service <nome_servizio>` viene generato un nuovo file con la struttura e le dipendenze già definite del servizio e all'interno del quale si devono definire solo i metodi da utilizzare. Infatti il servizio presenterà già il decoratore `@Injectable` a cui viene passato un oggetto che permette al servizio stesso di registrarsi con l'Injector, portando ad una ottimizzazione dell'applicazione rispetto alla vecchia procedura di registrazione delle classi presente nelle versioni precedenti del framework. [17]

Angular implementa anche il meccanismo del Routing, cioè la possibilità in un'applicazione web di navigare tra le sue diverse view evitando di ricaricare

le pagine sul browser. Ciò permette di creare un'applicazione SPA⁹, la quale presenta una view diversa in base all'URL digitato sul browser, consentendo di muoversi dinamicamente tra di esse caricando solo inizialmente la pagina. Questo è possibile facendo una mappatura tra gli URL e vari componenti.

1.4.6 Typescript

Typescript è il linguaggio di programmazione sviluppato da Microsoft, con il quale è stato riscritto completamente Angular2.

Pensato per la realizzazione di applicazioni complesse, questo è il linguaggio consigliato per lo sviluppo di applicazioni web in Angular.

È un super-set di Javascript poichè si basa interamente su esso aggiungendo alcune nuove funzionalità; infatti tutto il codice JS¹⁰ è anche codice TS¹¹, ma non viceversa. Per questo motivo è possibile scrivere del codice Typescript utilizzando solo Javascript, senza però usufruire delle funzionalità che Typescript offre.

Questo linguaggio è nato per superare i problemi legati all'uso di JS. Infatti JS è un linguaggio dinamicamente tipizzato che esegue conversioni implicite tra tipi senza fare preventivamente dei controlli statici e portando così a degli errori difficilmente interpretabili a run-time (le variabili non hanno un tipo definito, ma vengono tipizzate dall'interprete che esegue JS).

In contrapposizione a ciò che avviene in JS, per TS si è deciso di inserire dei controlli statici sui tipi durante la fase di compilazione, rendendo così TS un linguaggio fortemente tipizzato. Se vengono riscontrate irregolarità sui tipi, questo controllo porta alla nascita di errori in fase di compilazione.

In Typescript inoltre le variabili possono essere definite di uno specifico tipo direttamente dallo sviluppatore, così da non lasciare alcun beneficio del dubbio all'interprete.

Il compilatore in Typescript prende il nome di transpiler. Il suo compito è quello di prelevare il codice sorgente del programma e tradurlo in codice sorgente di un altro linguaggio.

Viene usato questo tipo di compilatore perché i browser non supportano il linguaggio Typescript. Per questo motivo, le applicazioni inizialmente vengono scritte in TS e successivamente vengono transpilate in JS per permettere

⁹SPA: Single Page Application

¹⁰JS: Javascript

¹¹TS: Typescript

il loro funzionamento nel browser.

Volendo fare un confronto tra Javascript e Typescript, si può prendere come esempio una funzione che effettua il quadrato di una variabile, passata come parametro.

In Javascript, il parametro passato è una variabile non tipizzata. Quando la funzione viene chiamata passandole come parametro una stringa, che è una variabile sulla quale non può essere effettuata l'operazione di moltiplicazione, il compilatore non lancerà nessun errore. Ci si accorgerebbe dell'errore solamente in fase di run-time, quando si cercherà di fare il quadrato di una stringa.

In Typescript, questo problema non si pone se si tipizza il parametro passato alla funzione. Infatti in questo caso, l'errore legato al passaggio di un parametro non compatibile a quello voluto dalla funzione, cioè una stringa invece di un numero, risulterebbe già in fase di compilazione.

L'esempio evidenzia la possibilità di continuare ad usare JS con la sua sintassi e le sue funzionalità, aggiungendo però delle caratteristiche nuove a vantaggio dello sviluppatore.

Javascript si presta meglio per essere utilizzato in progetti di dimensioni ridotte, mentre Typescript nasce proprio per sviluppare applicazioni Javascript di elevate dimensioni. [18]

L'uso di Typescript in Angular porta alcuni vantaggi:

- Migliora la leggibilità
- Aiuta a trovare errori in fase di compilazione grazie alla tipizzazione dei dati in number, bool, string o, per le strutture, in any.
- Usa il paradigma della programmazione ad oggetti, grazie alla definizione di classi ed interfacce e grazie all'uso dell'ereditarietà
- Nessun problema di compatibilità con codice già scritto in Javascript

Capitolo 2

Sistema Documentale

Il sistema documentale realizzato è un applicativo per la gestione di file e dei relativi metadati. Vi si accede mediante custom RESTful API seguendo le linee guida REST.

I sistemi documentali tradizionali fin qui realizzati sono caratterizzati dall'essere applicazioni monolite, cioè realizzate come un unico grande blocco mancante di modularità. Questo tipo di sistema ingloba una serie di funzionalità predefinite e dialoga con gli altri strumenti aziendali in maniera già definita in fase di progettazione. Un documentale tradizionale ha quindi una serie di complessità da affrontare.

Ad esempio, una di queste complessità può essere rappresentata dalla dipendenza da uno specifico database. Se il sistema utilizza Oracle, le macchine che ospitano il sistema devono aver installato Oracle. Questo è uno svantaggio nel caso in cui le macchine usino già altri database, in quanto adesso è necessaria l'installazione di un database aggiuntivo.

Il sistema potrebbe anche utilizzare il file-system per la memorizzazione dei dati, ma ciò rappresenta un limite dal punto di vista della scalabilità orizzontale in quanto il file-system è un sistema di storage limitato ad un organizzazione gerarchica di documenti in cartelle e sottocartelle.

La domanda posta inizialmente sul perchè serva un nuovo sistema documentale ha trovato risposta nella necessità di avere una struttura dove tutti i documenti siano:

1. condivisibili con il resto della propria struttura;
2. ricercabili;

3. versionabili, cioè la possibilità di poter ricostruire la storia del documento facendo riferimento alle sue varie versioni;

Trovato il motivo per la sua realizzazione, successivamente si sono posti degli obiettivi che la soluzione deve raggiungere. Essi possono essere riassunti nei seguenti punti:

- alta affidabilità del sistema, cioè la sua capacità di essere:
 1. funzionante, caratteristica assicurata dalla qualità dell'hardware che sta alla base del sistema;
 2. disponibile per il maggior numero di ore possibili;
 3. esente da errori. [2]
- integrità dei dati, cioè la garanzia di ottenere la completezza e la conformità dei dati tramite una serie di processi;
- estensibilità e manutenibilità, cioè la facilità con la quale il sistema potrà essere modificato in futuro per correggere nuovi bug scoperti o per implementare nuove funzionalità;
- scalabilità orizzontale, cioè la possibilità di poter aumentare le prestazioni del sistema aggiungendo nuove entità hardware o software ed ottenendo un funzionamento coerente con quello precedente;
- capacità di essere integrato nei sistemi di sicurezza aziendali;
- facilità di integrazione con l'infrastruttura già esistente;
- indipendenza da uno specifico database, così da poter scegliere se usare un database relazionale o non relazionale oppure salvare i dati su file-system.

Nello sviluppo del sistema sono stati seguiti i principi SOLID¹, cioè un insieme di linee guida per sviluppare un software leggibile, estendibile e manutenibile.

Volendo focalizzarsi su uno di questi principi, il "Principio di singola responsabilità" assume notevole importanza nella realizzazione del sistema. Esso

¹SOLID è un acrostico derivante dai principi: Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion

afferma che ogni entità del programma debba svolgere una sola funzionalità, incapsulata nella stessa entità, così da avere una singola responsabilità. Con entità si possono intendere sia gli elementi più piccoli come metodi o variabili sia gli elementi più grandi come le classi. Nel caso di questo sistema documentale, il principio si espande anche al sistema stesso, nel senso che esso deve avere come unico compito quello di salvare e gestire i documenti. Qualsiasi altro compito che si voglia assegnare a questo sistema deve essere affidato ad un sistema esterno, che a sua volta svolgerà solo un determinato compito e alla fine verrà messo in comunicazione con il sistema documentale.

La soluzione proposta per questo sistema è una piattaforma modulare nella quale le funzionalità sono integrate in base alle esigenze del cliente e in base all'integrazione con l'infrastruttura nella quale viene installata. Per raggiungere questi obiettivi l'architettura è basata su diversi componenti in grado di comunicare con il sistema aziendale e ognuno di essi fornisce una funzionalità che va a contribuire alla formazione del prodotto finale (molte di esse sono obbligatorie, altre sono opzionali).

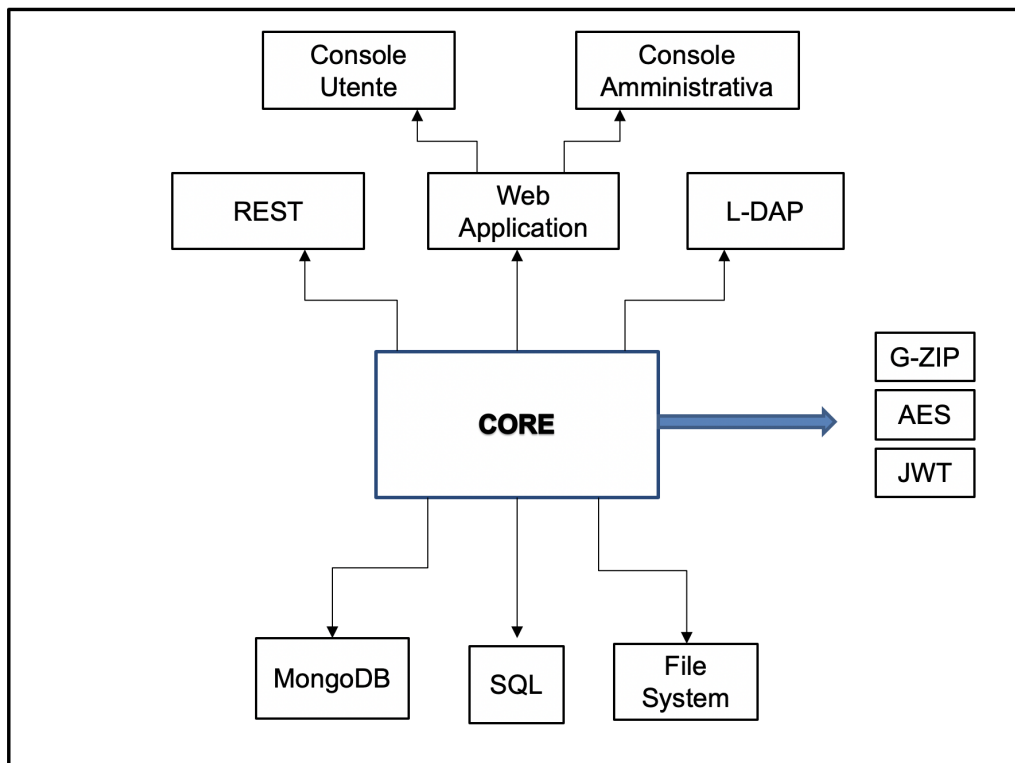


Figura 2.1. Composizione del sistema documentale

Nella figura 2.1 è raffigurato lo schema che mostra i vari componenti del sistema documentale. Oltre agli elementi di cui si parlerà successivamente, sono presenti: L-DAP per l'autenticazione degli utenti nel sistema, JWT per l'autenticazione delle richieste nei Web service, altri possibili servizi di cui si può usufruire come la compressione e la cifratura.

Il componente principale, denominato 'core', rappresenta il motore del documentale. Esso espone i suoi micro-servizi tramite delle API secondo lo standard REST. È facilmente pluggabile, cioè può interagire con altro software per estendere le sue funzionalità di base.

Per lo storage dei dati si appoggia a uno dei diversi sottosistemi selezionabili al momento della composizione e dell'installazione della piattaforma (DB SQL, DB NoSQL, Filesystem). Questo accesso ai dati è affidato ad un componente apposito che è configurabile per usufruire di diversi sistemi per il salvataggio dei dati. Infatti una soluzione può utilizzare uno stesso sistema di salvataggio sia per i metadati che per i dati veri e propri, oppure può scegliere di usare due sistemi differenti dove ognuno di essi ha lo scopo di salvare uno dei due possibili tipi di dato. La soluzione consigliata si basa sull'utilizzo di MongoDB poichè garantisce performance migliori al crescere delle chiamate al database, consentendo anche di memorizzare documenti con schema dinamico in stile JSON.

I restanti componenti del sistema consentono di configurare l'applicazione secondo le proprie esigenze. Alcuni di essi sono mutuamente esclusivi, come ad esempio l'archiviazione su MongoDB e su file System, mentre altri, come la compressione dei dati prima della loro scrittura, sono opzionali.

È proprio la modularità del sistema che permette di inserire, in fase di installazione, delle funzionalità aggiuntive. Il sistema documentale di per se fornisce le API di accesso ai dati, la capacità di salvare i documenti, la ricerca per metadati, il versionamento e l'ottimizzazione dei dati; opzionalmente è possibile ad esempio: limitare l'accesso alla risorsa facendo il locking sul documento, crittografare il documento, inserire la ricerca full-text, avere la console utente/amministrativa.

Dal punto di vista della struttura implementativa della soluzione, essa si basa su un insieme di 'controller' che espongono i servizi della piattaforma per mezzo di diversi protocolli. Questi controller si appoggiano agli 'engine', nei quali è implementata la loro logica applicativa. I dati vengono gestiti tramite gli 'adapter', componenti che si interfacciano direttamente con il sistema che ospita i dati. Questi dati possono transitare attraverso degli interceptor che

svolgono su essi varie elaborazioni, consentendo il pre/post processamento dei dati. Ad esempio, sui dati possono essere fatte operazioni di cifratura o decifratura, possono essere aggiunti dei metadati oppure possono essere applicate procedure di validazione.

Di seguito vengono mostrate due possibili configurazioni del sistema:

- REST + MongoDB + zip + aes + JWT. In questa configurazione i servizi CRUD sono esposti tramite REST. Tutti i metadati e i contenuti veri e propri del documento vengono salvati su MongoDB, dopo che essi sono stati compressi e cifrati. Inoltre la sicurezza è gestita tramite un token JWT.
- REST + Oracle + File System. In questa configurazione i servizi CRUD sono esposti via REST, con salvataggio dei dati in chiaro su file-system e dei metadati su un database relazione.

All'interno di questo sistema sono inserite anche le due console di cui si parlerà in dettaglio nel prossimo capitolo. Esse non fanno parte dei componenti obbligatori e non sono mutuamente esclusive, poichè una console è rivolta agli utenti mentre l'altra è rivolta agli amministratori.

Un cliente può decidere di usufruire di entrambe le console, di una di esse oppure di nessuna. In quest'ultimo caso il cliente usufruirà delle solo API messe a disposizione dal sistema, implementando da sè, in caso di necessità, le due console. Questo perchè ogni cliente può avere necessità differenti e fruitori di varia natura; tutto ciò può portarlo alla decisione di implementare una console più complessa e più adatta alle proprie esigenze. Infatti da come si vedrà, le due console implementate hanno come presupposto principale il fatto di dover essere semplici e di facile utilizzo, per compiere e gestire le varie operazioni in maniera meno problematica possibile.

Per lo sviluppo delle due console si è utilizzato il framework Angular, adatto per sviluppare Web Application e, in questo caso, per soddisfare tutte le esigenze richieste.

Infine, per quanto riguarda le tecnologie impiegate per lo sviluppo del sistema documentale e delle Web Application, si sono utilizzate soluzioni piuttosto recenti e performanti come:

- MongoDB o SQL per la persistenza;
- Spring Boot per la parte di back-end;

- Java 8;
- standard REST;
- JWT, L-DAP, G-ZIP e AES;
- Angular e il linguaggio Typescript.

Capitolo 3

Sviluppo, Test e Messa in Produzione delle due Web Application

3.1 Sviluppo

In questa sezione si spiegheranno gli aspetti principali nello sviluppo delle due Web Application.

La prima Web-app sarà rivolta agli utenti che vogliono utilizzare il sistema documentale, mentre la seconda sarà invece rivolta agli amministratori che gestiscono il sistema; loro potranno effettuare modifiche su qualsiasi documento e potranno anche creare nuove tipologie da applicare ai documenti.

Una tipologia, applicabile opzionalmente ad un qualsiasi documento, non è altro che un insieme di campi rappresentanti la tipologia stessa. Una volta che viene selezionata all'interno di un documento, questa comporta l'aggiunta dei suoi campi la cui valorizzazione è obbligatoria, pena la non possibilità di salvare il documento. Un esempio di tipologia può essere rappresentata dalla carta d'identità, nella quale si possono definire i vari campi corrispondenti ai dati della persona, come il nome, il cognome e tutti gli altri dati personali.

Questa tipizzazione risulta molto efficiente in quanto uniforma i documenti con la stessa tipologia e applica ai suoi campi delle regole per la loro valorizzazione.

In merito alle due console, nella tabella [3.1](#) si sintetizzano le differenze

presenti tra esse.

Funzionalità delle due console		
	Console Utente	Console Admin
Creazione documento	SI	NO
Lettura e modifica documento	SI	SI
Ricerca documento	SI	SI
Cancellazione versioni del documento	NO	SI
Cancellazione dell'intero documento	SI	NO
Gestione Tipo	NO	SI
Uso dei tipi	SI	//
Visualizzazione statistiche del sistema	NO	SI

Tabella 3.1. Confronto tra Console utente e Console amministrativa.

Osservazione 3 Come si può evincere dalla tabella, la console utente permette la gestione dei singoli documenti, mentre la console amministrativa permette un controllo più ampio sui documenti già creati, oltre a gestire aspetti supplementari come i tipi e le statistiche.

Nello sviluppo di queste applicazioni si sono seguite delle linee guida per migliorare la qualità del codice prodotto. Di seguito ne vengono elencate le principali:

- codice semplice e leggibile, poichè meno righe di codice diminuiscono il pericolo di bug e permettono una manutenzione futura meno complicata e più rapida;
- numero di commenti limitati, in quanto un codice ben scritto non deve aver bisogno di commenti per essere compreso. La presenza di variabili e metodi denominati in maniera auto-esplicativa garantisce già la comprensione dei loro ruoli e delle loro funzionalità;
- utilizzo minimo di istruzioni condizionali, come ad esempio gli "if", poichè questo tipo di istruzioni rappresentano la fonte principale di bug in un programma. È richiesto quindi un uso piuttosto limitato di questi costrutti, il cui utilizzo per creare condizioni annidate è altamente sconsigliato;
- metodi brevi che svolgano solo un determinato compito;
- utilizzo di una specifica nomenclatura, coerente in tutto il codice, per la definizione dei nomi degli identificatori;
- assenza di codice duplicato;
- definizione di classi e metodi in modo tale da poterli riutilizzare in futuro. Questi elementi non devono essere implementati solo in funzione dell'utilizzo che se ne farà in quel momento, ma anche in funzione di un possibile futuro riutilizzo da parte di altri componenti. Ciò può essere fatto creando una struttura quanto più generica possibile, che però risponda correttamente allo scopo per la quale è stata creata.

Di seguito si andranno ad analizzare alcuni aspetti riguardanti entrambe le console. Successivamente, si vedrà singolarmente per ogni console come queste siano state realizzate.

3.1.1 Promise

In Javascript, la Promise è un elemento fondamentale che garantisce una corretta comunicazione asincrona con il server. Questo strumento permette di scrivere codice asincrono mantenendo la struttura lineare e poco complessa di una comunicazione sincrona, facendo in modo che le istruzioni successive ad una chiamata asincrona vengano eseguite solo dopo che sia arrivata la risposta alla chiamata precedente. Vengono così forniti allo sviluppatore dei

mezzi per facilitare la scrittura del codice.

Il meccanismo delle callback permetteva già di eseguire istruzioni solo dopo che era stata ricevuta la risposta ad una chiamata asincrona, a fronte però di una notevole aggiunta di complessità al codice.

Per questo motivo in Javascript sono state introdotte le Promise. Queste permettono di sapere se una chiamata asincrona è terminata correttamente oppure è fallita, chiamando a loro volta delle funzioni a seconda dell'esito della chiamata.

Tramite l'introduzione delle due parole chiavi "async" e "await" si è semplificata di molto la scrittura delle Promise.

L'istruzione await interrompe l'esecuzione dell'istruzione successiva, fin quando non viene risolta la Promise restituita dall'istruzione precedente. La funzione che contiene una o più await deve essere dichiarata di tipo async.

Nell'implementazione di queste Web-App, le await/async vengono sfruttate nell'utilizzo del servizio HttpClient per fare una richiesta al server, convertendo la risposta, che si trova sotto forma di observable, in una Promise tramite il metodo 'toPromise'.

Ad esempio, per implementare la GET sulle varie risorse del server è stato creato un metodo all'interno di uno dei servizi dell'applicazione. Si è definito il metodo di tipo async e la vera e propria chiamata verso il server è stata preceduta dalla parola chiave await. Ciò ha garantito il corretto avvenimento della chiamata asincrona. Inoltre, le operazioni successive a questa chiamata appena descritta, attendono il verificarsi della chiamata stessa per poter procedere nel proprio lavoro.

3.1.2 Grafica

A livello grafico, le due Web Application presentano un aspetto abbastanza simile poichè visivamente si basano sugli stessi principi.

Per la console utente si è voluta creare un'interfaccia semplice e intuitiva che metta in risalto tutte le funzionalità principali, consentendo all'utente con poca esperienza di avere una navigazione fluida e senza particolari complicanze.

Per la console amministrativa si è voluta creare un'interfaccia che permetta all'amministratore di effettuare tutte le operazioni di sua competenza, in maniera piuttosto diretta e veloce poichè si presuppone che egli abbia un'elevata

conoscenza del funzionamento del sistema.

3.1.3 Single Page Application

Entrambe le console sono costruite secondo il principio della Single Page Application. Si risparmiano continue richieste di intere pagine web al server, il quale metterà a disposizione delle API che permettono di richiedere uno dei servizi che esso espone.

Un esempio che può spiegare questo principio è rappresentato dalla pagina iniziale della console dove è presente una lista di documenti. Quando si fa una ricerca sulla lista non viene ricaricata l'intera pagina con tutti i suoi componenti, ma viene aggiornata solamente la lista di documenti.

Questo approccio, innovativo rispetto al vecchio approccio classico client-server, è stato scelto per spostare la complessità sul browser, diminuendo il carico sul server.

3.1.4 Pipe

Nello sviluppo delle due soluzioni sono state di fondamentale importanza le Pipe. Il loro scopo è quello di tradurre, nella lingua impostata in fase di configurazione della Web-Application, le varie informazioni mostrate all'utente. Infatti durante la fase di inizializzazione dell'applicazione, si sceglie una lingua tra quelle disponibili in modo tale che l'intera console venga mostrata nella lingua desiderata.

Nel codice sviluppato, le informazioni da mostrare sono scritte sotto forma di label che si riconducono alle corrispondenti traduzioni nei file della pipe. In questo momento le lingue disponibili sono italiano ed inglese, ma è possibile in maniera piuttosto semplice aggiungere nuove lingue tramite la creazione di un nuovo file contenente le label delle parole da tradurre.

3.1.5 Angular Material e lo stile adottato

Uno strumento utilizzato di grande efficacia è il toolkit Angular Material. Esso è formato da un insieme di componenti e direttive predefinite, già testati e riutilizzabili più volte.

Gli elementi di questo toolkit si basano su un design sviluppato dalla stessa Google, chiamato Material Design. Grazie a questo strumento si facilita la

creazione di un'interfaccia ricca, interattiva e adattabile al dispositivo sul quale viene mostrata.

I componenti principali presenti in Material sono: un'ampia gamma di form (date, radio button, menù a tendina, slider e altri), varie tipologie di bottoni, modelli a comparsa per visualizzare informazioni all'utente, strutture per visualizzare collezioni di record come tabelle e paginatori, menù di navigazione per passare da una pagina all'altra e vari layout.

Essendo il sistema documentale un prodotto commerciale, le sue console si devono adattare allo stile dei committenti. Ne devono rispecchiare i colori, lo stile e le immagini principali che lo contraddistinguono.

Ad esempio, essendo il comune di Venezia un richiedente di questo sistema, si è adattata la console utente al sito web dello stesso comune. Se ne è preso il colore, il logo e lo stile del testo in modo da rendere il sistema affine alle caratteristiche del sito web. Differentemente, la console amministrativa rimane sempre neutra ed uguale per tutti i clienti.

Le figure che si vedranno nelle pagine successive fanno invece riferimento alla versione base del sistema, non adattata a nessun cliente e corrispondente alla versione distribuita internamente all'azienda.

Questi adattamenti ai vari clienti sono possibili tramite un file di configurazione compreso nel progetto, dentro il quale c'è una corrispondenza tra la configurazione riguardante il cliente e il relativo logo, stile, colori o altro che lo interessa. Ciò permette di avere un progetto generico che, tramite un semplice comando, si adatta alla configurazione del cliente.

Il colore principale adottato in ogni soluzione farà parte di un tema che anch'esso viene definito in fase di configurazione.

Seguendo l'approccio di Angular Material, inizialmente si definisce un tema composto da più colori ognuno dei quali sarà usato in determinati contesti: il colore 'primary' sarà il colore principale e maggiormente usato nei componenti, il colore 'accent' sarà presente nei pulsanti di azione e negli elementi interattivi, il colore 'warn' sarà invece utilizzato per evidenziare singoli errori oppure per rappresentare intere finestre di errore.

Ognuna di queste scelte è fatta in relazione alla rilevanza visiva che si vuole dare agli elementi. Ad esempio come colore 'accent' è stato usato il colore rosso poichè esso è un colore d'impatto e una volta visualizzato richiama subito alla mente dell'utente una situazione di 'pericolo', che nel nostro caso non è altro che una situazione di errore. Come colore 'primary' viene invece scelto un colore più tenue che rappresenti una situazione di non anomalia e ispirato ai colori del committente che richiede il prodotto.

3.1.6 Comunicazione tra componenti

In questa sezione si affronta l'argomento riguardante la comunicazione tra i vari componenti al fine di scambiarsi dei dati.

La comunicazione può avvenire facilmente nel caso in cui i componenti sono annidati tra loro, cioè un componente padre contiene all'interno del suo template gli elementi HTML legati ai vari componenti figlio.

In questo caso la comunicazione avviene nel seguente modo: nel file HTML si usa il meccanismo di binding delle proprietà per indicare i dati da passare, mentre nel file Typescript si utilizzano i decoratori `@Input` (da padre a figlio) e `@Output` (da figlio a padre) per catturare l'informazione passata.

Nel codice del sistema documentale si è implementata la comunicazione da padre a figlio, indicando all'interno di un componente padre il nome del componente figlio seguito dal binding di una proprietà.

In questo esempio, la proprietà del padre chiamata "progressFather" viene passata al componente figlio, dove prenderà il nome di "progressChild".

```
<son-component [progressChild]="progressFather"></son-component>
```

Il componente figlio per catturarla usa nella sua classe in Typescript il decoratore `@Input` nel seguente modo:

```
export class SonComponent {  
  @Input() progressChild: number;  
  ...  
}
```

Diversamente nel caso di comunicazione da figlio a padre, si usa il decoratore `@Output` insieme ad un oggetto di tipo `EventEmitter`.

Nel file HTML si utilizza il binding degli eventi per catturare l'evento emesso dal figlio e lanciare un metodo:

```
<son-component (progressChange)="updateProgress($event)"></son-component>
```

mentre nel file Typescript si associa al decoratore `@Output` un oggetto di tipo `EventEmitter` e si comunica il dato da passare al componente padre tramite la chiamata del metodo `emit()` sull'oggetto definito precedentemente.

```
export class SonComponent {
  @Output() progressChange = new EventEmitter<number>();

  onClick(event: MouseEvent) {
    this.progressChange.emit("example");
  }
}
```

Comunicazione più complessa da gestire è quella che avviene tra componenti che non sono direttamente legati tra loro, cioè tra componenti non annidati.

Nello sviluppo della console utente, il problema si è posto nel passaggio dal componente "documento" al componente "documenti". Il primo contiene il dettaglio di un documento, il secondo contiene la lista di documenti. Nello specifico, il problema si sollevava nella fase post ricerca di un insieme di documenti. Facendo un esempio concreto, si effettuava una ricerca che produceva n° documenti, si apriva un documento tra gli n° ritornati, ma una volta che si tornava indietro per vedere gli altri $n^\circ-1$ documenti risultanti dalla ricerca, il componente contenente la lista di documenti filtrata veniva resettato e mostrava nuovamente tutti i documenti. Il risultato della procedura era la perdita della lista di documenti ricercati. Ciò è causato dal fatto che è stato rieseguito nuovamente il metodo di inizializzazione del componente. Esistono due possibilità per superare questo problema.

La prima è creare un nuovo servizio che, data la sua natura persistente durante l'intero funzionamento della Web Application, permette di memorizzare variabili ed informazioni per poi poterle utilizzare in futuro. La seconda è utilizzare il meccanismo di Session o Local Storage.

È stata utilizzata la seconda opzione, nello specifico il Session Storage, poiché in questo specifico caso garantisce la scrittura di meno codice a parità di funzionalità. Esso consiste nel salvare nel browser una coppia chiave valore che può essere utilizzata fino al termine della sessione, cioè fin quando l'utente non chiude il tab o la finestra in cui è eseguita l'applicazione; il valore salvato può anche essere un oggetto JSON.

Tramite questo meccanismo, ogni qualvolta si effettua una ricerca si salva nel browser un oggetto JSON con i campi della ricerca. In questo modo, quando si carica il componente che rappresenta la lista di documenti, esso verifica se questo oggetto JSON sia presente e in caso positivo filtra i documenti da visualizzare in base ai campi prelevati dal JSON; in caso negativo sarà visualizzata l'intera lista di documenti.

3.1.7 Elementi delle due console

In entrambe le console sono presenti degli elementi che svolgono le medesime funzionalità, poichè essi non sono altro che elementi base dell'applicazione.

Gli elementi di configurazione principali sono:

- `pom.xml`, cioè un file XML che contiene le informazioni sul progetto e le varie configurazioni usate da Maven¹ per farne il build. In queste due soluzioni sono state definite soprattutto le varie fasi del processo di build, le versioni delle tecnologie da utilizzare e le dipendenze del progetto, tramite le quali si scaricano tutti i pacchetti delle librerie necessarie, comprese le dipendenze stesse delle librerie.
- `angular.json`, cioè il file di configurazione che Angular CLI ha prodotto in fase di creazione del progetto e che viene modificato da esso stesso oppure dallo sviluppatore. In questo caso nel file si sono aggiunte, oltre alla configurazione base, le varie configurazioni aggiuntive come ad esempio la configurazione per il debug, quella per l'azienda stessa oppure quella specifica per i vari clienti. Per questo motivo, quando viene lanciata la Web Application con un determinato parametro corrispondente a una configurazione già definita, ad esempio quella del cliente, allora verranno caricati tutti i file appartenenti a quella configurazione e verranno settate tutte le impostazioni definite. In particolare per ogni cliente verrà caricato il logo, il file CSS che indica uno stile specifico e un file contenente alcune informazioni, come ad esempio la lingua da utilizzare.

Il progetto, come tutti quelli in Angular, parte inizialmente da tre file che si possono definire come facenti parte di un componente generico; essi sono un file `index.html`, un `main.ts` e uno `style.css`.

Il file `.css` comprende gli stili dei vari componenti predefiniti, mentre il file `.html` non è altro che la pagina iniziale che non viene però qui visualizzata, poichè nel suo `body` è presente solo il rimando ad un altro componente (`AppComponent`). Il file `main.ts` è il punto di partenza dell'applicazione poichè richiama il modulo principale dell'applicazione (`appModule`), il quale contiene il campo che indica qual è il componente di avvio dell'applicazione.

Il componente principale, creato anch'esso automaticamente, è formato dal

¹Maven è uno strumento usato nei progetti JAVA per l'automatizzazione del processo di build

file `appComponent.html`, che nel `body` contiene solamente un rimando al modulo di routing tramite il quale si visualizza un determinato componente in base all'URL richiesto.

Nel modulo di routing chiamato `appRouting.module.ts`, si andrà a definire un array di oggetti `Route` nel quale si avrà per ogni oggetto la corrispondenza tra un `path` e un componente da visualizzare. Si definisce anche un `path` generico il quale porterà alla visualizzazione sempre dello stesso componente (utile in caso di `path` sbagliato o inconsistente).

Ad esempio nella console utente, l'array definito per il routing è il seguente:

```
const routes: Routes = [
  {path:"documents", component:DocumentsComponent},
  {path:"documents/:id_file", component:DocumentComponent},
  {path:"**", redirectTo:"documents"}
];
```

il quale visualizzerà il componente "Document" nel caso in cui il `path` base terminerà con la stringa "document" più l'id di un file esistente. Nel caso in cui invece il `path` base terminerà con la stringa "documents" o in qualsiasi altro modo, sarà visualizzato il componente "Documents".

In `appModule` sono indicati tutti i componenti, pipe e moduli esterni utilizzati, oltre all'informazione su quale sia il componente per il bootstrap.

3.1.8 Resizing

Prerogativa importante delle Web Application è che esse si adattino correttamente al dispositivo sul quale vengono eseguite. Gli elementi dell'interfaccia devono essere disposti correttamente nonostante il possibile cambio di risoluzione della pagina web.

L'applicazione infatti, essendo una Web Application, può essere aperta non solo su dispositivi come laptop o computer fissi, ma anche su dispositivi mobili di dimensioni e risoluzioni minori. Inoltre anche nel caso dello stesso laptop, un utente può ridimensionare la finestra del browser richiedendo implicitamente al contenuto dell'applicazione di adattarsi alla nuova dimensione della finestra.

Oltre alla possibilità data all'utente di ridimensionare manualmente la grandezza della finestra, il browser permette anche di visualizzare l'applicazione come se fosse eseguita su dispositivi mobili in modo da far vedere come essa si adatti alla nuova risoluzione. Si verifica frequentemente che molti elementi

non vengono correttamente visualizzati e per questo motivo bisogna intervenire sul codice per evitare queste problematiche.

Nelle due console, queste criticità risolvibili tramite la modifica dello style degli elementi HTML, riguardavano la visualizzazione delle informazioni nella barra superiore. Ad esempio nel caso della console utente, i problemi di risoluzione avrebbero portato ad avere la barra di ricerca solo parzialmente visibile (nascondendo così il pulsante di ricerca), mentre nella console amministrativa si avrebbe avuto la visualizzazione incompleta dell'informazione sulla pagina corrente.

La soluzione a queste problematiche è stata trovata nella definizione di selettori² nei file CSS: se la risoluzione della finestra del browser sarà minore di una certa soglia, si andrà a nascondere l'elemento che utilizza quello specifico selettore. Di seguito (figure 3.1 e 3.2) si riproduce un caso di cambio risoluzione nella console amministrativa.

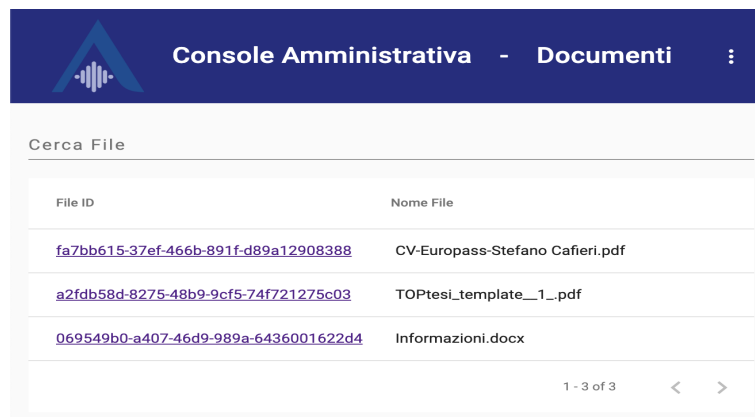


Figura 3.1. Finestra estesa del browser

²Nel foglio di stile CSS, un selettore è un identificatore di un insieme di proprietà che vengono valorizzate a seconda del comportamento che si vuole dare all'elemento HTML che lo utilizza

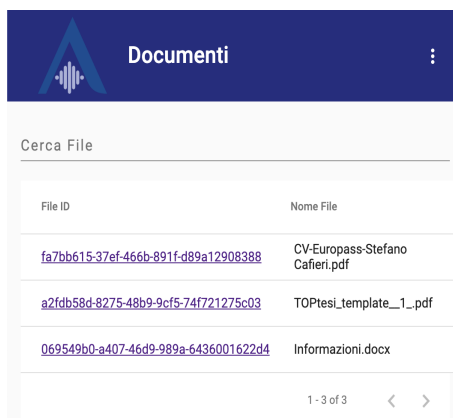


Figura 3.2. Finestra ridotta del browser

Si può notare come la barra superiore della stessa console cambi aspetto in base alla dimensione della finestra del browser.

Nel primo caso (figura 3.1), con una risoluzione orizzontale maggiore di 1000 pixel la barra mostrerà l'intera informazione sulla console amministrativa, cioè il logo dell'azienda con a fianco la scritta "Console Amministrativa" più l'indicazione su quale pagina si sta visualizzando.

Nel secondo caso (figura 3.1), invece, la barra sarà compressa e verrà eliminata l'informazione meno utile in quel momento; sarà perciò visualizzato solo il logo con a fianco l'indicazione sulla pagina corrente visualizzata.

Questo meccanismo garantisce all'applicazione una grande dinamicità a livello grafico, requisito fondamentale per le Web Application moderne.

Concluse le considerazioni sugli elementi comuni alle due console, adesso se ne analizzano singolarmente gli aspetti. Si presenterà dapprima la console utente e solo successivamente la console amministrativa.

3.2 Console Utente

La console utente è stata concepita per un utilizzo abituale da parte di utenti non esperti nel campo informatico. Questa scelta influenza molto il design dell'interfaccia. In essa infatti vengono messi in risalto gli elementi e le operazioni principali, dando anche informazioni visive di aiuto su tutti gli elementi

presenti. Queste informazioni vengono date tramite la comparsa di tooltip, cioè dei piccoli box che appaiono su finestre o icone quando vi si passa sopra con il cursore.

Ogni utente, dopo la fase di login in cui inserisce le proprie credenziali e verifica che sia autorizzato ad accedere al sistema, visualizza una lista di documenti comprendente tutti quei documenti di cui ha il diritto di lettura.

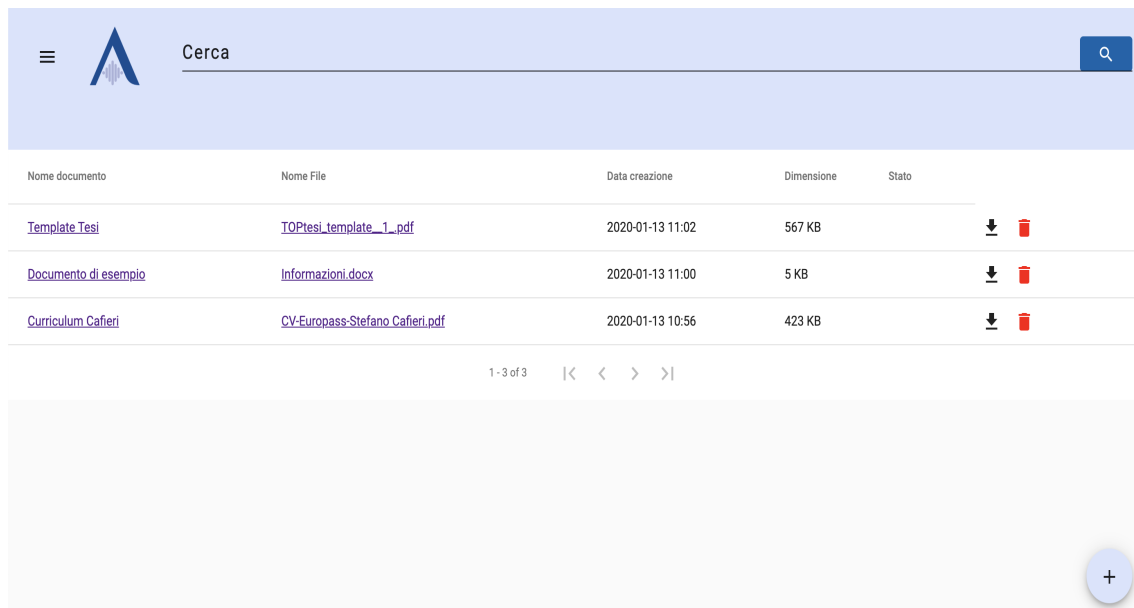


Figura 3.3. Pagina iniziale della console utente

La seguente immagine 3.3 mostra la view iniziale della console dopo che si è fatto il login. Come detto precedentemente, la pagina è molto semplice e lineare perché non fa altro che mostrare una lista di documenti, una barra di ricerca e un'eventuale menù di ricerca.

Ogni documento è associato ad alcune informazioni che lo contraddistinguono e a dei pulsanti per le operazioni di download e di cancellazione.

Con il meccanismo della paginazione, in una pagina possono essere elencati fino ad un massimo di dieci documenti. Questo meccanismo permette di navigare tra le varie view contenenti i documenti raggruppati dieci per volta.

Le immagini che seguono mostrano il menù di ricerca presente nella homepage.

L'immagine 3.4 mostra il menù di ricerca semplice. Esso permette la ricerca

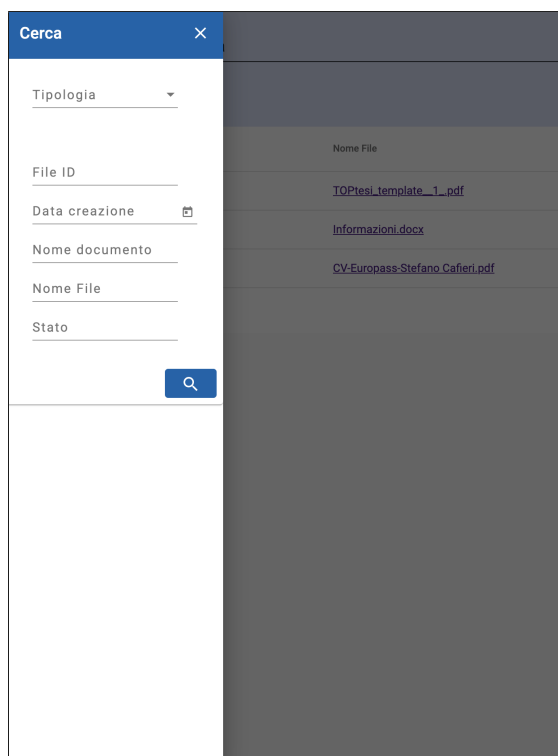


Figura 3.4. Menù di ricerca

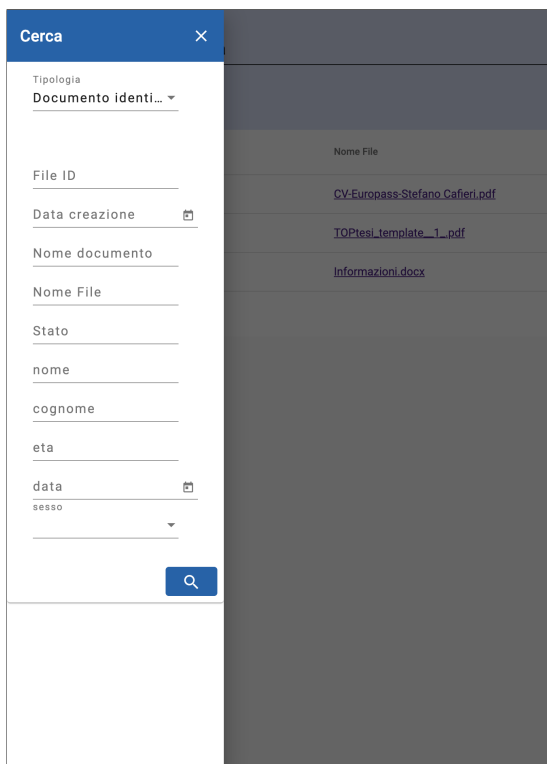


Figura 3.5. Menù di ricerca esteso per tipologia

dei campi principali di un documento, anche più di uno alla volta, come l'id del file, la data di creazione, il nome del documento, il nome del file e lo stato (ancora non disponibile). Una volta effettuata la ricerca, il menù si chiuderà e verranno visualizzati i documenti paginati e filtrati in base alla ricerca.

L'immagine 3.5 mostra il menù esteso in base alla tipologia selezionata. La selezione dal menu a tendina di una tipologia non solo consente di ricercare tutti i documenti tipizzati in quel modo, ma porta anche all'inserimento nel form di campi di ricerca appartenenti a quella tipologia, consentendo quindi di ricercare un campo specifico di una tipologia tra tutti quei documenti tipizzati in quel modo.

Nell'immagine, ad esempio, selezionando la tipologia "Documento d'identità" vengono fuori i suoi campi; se essi vengono lasciati vuoti la ricerca produrrà come risultato tutti i documenti di quella tipologia.

Per quanto riguarda la selezione di una tipologia, nell'implementazione finale è stata fatta una scelta diversa rispetto all'idea iniziale. Inizialmente infatti

la sola selezione di una tipologia portava al caricamento di tutti i documenti tipizzati in quel modo, oltre che all'aggiunta dei campi di ricerca. Nella soluzione finale invece vengono aggiunti solo i campi di ricerca della tipologia senza effettuare direttamente il caricamento dei documenti di quel tipo; questa scelta è stata fatta per non gravare sul server facendo delle ripetute chiamate al suo servizio. Se si vuole chiamare il servizio per ottenere tutti i documenti tipizzati nel modo selezionato è necessario solamente premere il tasto di ricerca, chiamando così il servizio solo quando effettivamente è richiesto dall'utente.

La creazione di un nuovo documento avviene tramite il bottone posto in fondo all'homepage. Questa operazione porta alla visualizzazione (figura 3.6) di un nuovo documento segnato come "`__new`", nel quale vi sono i due campi "Nome documento" e "Proprietario" e tutti gli altri campi riguardanti le informazioni sui gruppi abilitati alla lettura e/o scrittura.

Qui è possibile opzionalmente aggiungere nuovi campi o selezionare una tipologia; è invece obbligatorio caricare il file che conterrà il documento. Nel caso in cui non si fosse caricato alcun file, l'applicazione vieta di salvare il documento mostrando un popup di avviso.

Una volta caricato il file si potrà salvare il documento; esso avrà un numero di versione pari a zero e conterrà una serie di campi aggiuntivi automaticamente inseriti dal sistema.

The screenshot shows a web interface for creating a new document. At the top, there is a header bar with a back arrow, the text "no title", and a "Versione" dropdown menu. Below the header, the document name is displayed as ".__new - 0 B". The main form area contains several fields:

- Proprietario:** no owner
- Gruppi abilitati alla lettura:** admin
- Nome documento:** no title
- Gruppi abilitati alla scrittura del file:** admin
- Gruppi abilitati alla scrittura dei metadati:** admin

At the bottom left, there is a blue circular button with a white plus sign. At the bottom right, there are icons for email and a search icon.

Figura 3.6. Creazione di un nuovo documento

Non appena il documento viene salvato, verrà mostrato un documento simile a quello nella figura 3.7.

Si può notare la valorizzazione del campo versione (indicata anche come l'ultima), il calcolo dell'id-file e la visualizzazione della dimensione del file già compresso dal sistema. Inoltre si possono visualizzare i meta-dati del documento come la data di creazione, la lingua e l'ultima modifica, oltre ai tre metadati riguardanti i gruppi abilitati. I primi tre campi descritti sopra sono gestiti dal sistema in maniera automatica.

Nel caso di documento modificato più volte, è possibile quindi visualizzare l'elenco delle sue versioni precedenti e il loro contenuto. Non viene però resa possibile la modifica di queste vecchie versioni; infatti il sistema consente solamente di salvare una vecchia versione come ultima versione e modificare essa.

Nell'ultima versione del documento sarà sempre possibile cambiare o settare la tipologia, oltre che caricare nuovamente il file contenuto nel documento ed effettuare il download.

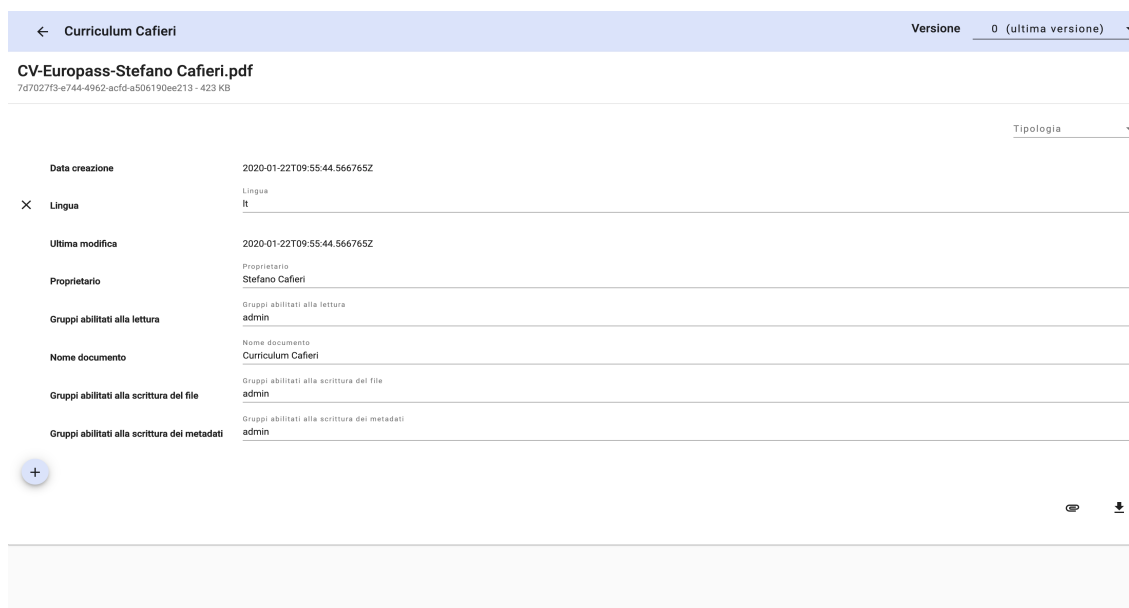


Figura 3.7. Nuovo documento salvato

Nella figura 3.8 si può vedere il dettaglio di un documento nel quale è appena stata selezionata una nuova tipologia.

← Curriculum Cafieri Versione ▾

CV-Europass-Stefano Cafieri.pdf
fa7bb615-37ef-466b-891f-d89a12908388 - 423 KB

Tipologia
Documento identità ▾

Data creazione: 2020-01-19T19:19:47.811519Z

✕ **Lingua**: Lingua
It

Ultima modifica: 2020-01-20T15:02:50.458295Z

Proprietario: Proprietario
Stefano Cafieri

Gruppi abilitati alla lettura: Gruppi abilitati alla lettura
admin

Nome documento: Nome documento
Curriculum Cafieri

Gruppi abilitati alla scrittura dei metadati: Gruppi abilitati alla scrittura dei metadati
admin

cognome:
required

data:
required

eta:
required

nome:
required

sesso:
required

🔍 📄 📄

Figura 3.8. Contenuto del documento con aggiunta di un tipo

Nello specifico, in questo caso si è scelto di tipizzare il documento con una tipologia tra quelle proposte (es: "Documento d'identità") il quale immette nel documento i propri campi obbligatori. L'obbligatorietà è evidenziata dal colore rosso del form, che può inoltre indicare la presenza di un errore nell'inserimento del valore del campo. Infatti per ogni campo della tipologia si specificano delle regole che l'utente deve rispettare quando lo compila. Una regola possibile è che ad esempio un campo deve contenere una stringa o una data, oppure un numero compreso tra determinati parametri o ancora può contenere anche una stringa che debba rispettare una regex (es: definizione di email o codice fiscale).

Alcuni campi della tipologia possono presentare dei valori di default, mentre altri possono essere settati in sola lettura, cioè dopo averli definiti non possono essere più modificati se non da un admin.

Adesso si descrivono a livello implementativo, le caratteristiche principali della console. I suoi elementi sono:

- componente "Documenti", contiene l'implementazione dell'homepage comprendente di menu di ricerca, lista di documenti e barra di ricerca.

- componente "Documento", contiene il dettaglio del documento. In esso è possibile modificare o aggiungere campi, modificare la tipologia e scegliere la versione da visualizzare.
- componente "addField", rappresenta la finestra che compare durante l'aggiunta di un nuovo campo all'interno di un documento. Essa è composta da un form che permette l'inserimento del nome del nuovo campo e del suo eventuale valore. È previsto un controllo sulla validità di questo input, che in caso di esito negativo non permette la creazione del campo.
- componente "errorDialog", rappresenta il popup mostrato quando vi è un errore importante, come ad esempio il fatto che il server non risponda o che si è effettuata una chiamata in maniera non corretta. Di questo errore ne viene fornita una descrizione, oltre alla rappresentazione dell'oggetto JSON corrispondente alla risposta d'errore del server. Si è scelto di creare un apposito componente, anche di dimensioni rilevanti, per evidenziare appunto all'utente la presenza di un problema, più o meno grave.
- componente "TypedInput", introdotto per avere la verifica sull'input dei campi. Infatti i campi appartenenti a una tipologia sono tipizzati e quindi vi è bisogno di controllare se questo input soddisfa i requisiti del campo. Si è scelto di creare un nuovo componente in quanto la logica dietro al processo di verifica è abbastanza complessa per poter essere inserita dentro il componente generico, cosa che avrebbe anche fatto crescere di troppo la lunghezza del codice nel singolo file. Questo componente, inserito dentro il componente "Documento", comunica con esso tramite i meccanismi di comunicazione descritti precedentemente. Passa le informazioni necessarie tramite il data binding e comunica i propri risultati, cioè se la validazione ha avuto successo o meno, tramite l'event binding che in questo caso lancia il metodo che aggiorna la versione temporanea del documento con il campo validato.
- servizio "backend-service", fornisce i veri e propri metodi che effettuano le chiamate verso le API del server. Utilizza il servizio di Angular denominato HttpClient per chiamare il backend del sistema documentale.
- servizio "documento-service", fornisce ai componenti l'interfaccia per effettuare le richieste al server, che poi avvengono tramite il servizio 'backend-service'. Presenta quindi le funzioni per get, put, post e delete, le quali sono gestite tramite il meccanismo di async/await ritornando

alla fine delle Promise.

Nel caso della GET su un singolo documento e sulla lista dei tipi, filtra la risposta del server passando ai componenti solamente il body della risposta; mentre nel caso della DELETE, passa invece l'intera risposta poiché il body risulta vuoto.

Nel salvataggio o aggiornamento di un documento ritorna l'intero documento aggiornato, mentre per la GET sull'intera lista di documenti ritorna un oggetto della classe Page. Questo oggetto conterrà al suo interno il body della risposta e una serie di informazioni ottenute direttamente dall'header della risposta: il numero di documenti che soddisfano la ricerca, il numero di documenti che in fase di ricerca si è deciso di scartare a priori e il numero massimo di documenti che si vogliono ottenere.

- file "util", contiene funzioni più volte usate dentro gli altri componenti. Quindi tutte quelle funzioni che saranno riutilizzate dallo stesso componente o anche dagli altri componenti, vengono scritte in questo file. In questo modo viene soddisfatta una delle linee guida imposte nello sviluppo del progetto, cioè la non riscrittura di codice identico a quello scritto precedentemente in qualche altro file. Si decide quindi di scriverlo una sola volta in modo che sia riutilizzabile ed adattabile a chiunque lo voglia utilizzare.

I componenti descritti sopra, come da definizione in Angular, conterranno 3 file: nel file CSS è contenuto lo stile applicabile all'HTML del componente, nel file HTML si ha il codice che rappresenta la struttura grafica della pagina, mentre nel file Typescript si hanno le funzioni utilizzate nel file HTML che offrono, tra le altre cose, funzionalità per interfacciarsi con il server passando dai servizi descritti sopra.

3.3 Console Amministrativa

La console amministrativa è nata a supporto della console utente per garantire la completa gestione dei documenti da parte di un amministratore, sia nei normali casi di manutenzione sia nei casi di criticità.

È quindi utilizzabile solo dagli amministratori e non dai comuni utenti.

L'importanza di realizzare una console amministrativa sta nel fatto che gli amministratori devono avere la possibilità di agire su qualunque documento,

leggere tutti i meta-dati non visibili a un normale utente e creare nuove tipologie non realizzabili dall'utente. Le tipologie, applicate poi nella console utente, non sono altro che è un insieme di campi definiti da un amministratore, per ognuno dei quali si inseriscono delle regole di valorizzazione.

Il comportamento degli utenti può non sempre essere corretto. Vi è quindi la necessità di gestire eventuali anomalie presentate da operazioni non concesse oppure modifiche non valide. Per questo motivo all'amministratore viene data la possibilità di poter visualizzare un intero documento comprensivo di tutti i suoi metadati.

Nonostante alcuni campi siano visibili all'utente, la loro modifica gli è vietata, a differenza dell'amministratore che invece può variarne il contenuto. Un esempio è rappresentato dai vari campi impostati automaticamente dal sistema. Queste modifiche da parte dell'amministratore sono possibili perchè a livello grafico il documento è presentato all'interno di un editor della stessa Web Application. Si ha un oggetto JSON in cui le chiavi e i valori possono essere liberamente modificati.

Quindi la figura dell'amministratore è intesa come una figura in grado di compiere consapevolmente e in maniera sicura delle operazioni poichè si ritiene che egli sia un esperto nel funzionamento del sistema.

Per tutti questi motivi la Web Application è realizzata in maniera totalmente neutrale ed uguale per qualsiasi cliente.

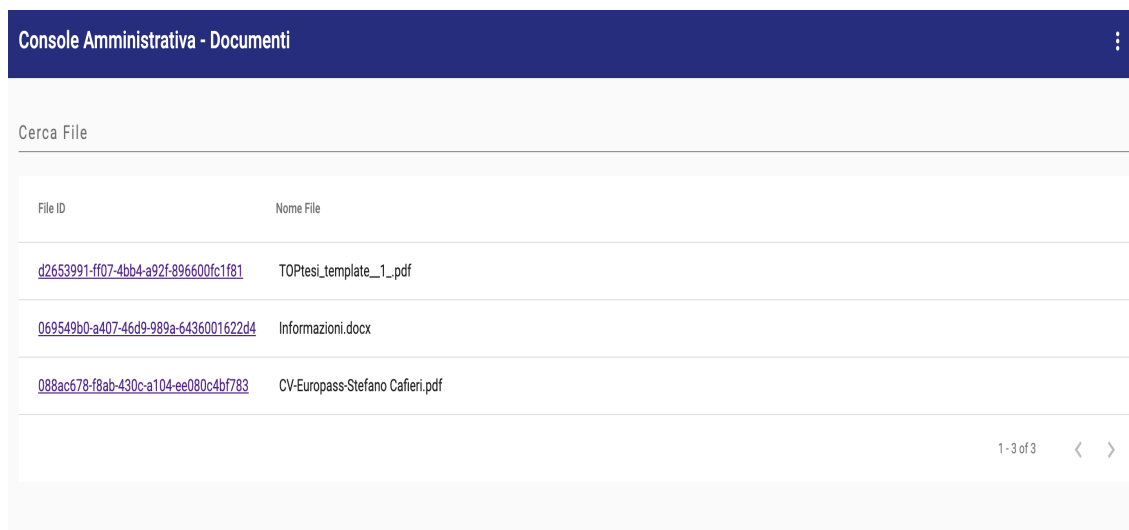
Unica novità stilistica presente in questa console rispetto a quella degli utenti è la presenza di un menù di navigazione che consente di passare rapidamente alla pagina dei documenti, a quella dei tipi o a quella delle statistiche.

Ricapitolando, le operazioni che gli sono consentite sono:

- visualizzazione e modifica di qualsiasi versione del documento e di tutti i suoi meta-dati;
- visualizzazione dell'intera lista di documenti con possibile ricerca su id del file;
- visualizzazione dell'intera lista dei tipi con la possibilità di ricercarli per nome;
- creazione e cancellazione di un nuovo tipo;
- visualizzazione e modifica di uno specifico tipo;

- visualizzazione delle statistiche del sistema documentale.

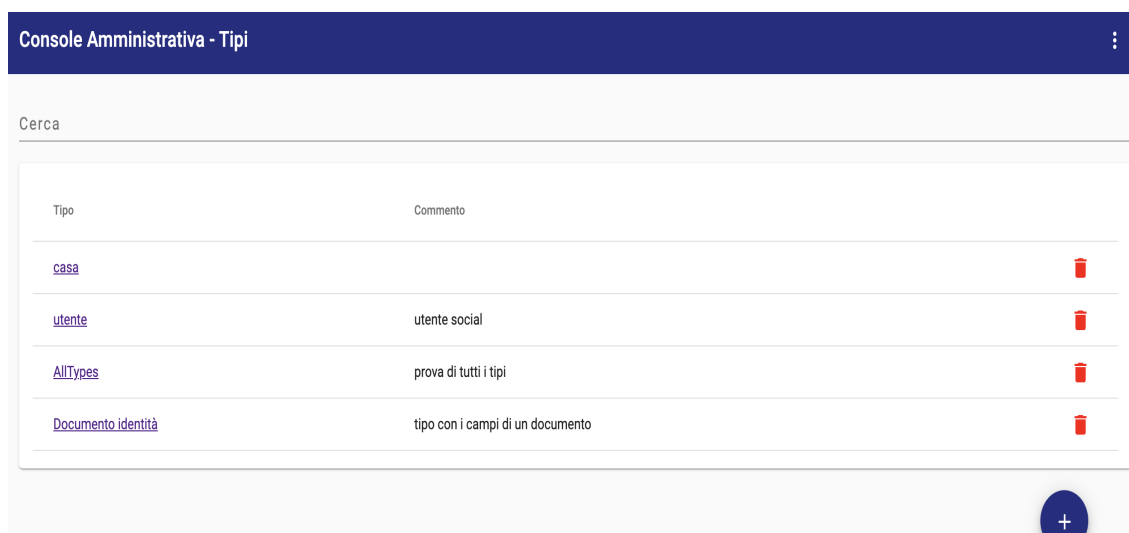
Di seguito saranno illustrate alcune immagini principali della Web Application amministrativa.



The screenshot shows the 'Console Amministrativa - Documenti' interface. It features a search bar labeled 'Cerca File' and a table with two columns: 'File ID' and 'Nome File'. The table contains three rows of document information. At the bottom right of the table, there is a pagination indicator '1 - 3 of 3' and navigation arrows.

File ID	Nome File
d2653991-ff07-4bb4-a92f-896600fc1f81	TOPtesi_template_1_.pdf
069549b0-a407-46d9-989a-6436001622d4	Informazioni.docx
088ac678-f8ab-430c-a104-ee080c4b7783	CV-Europass-Stefano Cafieri.pdf

Figura 3.9. Lista dei documenti



The screenshot shows the 'Console Amministrativa - Tipi' interface. It features a search bar labeled 'Cerca' and a table with two columns: 'Tipo' and 'Commento'. The table contains four rows of document type information. Each row has a red trash icon on the right side. At the bottom right of the interface, there is a blue circular button with a white plus sign.

Tipo	Commento
casa	
utente	utente social
AllTypes	prova di tutti i tipi
Documento identità	tipo con i campi di un documento

Figura 3.10. Lista dei tipi

Nella figura 3.9 è mostrata la pagina principale della console amministrativa. È visualizzata semplicemente la lista dei documenti presenti, insieme ai relativi id-file e nome file, oltre ad una barra di ricerca per id-file. Anche qui è presente il meccanismo della paginazione per raggruppare e mostrare al massimo 10 documenti per pagina.

Invece nella figura 3.10 è presente la lista dei tipi. Ognuno di essi può essere accompagnato dal commento che ne è stato inserito in fase di creazione.

Per i tipi, a differenza dei documenti, c'è la possibilità di essere creati e cancellati direttamente da questa console, poichè essi sono gestiti completamente e solamente dall'amministratore.

```

Console Amministrativa - 088ac678-f8ab-430c-a104-ee080c4bf783
Versione
0
# = {
3  "id_file": "088ac678-f8ab-430c-a104-ee080c4bf783",
4  "title": "Curriculum Cafieri",
5  "name": "Stefano Cafieri",
6  "readgroups": [
7    "admin"
8  ],
9  "writegroups": [
10   "admin"
11 ],
12 "version": 0,
13 "_size": 433107,
14 "filename": "CV-Europass-Stefano Cafieri.pdf",
15 "md5": "e6f048dd2e5d1891fbc52b98de4f23d1",
16 "created_at": "2019-01-13T09:56:18.321586Z",
17 "_language": "en-gb",
18 "_access_permission_extract_for_accessibility": "true",
19 "_access_permission_extract_content": "true",
20 "_access_permission_fill_in_form": "true",
21 "_access_permission_can_modify": "true",
22 "_meta_save_date": "2019-01-09T22:39:56Z",
23 "_keywords": "Europass, CV, Cedefop",
24 "_meta_creation_date": "2019-01-09T22:39:56Z",
25 "_last_save_date": "2019-01-09T22:39:56Z",
26 "_pdf_docinfo_producer": "LibreOffice 4.0; modified using iText® 5.5.11 ©2000-2017 iText Group NV (AGPL-version)",
27 "_xmp_creator_tool": "Writer",
28 "_subject": "Stefano Cafieri Europass CV",
29 "_cp_subject": "Stefano Cafieri Europass CV",
30 "_pdf_pdf_version": "1.4",
31 "_dc_format": "application/pdf; version=1.4",
32 "_xmpTp_nPages": "1",
33 "_last_modified": "2019-01-09T22:39:56Z",
34 "_access_permission_can_print": "true",
35 "_dcterns_created": "2019-01-09T22:39:56Z",
36 "_pdf_docinfo_created": "2019-01-09T22:39:56Z",
37 "_modified": "2019-01-09T22:39:56Z",
38 "_content_type": "application/pdf",
39 "_creation_date": "2019-01-09T22:39:56Z",
40 "_access_permission_modify_annotations": "true",
41 "_producer": "LibreOffice 4.0; modified using iText® 5.5.11 ©2000-2017 iText Group NV (AGPL-version)",
42 "_pdf_docinfo_title": "CV-Europass-20190109-Cafieri-ST.pdf",
43 "_access_permission_can_print_degraded": "true",
44 "_dcterns_modified": "2019-01-09T22:39:56Z",
45 "_dc_title": "CV-Europass-20190109-Cafieri-ST.pdf",
46 "_date": "2019-01-09T22:39:56Z",
47 "_pdf_docinfo_modified": "2019-01-09T22:39:56Z",
48 "_pdf_docinfo_keywords": "Europass, CV, Cedefop",
49 "_access_permission_assemble_document": "true",
50 }

```

Figura 3.11. Dettaglio di un documento

L'immagine 3.11 rappresenta il dettaglio di un documento. Appare subito evidente che non vi è più la gestione dei singoli campi come nella console utente. Qui l'intero documento appare all'interno di un editor, sotto forma di oggetto JSON. Sono visibili e liberamente modificabili tutti i campi, nonché metadati, del documento. La maggior parte di essi sono inseriti automaticamente dal sistema e quindi raramente vengono modificati (identificabili dall'underscore che precede i nomi dei campi). I campi che più spesso possono essere modificati dall'amministratore sono

quelli inseriti dall'utente oppure quelli riguardanti i diritti sul documento; con questi ultimi ci si riferisce soprattutto a "readgroups" e "writegroups", cioè array che indicano il gruppo di utenti che hanno i permessi di lettura e scrittura sul documento.

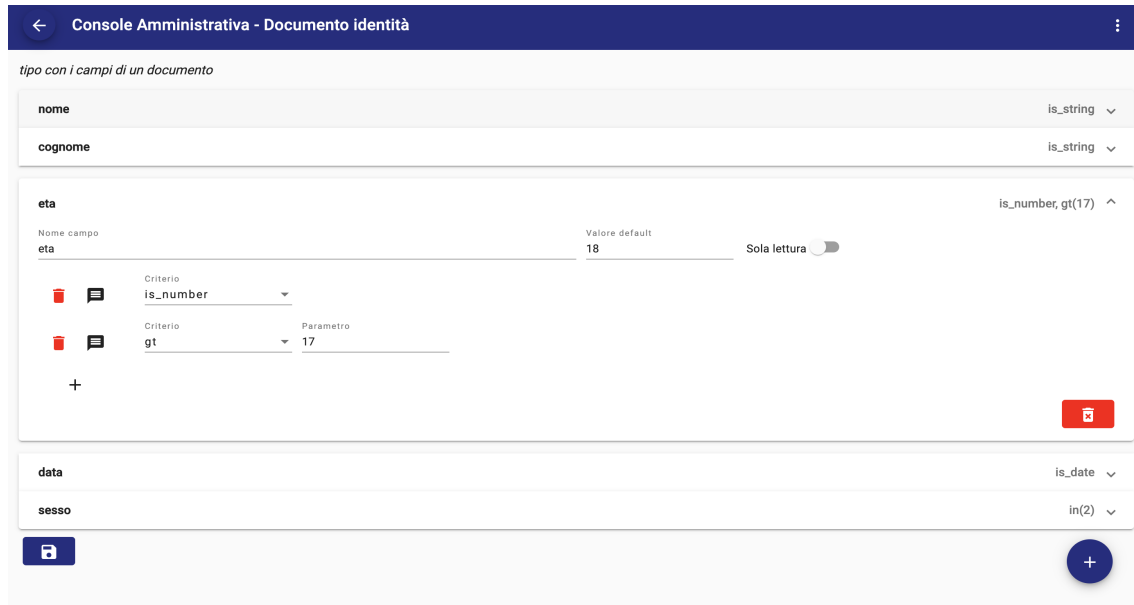


Figura 3.12. Visualizzazione dettaglio di una tipologia

L'immagine 3.12 mostra la parte più interessante e complessa della console amministrativa. Essa raffigura la gestione di una tipologia, in questo caso valorizzata già con diversi campi e nella quale è possibile inserire ulteriori campi.

La creazione di una nuova tipologia richiede solo un nome ed un eventuale commento. Un esempio di come è strutturata una tipologia è il seguente:

```
{
  type_id: "Documento identità",
  attributes: [
    {
      name: "nome",
      validation: [
        {
          f: "is_string",
          p: "",
          m: ""
        }
      ]
    }
  ]
}
```

```
        }
      ],
      default_value: "",
      read_only: false
    }
  ]
}
```

Tramite questo tipo appena visto, chiamato "Documento identità", possiamo vedere da cosa è composto un generico tipo:

- un "type_id" indicante il nome del tipo;
- un array chiamato "attributes" che contiene tutti i campi definiti all'interno del tipo. Ogni campo a sua volta contiene un oggetto JSON composto da:
 - nome del campo;
 - array di validazione, composto dai criteri che il valore del campo deve rispettare:
 1. f: indica il field, cioè un criterio vero e proprio tra quelli definiti in un file javascript (es: essere una stringa è definito con 'is_string', una data con 'is_date', una lista di valori con 'in', una email con 'is_email' e così via);
 2. p: indica un eventuale parametro per il criterio (es: il criterio 'eq' deve avere un parametro con il quale confrontare il valore);
 3. m: indica un messaggio (in caso di sua assenza compare il messaggio di default) che indica all'utente la non validità nella valorizzazione del campo;
 - valore di default opzionale;
 - booleano "read_only" indicante se il valore sarà o meno presente in sola lettura.

Nella console utente si avrà quindi che il valore di un campo appartenente ad una determinata tipologia dovrà seguire le regole definite. Sarà impedita l'immissione di caratteri se ad esempio un campo deve essere un numero, si aprirà un calendario nel caso in cui il campo richiesto sia una data e vi sarà un menù a tendina nei campi in cui è possibile scegliere solo determinati valori definiti dall'amministratore (es: UOMO/DONNA).

Nello sviluppo di questa parte della Web Application si è preferito avere maggiore complessità grafica per permettere all'amministratore una gestione migliore delle tipologie, vista la struttura complessa rispetto al resto della console.

Un requisito necessario per la gestione delle tipologie riguarda la possibilità di poter ordinare i campi di ogni tipologia in base a come si vuole che essi vengano visualizzati all'utente. In questa soluzione è possibile effettuare l'operazione tramite il trascinarsi verticale dei pannelli contenenti i nomi dei campi.

In questi pannelli, accanto al nome del campo, è richiesto che ne venga visualizzata un'anteprima dei criteri applicati.

La selezione di un singolo campo porta all'apertura di un pannello sottostante, come si vede sempre dall'immagine 3.12, in cui è possibile cancellare l'intero campo, aggiungere e/o modificare criteri e definire le altre eventuali opzioni.



Console Amministrativa - Statistiche	
Spazio occupato dai dati	8151040
Spazio occupato dai metadati	389120
Numero documenti	2793
Numero documenti cancellati	1315

Figura 3.13. Visualizzazione statistiche della console

La figura 3.13 mostra le statistiche principali del sistema, inserite per avere un resoconto dello storico riguardante il numero di documenti totali e dello spazio occupato da dati e metadati.

Il numero così elevato di documenti è giustificato dal fatto che ogni versione del documento viene salvata come se fosse un documento vero e proprio a se stante, incrementando quindi il contatore dei documenti totali. La discrepanza così ampia tra il numero di documenti presenti e quelli cancellati è dovuta al fatto che ad ogni cancellazione, oltre ad aumentare il contatore delle cancellazioni, aumenta anche quello dei documenti presenti, poichè viene creata

una nuova versione del documento cancellato con un campo 'deleted' settato a true.

L'implementazione di questa console ricalca molti aspetti già presentati nella console utente.

Infatti qui, a parte la gestione dei tipi che non compare nella console utente, sono presenti molti componenti simili.

Gli elementi principali di questa soluzione sono:

- componente "Tipi", contiene l'implementazione della pagina comprendente la lista dei tipi e la barra di ricerca per nome;
- componente "Tipo", contiene il dettaglio di un tipo con l'elenco di tutti i suoi campi e i relativi criteri. Sono possibili numerose operazioni come la creazione e la cancellazione dei campi, l'inserimento e la cancellazione dei criteri e infine il riordinamento dei campi.
- componente "Statistiche", contiene l'implementazione per la visualizzazione delle statistiche del sistema, mostrate all'interno di una tabella.
- componenti "Documenti" e "Documento", simili ai componenti presenti nella console utente. In 'Documenti' le differenze sono rappresentate dall'assenza del menu di ricerca e dalla presenza della barra di ricerca per id-file. In 'Documento' invece è presente l'implementazione dell'editor nel quale viene rappresentato il dettaglio del documento.
- componenti "aggiungi" e "aggiungiLista", rappresentano le finestre che compaiono rispettivamente al momento dell'aggiunta di un nuovo campo all'interno di una tipologia e al momento dell'aggiunta di un criterio 'in' all'interno di un campo. In quest'ultimo caso, cioè nel componente 'aggiungiLista', la finestra deve permettere di inserire un numero variabile di coppie chiave valore, le cui chiavi saranno mostrate nel menu a tendina del campo visualizzato nella console utente, e che a sua volta verrà valorizzato con il rispettivo valore.
- componente "conferma", rappresenta la finestra per confermare o meno l'operazione appena effettuata.
- servizio "tipo-service", fornisce i metodi per effettuare le chiamate alle API del server. Le chiamate avvengono tramite il servizio di Angular chiamato HttpClient.
- altri servizi molto simili a quelli della console utente.

3.4 Testing

La fase di testing è una fase fondamentale nel ciclo di vita di un progetto poichè permette di capire l'efficacia e la correttezza del codice scritto, cioè se esso risponde correttamente ai requisiti posti inizialmente per la realizzazione della soluzione.

Rappresenta la zona di confine tra lo sviluppo e la messa in produzione di un prodotto. Infatti un progetto può ritenersi completo e funzionante (a meno di scoperte future di bug) solamente quando si ha un esito positivo per tutti i test e per tutte le sollecitazioni degli sviluppatori o, più specificatamente, dei software tester.

Si possono definire due tipi di test: automatici e manuali.

Il *testing automatico* è un approccio molto interessante poichè permette di rieseguire liberamente i test tutte le volte che si vuole, senza il vincolo di effettuare su di essi delle modifiche.

Questi test consistono in brevi programmi scritti dallo sviluppatore, il cui scopo è verificare la correttezza delle risposte alle chiamate effettuate verso il software del prodotto. Si valuta se queste risposte corrispondono alle condizioni attese dallo sviluppatore, proprio perchè chi scrive i test conosce l'ambiente in cui questi vengono eseguiti. Dopo aver scoperto e corretto eventuali bug, i test vengono nuovamente rilanciati per verificare la correttezza delle modifiche.

I test devono essere legati a parti di codice del software che non prevedono interazione con gli utenti.

Test di questo tipo possono ad esempio verificare la risposta di un server ad una possibile richiesta di un client oppure possono verificare la reazione di un server quando esso viene fortemente stressato a causa dell'immissione di una grande mole di dati, valutandone così la robustezza.

Il *testing manuale* è usato invece nei casi in cui si deve verificare il comportamento del software in risposta all'interazione umana. È buona prassi che questi test vengano effettuati anche da persone che non siano gli sviluppatori dell'applicazione. Vi è questa prassi perchè devono essere valutate tutte le eventuali criticità che lo sviluppatore non ha considerato o che ha valutato di poco conto.

Il testing sul sistema documentale ha previsto l'uso di test automatici e test manuali.

I test automatici sono stati utilizzati per verificare il corretto funzionamento delle API messe a disposizione dal server, mentre i test manuali sono stati utilizzati per constatare che le Web Application si comportino nel modo corretto.

I test automatici effettuati hanno avuto la prerogativa di dover rispettare 3 semplici vincoli, i quali ne garantiscono l'autoconsistenza:

- Indipendenza, cioè ogni test non deve dipendere da altri test
- Riesequività, cioè alla fine di ogni test devono essere ripristinate le condizioni di partenza per poter rieseguire nuovamente il test nelle stesse condizioni
- Riproducibilità, cioè il test consegue gli stessi risultati a prescindere da chi lo esegue

Questi tipi di test sono stati svolti tramite l'uso di un tool di grande successo chiamato Postman. Questo strumento permette di creare test, facilmente riesequivibili e raggruppabili in collezioni, sui tanti metodi HTTP.

Principalmente sono stati testati i metodi GET, POST, PUT e DELETE.

Il tool, nel rispetto dei vincoli illustrati precedentemente, permette di creare degli script pre-test tramite i quali si arricchisce l'ambiente su cui vengono effettuati i test. Ad esempio si aggiungono dei documenti da testare, i quali però verranno successivamente cancellati per ripristinare l'ambiente iniziale, in modo da non violare il vincolo sulla riesequivibilità dei test.

Si mostrano gli step per eseguire uno specifico test su un documento. In particolare si verifica che la GET funzioni correttamente:

1. Nello script pre-test si crea un nuovo documento tramite una POST. Nella struttura contenente il messaggio che sarà inviato, saranno compilati i campi riguardanti l'header e il body del messaggio. Nello specifico, l'header avrà l'informazione sul tipo del contenuto del messaggio, che in questo caso sarà un oggetto json. In questo oggetto, che sarà presente nel body del messaggio, verranno immessi e valorizzati i campi obbligatori per la creazione di un documento. L'unico campo obbligatorio è il campo 'content', che rappresenta il contenuto del file caricato nel documento.
2. Si eseguono i veri e propri test tramite una GET sul documento creato. Il primo test è effettuato sul codice di stato della risposta; esso

deve essere pari a 200 per avere la conferma dell'esistenza della risorsa rappresentante il documento appena creato. Il secondo test verifica che l'id dell'oggetto ritornato nel body della risposta sia uguale all'id che ci si aspetta, cioè l'id con il quale si è costruito l'URL per effettuare la chiamata. Se la condizione attesa è rispettata, allora il test è stato superato.

3. Nello stesso script per i test, si effettuano le operazioni post-test. In questo caso si ha la cancellazione, tramite DELETE, del documento creato inizialmente e sul quale sono stati fatti i test. Grazie a questa cancellazione finale, si riporta l'ambiente alla sua condizione iniziale.

Dopo aver mostrato l'esempio di uno specifico test, si elencano di seguito i test principali eseguiti sui documenti:

- GET sull'intera lista di documenti, verificando che il body della risposta sia un array;
- GET su una lista di documenti filtrata secondo: valori di campi specifici, numero massimo di risultati ottenibili, offset e ricerca fulltext;
- GET su un singolo documento, sia nel caso in cui esista sia nel caso in cui non esista;
- POST di un documento, sia in maniera corretta sia mancante di alcuni campi obbligatori;
- PUT su un documento esistente e uno non-esistente, fatta sia in maniera corretta sia in maniera errata;
- DELETE su un documento esistente e su uno non esistente;
- altri test minori

Per ognuno dei test elencati sopra, grazie alla combinazione di vari elementi da testare, sono stati sviluppati più test. Sono stati effettuati anche dei test per la gestione delle tipologie.

I test manuali sono stati fatti sulle Web Application. In questi tipi di test si deve cercare di replicare il più possibile l'imprevedibile

comportamento di un qualsiasi utente inesperto, verificando che l'applicazione continui a comportarsi in maniera corretta.

Per rendere i test più veritieri, essi devono essere fatti da più persone con target diversi.

Il punto di partenza di questi test è l'esecuzione completa di una procedura, come ad esempio la creazione, la modifica o la cancellazione di un documento, alla quale via via si vanno inserendo dei comportamenti non corretti per vedere se il sistema reagisce nella maniera attesa.

Si prenda come esempio la creazione di un documento. Innanzitutto si fa la procedura corretta di creazione e si verifica che essa funzioni correttamente. Dopo aver verificato ciò, ad esempio si può controllare se, come previsto, il sistema impedisca la creazione del documento a causa ad esempio della mancanza del file da salvare o di un campo obbligatorio non compilato, o ancora di un campo compilato non correttamente oppure a causa di altri comportamenti errati.

Oltre al verificare il corretto funzionamento delle varie procedure nel sistema, vi sono altri aspetti importanti da esaminare: [19]

- Usabilità, cioè se la navigazione è fluida ed intuitiva, se tutti gli elementi delle pagine vengono mostrati secondo i requisiti e se i testi sono chiari e corretti grammaticalmente;
- Performance, cioè se l'applicazione in condizioni di alto carico non ha comportamenti imprevisti che possano portare a crash che bloccano o addirittura chiudono il browser; in caso di verifica di alcuni di questi eventi, si deve verificare se l'applicazione riesce normalmente a riprendere il suo lavoro;
- Compatibilità, cioè se l'applicazione è compatibile con tutti i browser di interesse e se viene mostrata regolarmente in dispositivi diversi come PC o smartphone.

3.5 Messa in produzione

Un ulteriore fase del ciclo di vita del sistema documentale è rappresentato dalla sua messa in produzione, detta anche deployment.

Infatti dopo aver verificato l'effettivo rispetto di tutti i requisiti iniziali e la correttezza di tutte le sue funzionalità, si procede alla pubblicazione del

software nella sua versione stabile, pronto per essere consegnato al cliente o a chiunque ne abbia il diritto di utilizzo.

Questa fase, secondo accordi commerciali, comprende l'installazione e la configurazione nell'ambiente operativo del cliente, in modo tale da essere già pronto all'utilizzo dopo che ne è stato spiegato dettagliatamente il funzionamento.

Il cliente, dopo aver ricevuto il software con allegata documentazione, potrà verificare autonomamente tramite una fase di testing se esso rispecchia tutti i requisiti comunicati inizialmente.

Sempre secondo determinati accordi commerciali, il cliente avrà un determinato periodo di tempo per richiedere la correzione di eventuali nuovi bug scoperti o anche per richiedere delle piccole migliorie in base alle proprie esigenze. Al termine di questo periodo, dopo aver espletato le ultime richieste del cliente, parte la fase di manutenzione del software che consiste nell'apportare, dove necessario, modifiche al software. Questa fase è abbastanza onerosa e viene trattata separatamente rispetto alla realizzazione del software.

Si definiscono 3 tipi di manutenzione:

- manutenzione adattiva, quando si richiede di adattare in un nuovo ambiente operativo il prodotto già in possesso (a causa di modifiche hardware o software dell'ambiente);
- manutenzione correttiva, quando tramite l'emissione di patch si correggono i nuovi bug scoperti;
- manutenzione evolutiva, quando si aggiorna il software per far sì che offra nuove funzionalità o ne elimini alcune che non si sono rilevate utili.

Questi tipi di manutenzione possono essere richiesti in blocco oppure singolarmente. Nonostante tutto ciò rappresenti un costo aggiuntivo per il cliente, esso rappresenta uno step molto importante nella vita di un prodotto poiché lo mantiene sempre aggiornato sia dal punto di vista delle funzionalità sia dal punto di vista della sicurezza. Lo rende il più possibile libero da bug e sostenuto da un'assistenza tecnica fornita da chi ha pensato e realizzato il prodotto. [20]

Capitolo 4

Conclusioni e Sviluppi Futuri

La seguente tesi ha voluto evidenziare alcuni aspetti ritenuti importanti per il raggiungimento dello scopo prefissato dall'azienda AlmavivA, cioè quello di integrare, nel sistema documentale già presente, due Web Application e di testarne in maniera completa le funzionalità.

Si è cercato di sottolineare i vantaggi portati dallo sviluppo di queste applicazioni. A discapito di altre soluzioni con caratteristiche magari più performanti, ma che in questo contesto non erano necessarie, si è spiegato il motivo principale per cui è stata adottata questa soluzione, che garantisce versatilità, facilità di utilizzo ed economica manutenibilità.

Infatti il grande vantaggio di queste applicazioni è rappresentato dal fatto che esse possano essere eseguite in qualsiasi contesto. Ciò è possibile grazie alla sola presenza di un browser, elemento presente nella maggior parte dei dispositivi recenti; unica prerogativa è che esso sia compatibile per l'esecuzione di queste applicazioni. I browser maggiormente diffusi come Chrome, Safari, Firefox e molti altri sono pienamente compatibili con le Web Application.

Si è parlato anche del framework utilizzato, cioè Angular, evidenziando i suoi vantaggi e le caratteristiche che lo contraddistinguono. Nel mercato possono trovarsi soluzioni più performanti per lo sviluppo di queste applicazioni, ma l'esperienza maturata nell'ambiente di sviluppo dell'azienda e le funzionalità che si prestavano perfettamente allo scopo che si voleva raggiungere, hanno portato alla decisione di utilizzare questo framework. Quest'ultimo, inizialmente non è di semplice comprensione, ma con l'utilizzo giornaliero si rivela

uno strumento molto potente grazie anche all'utilizzo del linguaggio *Type-script* che permette, a scapito dei motivi che hanno reso famoso il linguaggio da cui deriva, una scrittura più efficiente del codice. Infatti tramite questo linguaggio risulta più semplice scovare degli errori che sarebbero stati individuati solo successivamente.

Le immagini delle console hanno cercato di mostrare nella miglior forma possibile la sequenza di operazioni che possono essere svolte sulle console, evidenziando le scelte fatte e come alcuni problemi siano stati affrontati.

Si è voluto sottolineare come queste console siano state implementate rispettando il requisito principale che prevedeva semplicità e facilità d'uso.

Un aspetto fondamentale che si è voluto porre in risalto è la fase di testing. Tramite questa fase si valida tutto il lavoro fatto fino a quel momento, valutando sia i tempi di risposta sia soprattutto la correttezza delle risposte. Per questo motivo la figura del software tester all'interno delle aziende assume sempre più importanza. È fondamentale rilasciare ad un cliente un prodotto che risponda correttamente ai requisiti che si sono posti, a discapito anche di un leggero ritardo, piuttosto che rilasciare un prodotto non perfettamente funzionante sul quale presto si dovrà intervenire per individuare e correggere i bug presenti.

Importanti sono anche le fasi successive di manutenzione e di ricerca. Un sistema funzionante ma con tecnologie poco recenti e/o funzionalità limitate, a causa del progresso tecnologico e dell'immissione di nuove funzionalità da parte dei competitor, ha la necessità di innovarsi e sviluppare nuove soluzioni per far sì che il cliente rimanga sempre soddisfatto del prodotto che possiede. È quindi fondamentale per l'azienda avere un prodotto aggiornato con nuove funzionalità, oltre che esente da bug, protetto da qualsiasi nuovo attacco che riguardi la sua sicurezza e che sia efficiente nelle sue vecchie funzionalità, rendendole possibilmente anche più performanti.

Per quanto riguarda le nuove funzionalità del sistema documentale e nello specifico delle Web Application, rispondendo alle richieste dei clienti, vengono valutate costantemente nuove implementazioni.

È stata ad esempio richiesta la valutazione dell'inserimento di un meccanismo di controllo dello stato dei documenti che permetta di monitorarli e far vedere lo stato nel quale si trovano, cioè se il documento per esempio è stato validato oppure si trova ad uno step precedente della sua fase di validazione. A livello front-end, l'inserimento di questo meccanismo dovrebbe dare la possibilità a chi ne ha il diritto di poter modificare lo stato del documento o poter effettuare delle modifiche, per così farne cambiare conseguentemente lo stato.

Oltre la valutazione della precedente richiesta, ancora in fase di discussione, un'evoluzione certa del sistema riguarda il voler garantire ad esso il pieno supporto alla multimedialità.

Il sistema documentale fino a questo momento è capace di memorizzare documenti di qualsiasi tipo purché siano di piccole dimensioni (documenti, immagini o brevi video), senza però offrirne il supporto per la visualizzazione o altre operazioni inerenti ad essi. I contenuti multimediali vengono trattati come tutti gli altri documenti per i quali si possono leggere e/o modificare i metadati, oltre che visualizzarne il contenuto dopo il download.

Un problema legato a questo tipo di contenuti è il loro caricamento nel sistema, poichè essendo un'operazione fatta su un file possibilmente molto grande, come appunto può essere un video, questa operazione risulta per nulla ottimizzata, onerosa e soprattutto bloccante.

I temi fondamentali identificati per lo sviluppo di questa soluzione, nello specifico per i video, sono: upload di video di grandi dimensioni, conversione in vari formati disponibili e streaming del contenuto sul browser.

L'idea è di aggiungere un sistema parallelo che si occupi delle operazioni preliminari sul contenuto del video, come la conversione in altri formati, garantendo però per esso un certo grado di affidabilità a seguito dei possibili guasti che possono colpirlo, oltre che un'alta reattività in caso di molteplici richieste. Per l'upload dei video si è pensato di usare il protocollo FTP.

Per il lato front-end vi deve essere la possibilità di visualizzare lo stato di avanzamento dei lavori sul video, oltre che una sua anteprima e il suo streaming.

La soluzione trovata consta nell'avere un dispatcher (un thread che legge le richieste in arrivo dalla rete e le gira a dei thread in quel momento non attivi) che dialoga con i vari convertitori e con un database, occupandosi quindi di fare da tramite tra gli oggetti esterni e il documentale. Quest'ultima affermazione permette di garantire il principio fondamentale sui si basa il sistema, cioè il principio di Single Responsibility, secondo il quale esso si occuperà di svolgere solo il suo compito originale, cioè quello di salvare e gestire i documenti.

Queste nuove funzionalità si ripercuoteranno direttamente sulle Web Application, le quali dovranno integrare queste nuove caratteristiche. Verrà quindi sviluppato:

- un nuovo sistema di caricamento dei file che preveda la visualizzazione dello stato di avanzamento;
- un'interfaccia che permetta di scegliere in quali formati si vuole salvare

il video e la relativa percentuale di conversione;

- un'interfaccia che permetta lo streaming del contenuto video direttamente sul browser e che porti alla possibilità di poter effettuare delle operazioni su di esso, come il cambio di risoluzione o di lingua (se disponibile).

Altri sviluppi futuri sul progetto che si potrebbero realizzare riguardano l'integrazione di questo sistema documentale con un altro sistema aziendale che permette di inviare notifiche agli utenti tramite ad esempio SMS, email, social network o altro. Ciò potrebbe risultare utile nei casi in cui per un documento vi è bisogno di notificare un qualcosa, come ad esempio una data di scadenza indicata nei metadati di un documento.

Bibliografia

- [1] https://www.almaviva.it/it_IT
- [2] <http://www.giampaolospaggiari.it/ingegneria-dei-sistemi-informatici.htm>
- [3] <https://italiancoders.it/introduzione-a-rest/>
- [4] https://it.wikipedia.org/wiki/Representational_State_Transfer
- [5] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, 2000
- [6] https://it.wikipedia.org/wiki/JavaScript_Object_Notation
- [7] https://it.wikipedia.org/wiki/Software_as_a_service
- [8] <https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app>
- [9] <https://www.HTML.it/pag/16727/web-application-e-application-server/>
- [10] <https://www.HTML.it/18/12/2018/progressive-web-app-vs-app-native-quali-scegliere/>
- [11] https://it.wikipedia.org/wiki/Progressive_Web_App
- [12] https://www.mrwebmaster.it/javascript/introduzione-angular_12716.HTML
- [13] https://www.mrwebmaster.it/img/guide/angular/lesson_04/applicazione-angular-lista-della-spesa-struttura-componenti.jpg

- [14] <http://webappfromzero.com/2018/05/20/binding-angular-2-come-funziona/>
- [15] https://www.mrwebmaster.it/javascript/pipe-angular_12753.HTML
- [16] <https://www.HTML.it/pag/60329/direttive-in-angular-2/>
- [17] https://www.mrwebmaster.it/javascript/servizi-dependency-injection-angular_12761.HTML
- [18] <https://www.HTML.it/pag/55620/introduzione-a-typescript/>
- [19] <https://www.guru99.com/web-application-testing.HTML>
- [20] <http://docenti.ing.unipi.it/a009435/issw/isw1516.pdf>
- [21] https://en.wikipedia.org/wiki/Systems_development_life_cycle