Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

CLOUD COMPUTING: fondamenti e implementazione in una Web App

Relatore: *Prof. Claudio Zandron*

Relazione della prova finale di:

Niccolò Balzarotti

Matricola 852003

Anno Accademico 2024-2025

Alla mia famiglia, che mi ha sostenuto e supportato economicamente per tutto il percorso, insegnandomi a non arrendermi mai davanti alle difficoltà.

Alla mia ragazza, Sofia, che ha sempre creduto in me supportandomi e sopportandomi nei momenti più difficili.

Ai miei compagni di corso, Matteo e Alaa, che hanno reso questo percorso più leggero e pieno di momenti spensierati.

Ai miei amici, che hanno saputo trasformare ogni problema e difficoltà in un'occasione per ridere e stare insieme.

A tutte le persone che mi hanno aiutato e spronato lungo il cammino, insegnandomi a dare sempre il massimo e a migliorarmi ogni giorno.

A me stesso, anche se non è andata come sperato.
È un percorso che porterò sempre con me, perché mi ha insegnato quanto io possa essere fragile e forte allo stesso tempo.
Ai miei sogni, che ora so di poter affrontare con consapevolezza e determinazione.

Introduzione

Con l'avvento di Internet e il suo rapido sviluppo, il cloud computing è diventato indispensabile sopratutto in contesti aziendali offrendo maggiore flessibilità e scalabilità rispetto alla tradizionale infrastruttura on-premise. L'obiettivo di questa tesi è quello di analizzare questa tecnologia partendo dalla sua nascita fino ad oggi, analizzandone gli aspetti tecnici e mostrando in modo pratico l'utilizzo di un servizio di cloud computing all'interno di una web app aziendale.

La relazione è strutturata come segue:

Nel capitolo 1 ci occuperemo di trattare la nascita del cloud computing e il suo sviluppo per come lo conosciamo oggi

Nel capitolo 2 ci concentreremo sugli aspetti che denotano questa tecnologia e le sue caratteristiche.

Nel capitolo 3 entreremo nel dettaglio analizzando l'architettura di questa tecnologia.

Nel capitolo 4 parleremo della trasformazione digitale e evidenzieremo vantaggi e svantaggi di questa tecnologia mostrandone criticità e punti di forza.

Nel capitolo 5 specificheremo i requisiti richiesti per lo sviluppo della web app e analizzeremo le tecnologie e gli strumenti utilizzati.

Nel capitolo 6 analizzeremo in modo più tecnico la web app sviluppata, mostrandone i punti fondamentali e una piccola demo.

Indice

Capitolo 1

Storia Del Cloud Computing

1.1 La nascita e l'evoluzione del Cloud Compunting

A porre le basi del cloud computing negli anni '60 è stato il professor John McCarthy, scienziato informatico noto anche per aver coniato il termine Intelligenza Artificiale. In un celebre discorso tenuto al MIT (Massachusetts Institute of Technology), John McCarthy ha introdotto l'idea di Time Sharing, ossia di un computer in grado di supportare la presenza simultanea di più utenti.

Sempre nel 1961 fu McCarthy a introdurre il concetto di Utility computing, un modello di fornitura di servizi on demand per i clienti con tariffe a consumo. Il cloud computing che conosciamo oggi però non nasce solo dalle sue idee, ma anche da quelle di Joseph Carl Robnett Lickelider, il primo a parlare di "*rete intergalattica di computer*".[1]

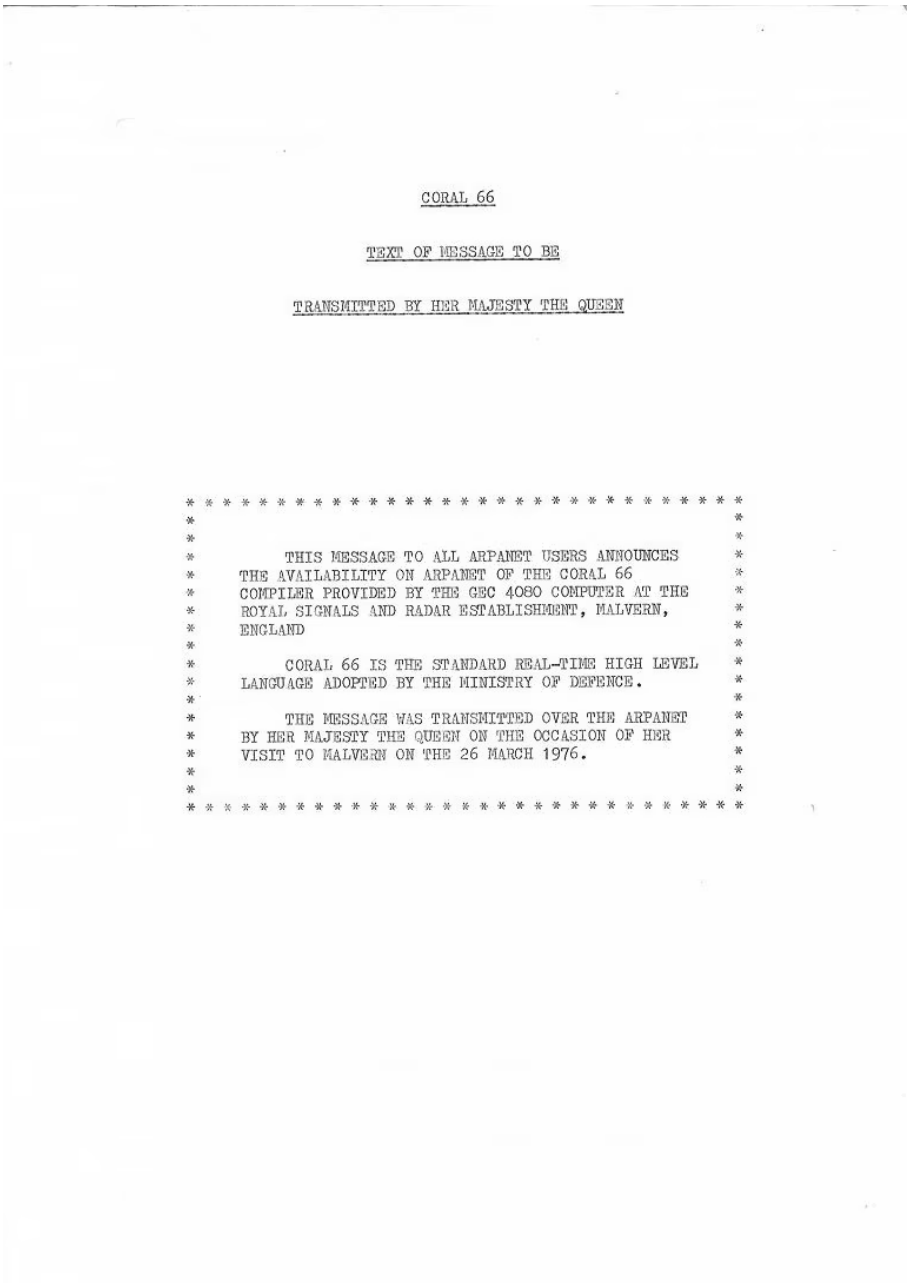
1.1.1 La "rete intergalattica di computer" di Lickelider

In molti fanno risalire la nascita del cloud computing proprio a Joseph Licklider che, nel 1969, lavorava allo sviluppo del famoso ARPANET (Advanced Research Projects Agency Network), una rete commissionata dal Dipartimento della Difesa degli Stati Uniti per scopi militari. Questa rete consentiva uno scambio di informazioni veloce, sicuro ed efficiente rendendolo sostanzialmente il network progenitore di internet. Nella visione di Licklider tutto il mondo poteva essere interconnesso e poteva accedere ai programmi da qualsiasi luogo e in qualsiasi momento portando così alla nascita del *grid computing*, il primo vero antenato del cloud.

L'unione di queste idee e progetti come il MAC (Multiple Access Computing) del 1965 e il sistema operativo time sharing CP-40/CMS messo a punto da IBM nel 1967, hanno fatto sì che il mondo si aprisse all'idea di un sistema informatico utilizzato da più persone contemporaneamente.[1]

1.2 La macchina virtuale di IBM

Le macchine virtuali di IBM negli anni ’70 CP-40/CMS non erano altro che software in grado di eseguire sistemi operativi e applicazioni. In pratica, ognuna di queste macchine possiede dei dispositivi che forniscono le stesse funzionalità dell’hardware fisico e offrono vantaggi in termini di portabilità, gestione e sicurezza, permettendo agli utenti di accedere a un hardware gestito esternamente. Gli anni ‘70 e ‘80 furono quelli della svolta; in particolare, nel 1976 il funzionamento dei progressi del networking fu mostrato alla regina Elisabetta II, che inviò un’e-mail proprio con ARPANET.[1]



CORAL 66

TEXT OF MESSAGE TO BE

TRANSMITTED BY HER MAJESTY THE QUEEN

THIS MESSAGE TO ALL ARPANET USERS ANNOUNCES THE AVAILABILITY ON ARPANET OF THE CORAL 66 COMPILER PROVIDED BY THE GEC 4080 COMPUTER AT THE ROYAL SIGNALS AND RADAR ESTABLISHMENT, MALVERN, ENGLAND

CORAL 66 IS THE STANDARD REAL-TIME HIGH LEVEL LANGUAGE ADOPTED BY THE MINISTRY OF DEFENCE.

THE MESSAGE WAS TRANSMITTED OVER THE ARPANET BY HER MAJESTY THE QUEEN ON THE OCCASION OF HER VISIT TO MALVERN ON THE 26 MARCH 1976.

Figura 1.1: Elisabetta II, la prima sovrana a inviare una e-mail con Arpanet [2]

1.3 Il lancio del World Wide Web

Le tecnologie fondamentali per il funzionamento del cloud si sono consolidate negli anni ’90, semplificate dal lancio del World Wide Web del 1991.

E’ proprio in questo decennio rivoluzionario per la tecnologia e contraddistinto dalla nascita dei primi e-commerce, dei primi modelli client-server e dei siti web che sviluppano il front-end per gli utenti e il back-end per i tecnici che troviamo la prima menzione ufficiale di cloud computing. Precisamente nel 1996, all’interno di un documento scritto da dirigenti tecnologici di Compaq (azienda statunitense finalizzata alla produzione nel settore dei personal computer), che andò a sostituire l’allora termine popolare “grid computing”.[1]

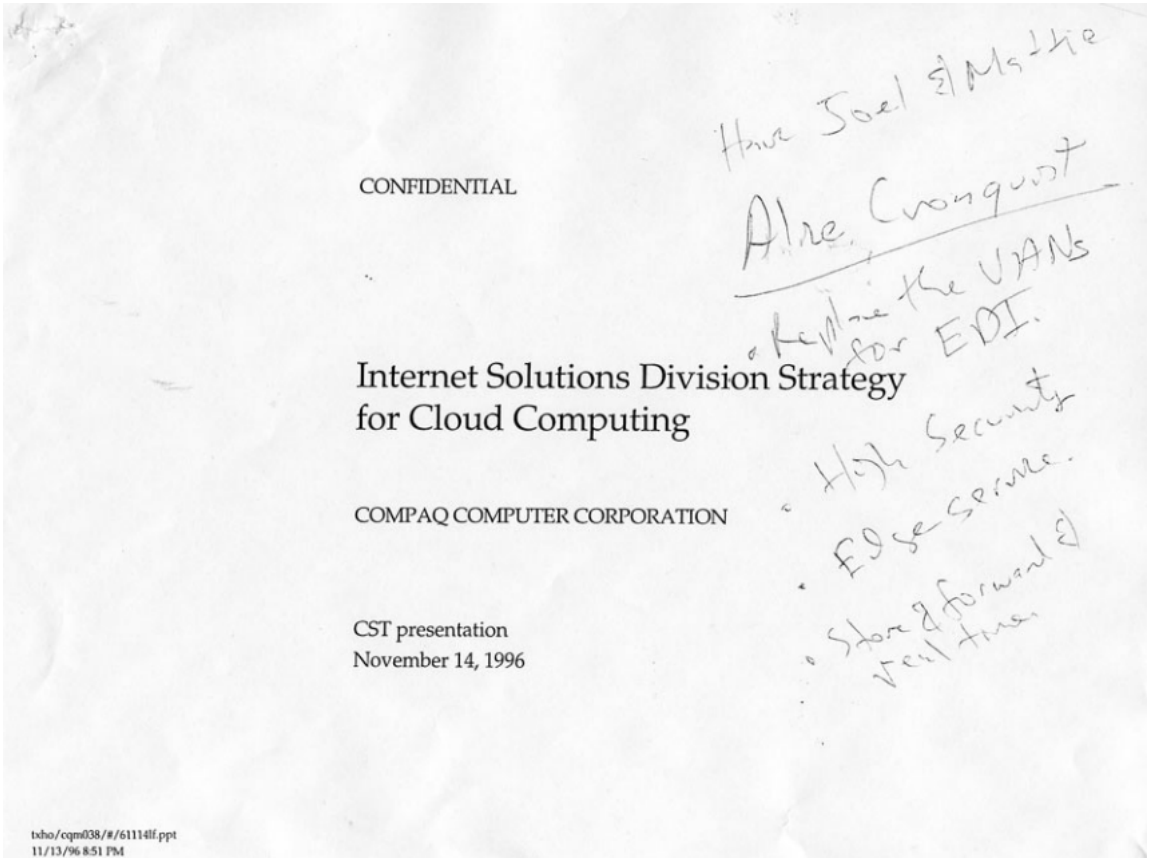


Figura 1.2: Prima menzione ufficiale del termine cloud computing
[3]

1.4 La nascita ufficiale del cloud computing

Nel momento in cui i personal computer sono diventati sempre più accessibili e tutti potevano finalmente connettersi, Salesforce è stata la prima azienda globale a offrire applicazioni in rete in modalità cloud lanciando il Software as a Service alle imprese.

La nascita ufficiale del cloud computing però, coincide con la creazione del primo cloud moderno realizzato da Amazon Web Services nel 2006, offrendo servizi di cloud computing su richiesta a prezzi accessibili e diventando il primo servizio su larga scala per il grande pubblico. A seguire, anche Google ha introdotto la sua Google Cloud Platform e Microsoft nel 2011 ha lanciato la sua prima applicazione, Microsoft Office, accelerando ancor di più la diffusione del cloud computing nel mondo.[1]

Capitolo 2

Introduzione Al Cloud Computing

2.1 Cos'è il cloud computing

Il cloud computing è un modello di erogazione di servizi offerti su richiesta da un fornitore a un utente finale in maniera on-demand e attraverso Internet. Volendo essere più precisi per addentrarci nell'argomento possiamo utilizzare la definizione di Cloud del NIST (National Institute of Standards and Technology): "Il cloud computing è un modello che consente un accesso di rete onnipresente, comodo e *on-demand* [1] a un pool condiviso di risorse informatiche configurabili (ad esempio reti, server, storage, applicazioni e servizi) che possono essere rapidamente fornite e rilasciate con il minimo sforzo di gestione o interazione con il fornitore di servizi. Questo modello cloud è composto da cinque caratteristiche essenziali, tre modelli di servizio e quattro modelli di distribuzione."[5]

In un modello cloud possiamo quindi riconoscere tre figure essenziali:[6]

- il fornitore di servizi: che offre servizi quali archiviazione,server e applicazioni solitamente secondo un modello *pay per use* (PPU) [2].
- utente amministratore: sceglie e configura i servizi offerti dal fornitore per l'utente finale.
- utente finale: utilizza i servizi forniti.

In determinati casi il cliente amministratore e il cliente finale possono coincidere.

[1] Per on demand o su richiesta, nel campo dell'informatica aziendale, si intende l'accesso alle risorse informatiche tramite internet solo quando necessario, eventualmente pagando le stesse in base all'utilizzo e non in base a un canone fisso, o acquistando una licenza una tantum.[4]

[2] Il termine PPU indica una forma di remunerazione in base alla quale si paga in funzione dell'utilizzo effettivo.[7]

2.1.1 Precisazione

Molto spesso i servizi cloud vengono erroneamente fraintesi con i servizi offerti da un datacenter, riguardo a questo la direttiva UE 2022/2555 NIS2 definisce:[8]

- Cloud: I servizi di cloud computing dovrebbero comprendere servizi digitali che consentono l'amministrazione su richiesta di un pool scalabile ed elastico di risorse di calcolo condivisibili e l'ampio accesso remoto a quest'ultimo, anche quando tali risorse sono distribuite in varie ubicazioni.
- Datacenter: Il termine «servizio di data center» dovrebbe applicarsi alla fornitura di un servizio che comprende strutture, o gruppi di strutture, dedicate a ospitare, interconnettere e far funzionare in modo centralizzato apparecchiature informatiche e di rete che forniscono servizi di conservazione, elaborazione e trasporto di dati insieme a tutti gli impianti e le infrastrutture per la distribuzione dell'energia e il controllo ambientale.

2.2 Caratteristiche essenziali dei modelli cloud

Come definito in precedenza il modello cloud è composto da cinque caratteristiche fondamentali:[5] [9]

- On-demand self-service : Un utente può richiedere i vari servizi offerti dal fornitore in totale autonomia, senza dover passare per gestori e/o provider.
- Broad network access: Tramite hardware fisico distribuito a livello globale, le funzionalità offerte sono disponibili in rete e accessibili ovunque.
- Resource Pooling: Le risorse informatiche in un modello cloud vengono raggruppate per servire più utenti utilizzando un modello multi-tenant, in questo modo le risorse sono suddivise dinamicamente in base alla domanda dell'utente rendendo così l'hardware cloud completamente ottimizzato per il massimo utilizzo.
- Rapid elasticity: Le infrastrutture cloud possono crescere e ridursi in modo dinamico, consentendo agli utenti di richiedere che le loro risorse di calcolo si adattino automaticamente alle richieste di traffico, avendo così la massima scalabilità.
- Measured service: I sistemi cloud controllano e ottimizzano automaticamente l'utilizzo delle risorse che possono essere monitorate, controllate e segnalate (tipicamente usate nei PPU - pay per use), garantendo trasparenza sui costi di utilizzo del servizio utilizzato al consumatore.

2.3 Modelli di servizio

I servizi cloud sono formati da infrastrutture, piattaforme o software in hosting presso provider esterni e messi a disposizione degli utenti attraverso Internet. Partendo dal modello "base" chiamato on-premise/on-site in cui è l'utente il proprietario e responsabile di ogni aspetto, dall'hardware alle applicazioni, fino alla scalabilità, esistono tre modelli di servizi cloud principali: *IaaS* (Infrastructure as a Service [3]), *PaaS* (Platform as a Service) e *SaaS* (Software as a Service).

Questi termini si riferiscono al modo in cui il cloud viene utilizzato all'interno dell'organizzazione e al grado di gestione di cui si è responsabili. Di seguito andremo a definirne le caratteristiche:[11]

- IaaS: Con il modello Infrastructure as a Service, o IaaS, il provider fornisce al cliente un'infrastruttura completa che include server fisici, storage, rete e componenti di virtualizzazione, in questo modo l'utente accede a tale infrastruttura, sostanzialmente a noleggio, tramite un'*API* [4] o una dashboard specifica, e gestisce aspetti come il sistema operativo, le app e il *middleware* [5] senza doversi preoccupare della gestione, della manutenzione o dell'aggiornamento dell'hardware sottostante, dei data center, delle connessioni di rete o degli interventi in caso di guasti. Questo modello consente quindi alle aziende di ridurre i costi legati all'infrastruttura fisica e di concentrarsi sugli aspetti applicativi e di business, lasciando al provider le complessità operative e la continuità del servizio.

- SaaS: Con il modello software as a Service, o SaaS, il provider gestisce interamente un'applicazione software che fornisce agli utenti; solitamente, queste applicazioni sono accessibili tramite un browser web che permette di eliminare la necessità di dover installare le applicazioni sui computer dei singoli utenti. L'utente che si connette a queste applicazioni tramite un'API o una dashboard dedicata dovrà preoccuparsi solamente della manutenzione del software.

- PaaS: con il modello Platform as a Service, o PaaS, il provider fornisce e si occupa di tutte le risorse hardware e software per lo sviluppo delle applicazioni tramite cloud, mentre l'utente gestisce le app eseguite sulla piattaforma e i dati che esse utilizzano. Questi servizi, che inoltre offrono agli utenti una piattaforma cloud condivisa per lo sviluppo e

[3] L'espressione "As a Service" indica che il modello di servizio viene offerto da una terza parte nel cloud e che quindi non è necessario acquistare, gestire o utilizzare hardware, software o strumenti come nel modello on-premise.[10]

[4] API o application Programming Interface, ovvero "interfaccia di programmazione delle applicazioni", in informatica è un insieme di regole e protocolli che consente a diversi programmi software e servizi di comunicare e scambiare dati, caratteristiche o funzionalità tra loro.[12]

[5] Il middleware è un software che funge da strato intermedio tra le applicazioni e le componenti sottostanti, come ad esempio sistemi operativi, database o hardware, facilitando la comunicazione, l'integrazione e la gestione delle risorse nei sistemi distribuiti. Il suo ruolo primario è quello di nascondere la complessità dell'infrastruttura sottostante, garantendo interoperabilità, scalabilità e sicurezza.[13]

la gestione delle applicazioni (un aspetto importante della metodologia *DevOps* [6]), sono pensati soprattutto per sviluppatori e programmatori.

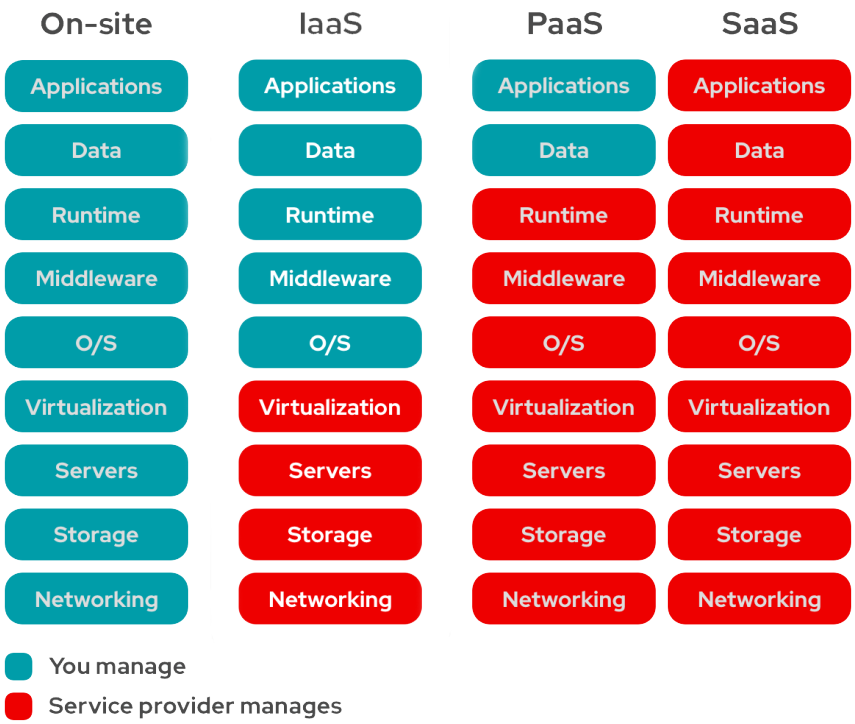


Figura 2.1: Tabella modelli di servizio cloud
[11]

[6]DevOps (dalla contrazione inglese di development, "sviluppo", e operations, qui simile a "messa in produzione" o "deployment") è una metodologia di sviluppo del software utilizzata in informatica che punta alla comunicazione, collaborazione e integrazione tra sviluppatori e addetti alle operations, puntando ad aiutare un'organizzazione a sviluppare in modo più rapido ed efficiente prodotti e servizi software.[14]

Oltre a queste tre principali categorie, è possibile trovare anche altri tipi di cloud che usano tecnologie differenti, come ad esempio i *container* [7] che hanno avuto sempre più successo nell'architettura a microservizi, portando alla creazione del Modello CaaS (Container as a Service). Con questo modello il provider fornisce e gestisce tutte le risorse hardware e software per lo sviluppo e il deployment delle applicazioni con i container utilizzati come risorsa principale al posto delle macchine virtuali. In questo modo sviluppatori e team delle operazioni IT possono sfruttare il modello CaaS per sviluppare, eseguire e gestire le applicazioni senza dover creare e mantenere l'infrastruttura o la piattaforma per l'esecuzione e la gestione dei container preoccupandosi solo della scrittura del codice e della gestione dei dati e delle applicazioni.[10]

[7]Container: I container consentono di raggruppare e isolare le applicazioni insieme al relativo ambiente di runtime, che include tutti i file necessari per l'esecuzione. In questo modo è più facile spostare applicazioni tra gli ambienti (sviluppo, test, produzione, ecc.) conservandone tutte le funzionalità.[15]

2.4 Modelli di distribuzione

Dopo aver analizzato le caratteristiche fondamentali del cloud computing, è importante capire come questa infrastruttura viene distribuita e resa disponibile agli utenti. In base a chi possiede e gestisce le risorse (hardware e non), il modo in cui sono allocate e gli accessi concessi si denotano diverse tipologie di modelli di distribuzione:[11]

- Cloud pubblico: I cloud pubblici sono ambienti cloud basati su un'infrastruttura IT che non appartiene all'utente finale e solitamente vengono eseguiti off-premise (non fisicamente nei locali del cliente) anche se ad oggi molti provider di cloud pubblici eseguono i propri servizi nei datacenter on-premise dei propri clienti. I provider di cloud pubblico più importanti sono Amazon Web Services (AWS), Google CLoud, IBM Cloud, Microsoft Azure e Alibaba Cloud.

  Tutti i cloud diventano pubblici quando gli ambienti sono suddivisi e ridistribuiti su più *tenant*[8]. Nemmeno le strutture tariffarie caratterizzano più i cloud pubblici, poiché alcuni provider consentono ai tenant di usare i loro cloud gratuitamente. L'infrastruttura IT *bare metal,* [9] utilizzata dai provider di cloud pubblico, può essere astratta e venduta come IaaS, oppure sviluppata in piattaforma cloud e proposta come PaaS.

- Cloud privato: I cloud privati sono ambienti cloud riservati a un singolo utente o gruppo ed eseguiti dietro il loro *firewall.* [10] Non occorre più che i cloud privati abbiano origine in un'infrastruttura IT on-premise. Le organizzazioni infatti realizzano i propri cloud privati su datacenter in affitto, di proprietà del fornitore e ubicati off-premise.

  Quando l'infrastruttura IT sottostante è dedicata a un singolo cliente, con un accesso completamente isolato, tutti i cloud diventano privati. Questa strutturazione genera anche diversi sottotipi di cloud privato, tra cui:

  - Cloud privato gestito: Permette di creare e usare un cloud privato che viene implementato, configurato e gestito da un fornitore terzo, questo lo rende adatto alle aziende di piccole dimensioni o con personale IT non specializzato.
  - Cloud dedicato: Si tratta di un cloud contenuto in un altro cloud che può esistere all'interno di un cloud pubblico o di un cloud privato. Ad esempio, il reparto contabilità di un'azienda può utilizzare il proprio cloud dedicato, ubicato all'interno del cloud privato aziendale.

- Cloud ibrido: I cloud ibridi appaiono come un singolo ambiente IT creato a partire da più ambienti connessi tramite reti LAN (Local Area Network), WAN (Wide Area Network) o VPN (Virtual Private Network) e/o API. Un cloud ibrido può, ad esempio, dover includere:

[8] Un tenant può essere un singolo utente ma, più frequentemente, è un gruppo di utenti, come l'organizzazione di un cliente, che condivide un accesso comune all'istanza dell'applicazione e i privilegi al suo interno.[16]

[9] È una forma di cloud service in cui l'utente noleggia una macchina fisica da un provider che non condivide con altri tenant. In questo modo gli utenti hanno il controllo completo sulla macchina fisica e hanno la possibilità di scegliere il loro sistema operativo, evitare le problematiche legate ai vicini delle infrastrutture condivise e adattare hardware e software per carichi di lavoro specifici.[17]

[10] Nell'informatica e nell'ambito delle reti di computer, un firewall è un componente hardware e/o software di difesa perimetrale di una rete, che può anche svolgere funzioni di collegamento tra due o più segmenti di rete, o tra una rete e un computer locale, fornendo dunque una protezione in termini di sicurezza informatica della rete stessa e proteggendo il computer da malware o altri pericoli di internet.[18]

  - Almeno un cloud privato e almeno un cloud pubblico.
  - Due o più cloud privati.
  - Due o più cloud pubblici.
  - Un ambiente bare metal o virtuale connesso almeno a un cloud, pubblico o privato.

  Ogni sistema IT si trasforma in cloud ibrido quando le app che vi risiedono possono muoversi tra più ambienti, separati ma connessi, e questi ultimi devono essere gestiti come un singolo ambiente.

- Multicloud: Un ambiente multicloud è formato da più di un servizio cloud e da più di un fornitore di servizi cloud, pubblici o privati. Tutti i cloud ibridi sono multicloud, ma non tutti i multicloud sono cloud ibridi. I multicloud diventano cloud ibridi quando i diversi cloud vengono connessi tramite una forma di integrazione o di orchestrazione. L'utilizzo di questo ambiente è sempre più frequente nelle aziende il cui è obiettivo è migliorare sicurezza e prestazioni attraverso l'utilizzo di più ambienti.

Public Cloud Versus Private Cloud		
	Public Cloud	**Private Cloud**
Description	A cloud platform available to the general public	A cloud platform dedicated to a single organization
Pros	• Scalability • Simplicity • No upfront capital costs	• Complete control • Isolated from other users • Potentially increased performance
Cons	• Less control • Less isolation	• Complexity and cost of infrastructure management • Harder to scale
Example platforms	AWS, Azure, Oracle Cloud, and Akamai Connected Cloud	Nutanix Enterprise Cloud, Red Hat OpenStack Platform, VMware Cloud

Figura 2.2: Pro e contro tra cloud pubblico e cloud privato
[19]

Capitolo 3

L’architettura Del Cloud Computing

Per "architettura cloud" si intende il modo in cui le singole tecnologie sono integrate per creare cloud: ambienti IT che astraggono, raggruppano e condividono risorse scalabili attraverso una rete. È anche il modo in cui tutti i componenti e le funzionalità necessarie per creare un cloud sono connessi per erogare una piattaforma online su cui eseguire le applicazioni. In sostanza, l’infrastruttura cloud comprende tutto il materiale necessario, mentre l’architettura cloud costituisce il progetto.[20]

3.1 Componenti dell’architettura cloud

L’architettura del cloud computing integra quattro componenti essenziali per creare un ambiente IT che astrae, raggruppa e condivide risorse scalabili in uno o più ambienti cloud rendendole disponibili ai propri clienti. Questa architettura varia in base ai driver di business e ai requisiti tecnologici che caratterizzano un’organizzazione, ma condividono tutte lo stesso obiettivo che è quello di creare una roadmap che consideri i workload delle applicazioni, i modelli di distribuzione nel cloud, la gestione dei servizi e le esigenze di progettazione.

I componenti fondamentali sono:[21]

- Un front-end
- Un back-end
- Una rete
- Una piattaforma di distribuzione basata su cloud

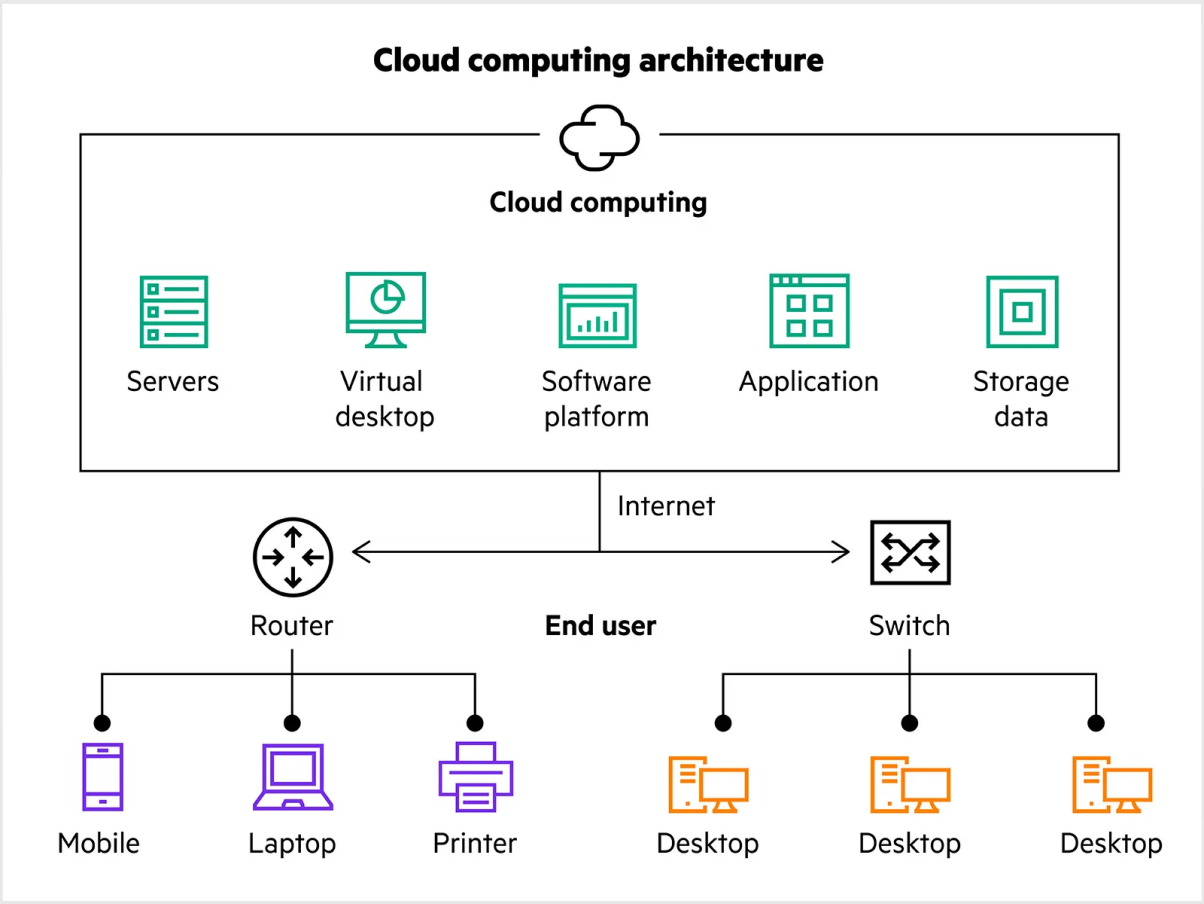


Figura 3.1: Architettura di un modello cloud
[22]

3.1.1 Il front-end

La piattaforma front-end è l'interfaccia attraverso la quale gli utenti interagiscono direttamente con i servizi cloud e include: browser web, client e dispositivi mobili. I browser web costituiscono il punto di accesso principale per i servizi cloud, in quanto consentono agli utenti di interagire con le applicazioni e le risorse in hosting nel cloud. I client, che possono essere dispositivi o software, trasferiscono il carico dell'elaborazione e dello storage sui server cloud, migliorando l'efficienza e le prestazioni. I dispositivi mobili come smartphone e tablet accedono ai servizi cloud tramite applicazioni o browser dedicati, in modo che gli utenti si possano connettere alle risorse praticamente da qualsiasi luogo.[23]

3.1.2 Il back-end

Mentre il front-end include tutti gli elementi con cui il cliente interagisce, il back-end (o "lato server") è la vera e propria spina dorsale dell'architettura cloud e si riferisce alla strutturazione del sito e alla programmazione delle sue funzionalità principali. Fornisce tutta la tecnologia dietro le quinte (server cloud, database cloud,API per accedere ai file) usata dal CSP (Cloud Service Provider) per supportare il front-end, compreso l'intero codice che aiuta un database o un server Web a comunicare con un browser Web o un sistema operativo mobile.

I componenti dell'architettura cloud back-end includono:[21]

- Applicazioni: le applicazioni back-end sono il software o le piattaforme che forniscono le richieste di servizio client sul front-end.
- Servizio di cloud computing: il servizio back-end fornisce le utility nell'architettura cloud e gestisce l'accessibilità delle risorse basate sul cloud (come servizi di storage basati sul cloud, servizi di sviluppo di applicazioni, servizi Web, servizi di sicurezza e altro).
- Tempo di esecuzione nel cloud: il tempo di esecuzione fornisce l'ambiente per la messa in opera o l'esecuzione dei servizi. La virtualizzazione svolge un ruolo cruciale nell'abilitazione di più tempi di esecuzione sullo stesso server.
- Storage cloud: lo storage cloud nel back-end si riferisce al servizio di storage flessibile e scalabile oltre alla gestione dei dati archiviati per eseguire le applicazioni.
- Infrastruttura: l'infrastruttura è costituita da tutte le risorse, dall'hardware back-end e da tutti i software utilizzati per eseguire e gestire i servizi basati su cloud.
- Software di gestione: il middleware coordina la comunicazione tra front-end e back-end in un sistema di cloud computing. Questo componente consente la fornitura di servizi in tempo reale per garantire esperienze fluide tra utente e front-end.
- Strumenti di sicurezza: gli strumenti di sicurezza forniscono la sicurezza back-end contro potenziali attacchi informatici o guasti del sistema. I firewall virtuali proteggono le applicazioni web, prevengono la perdita di dati e garantiscono il *backup e il disaster*

recovery [1]. I componenti di back-end includono la crittografia, la limitazione dell'accesso e i protocolli di autenticazione per proteggere i dati dalle violazioni.

3.1.3 Una rete

Una rete cloud dovrebbe fornire un'elevata larghezza di banda e una bassa latenza, consentendo agli utenti di accedere costantemente ai propri dati e alle proprie applicazioni. La rete deve inoltre offrire agilità in modo che l'accesso alle risorse possa verificarsi rapidamente ed efficientemente tra server e ambiente basato sul cloud. Altri importanti dispositivi di rete con architettura cloud includono i bilanciatori del carico, le content delivery network (CDN) [2] e la Software-Defined Networking (SDN)[3] per garantire flussi di dati senza interruzioni e in sicurezza tra utenti front-end e risorse back-end. [21]

3.1.4 Modelli di servizio cloud

Per finire, come ultimo componente dell'architettura cloud troviamo i modelli di servizio cloud. Dei tre principali, ovvero: IaaS, PaaS e SaaS ne abbiamo già parlato nel capitolo 2.3, ma ne esistono anche altri altrettanto popolari, come: [21]

- Serverless computing (o serverless): il serverless è un modello di sviluppo ed esecuzione di applicazioni cloud che consente agli sviluppatori di creare ed eseguire un codice senza eseguire il provisioning o gestire server o infrastrutture di back-end.
- Business-Process-as-a-Service (BPaaS): BPaaS è una piattaforma di outsourcing di processi aziendali che combina servizi IaaS, PaaS e SaaS.
- Function-as-a-Service (FaaS): il FaaS è un sottoinsieme di SaaS in cui il codice dell'applicazione viene eseguito solo in risposta a eventi o richieste specifiche.

3.2 Principali tecnologie per l'architettura cloud

- Virtualizzazione: Fondamentale per l'architettura cloud, la virtualizzazione ha funzione di livello di astrazione che consente di suddividere le risorse hardware di un singolo computer (processori, memoria, storage e altro) in più computer virtuali noti come macchine virtuali (VM). La virtualizzazione collega i server fisici gestiti da un provider di cloud service (CSP) in numerose località, quindi divide e astrae le risorse in modo da renderle accessibili agli utenti finali ovunque sia presente una connessione a Internet. [21]

[1] Il backup e il disaster recovery comportano la creazione o l'aggiornamento periodico di più copie dei file, la loro memorizzazione in una o più sedi remote e l'utilizzo delle copie per continuare o riprendere le operazioni aziendali in caso di perdita di dati dovuta a danneggiamenti dei file e dei dati, attacchi informatici o disastri naturali. [24]

[2] Una CDN (Content Delivery Network) è una rete di server geograficamente dispersa per consentire prestazioni web più veloci localizzando copie di contenuti web più vicine agli utenti o facilitando la distribuzione di contenuti dinamici (ad esempio, feed video in diretta). [25]

[3] L'SDN è un approccio al networking che utilizza controller software che possono essere gestiti da API per comunicare con l'infrastruttura hardware per dirigere il traffico di rete. [26]

- Automation: L'automazione del cloud implica l'implementazione di strumenti e processi che riducono o eliminano il lavoro manuale associato alla configurazione e alla gestione degli ambienti cloud. Gli strumenti di automazione del cloud vengono eseguiti su ambienti virtualizzati e svolgono un ruolo essenziale nel consentire alle organizzazioni di utilizzare al meglio i benefici del cloud computing, come la possibilità di usare le risorse nel cloud on demand e di aumentarle o ridurle in base alle necessità. L'automazione svolge un ruolo fondamentale nei flussi DevOps, accelerando le attività legate alla creazione, al test, alla distribuzione e al monitoraggio delle applicazioni, con conseguenti risparmi sui costi.[21]

- Container: I container raggruppano un'applicazione e le sue dipendenze in una singola unità eseguibile in modo coerente in diversi ambienti di elaborazione. A differenza delle VM, i container condividono il kernel del sistema operativo host, il che li rende leggeri e più rapidi da avviare. Per questo motivo i container risultano ideali per le architetture dei microservizi, in cui le applicazioni sono suddivise in servizi più piccoli e gestibili. I container migliorano la portabilità e la coerenza, garantendo che le applicazioni vengano eseguite allo stesso modo, indipendentemente dall'infrastruttura sottostante.[23]

3.3 Best practice per l'architettura cloud

Un'architettura cloud ben definita dovrebbe includere best practice e linee guida per aiutare gli architetti a creare soluzioni cloud che siano performanti e sicure. Le best practice dovrebbero includere quanto segue:[21]

- Automatizzare le operazioni per ridurre i costi e supportare l'affidabilità, la disponibilità e la sicurezza della soluzione.

- Rispettare la data gravity, ovvero il concetto che i dati hanno una loro massa e una loro forza, si dovrebbero adottare strategie di backup e disaster recovery assicurando la protezione dei dati e la continuità operativa in caso di guasti o disastri. Maggiore è la massa di dati, maggiore è lo sforzo necessario per spostarli, che di solito si traduce in più tempo, costi e potenza di elaborazione.

- Scegliere la piattaforma migliore per ogni workload per utilizzarne al meglio le funzionalità al fine di ottimizzare i livelli di servizio e le caratteristiche operative del workload.

Capitolo 4

Trasformazione Digitale

La trasformazione digitale utilizza le moderne tecnologie digitali, inclusi tutti i tipi di piattaforme cloud pubbliche, private e ibride, per creare o modificare le esperienze dei clienti, la cultura e i processi aziendali per soddisfare le mutevoli dinamiche di business e di mercato. La trasformazione digitale è mossa da una serie di tecnologie innovative che stanno rimodellando i settori e ridefinendo le possibilità di business. Queste tecnologie consentono alle organizzazioni di semplificare le operazioni, ottenere informazioni preziose dai dati e offrire prodotti e servizi innovativi. Alcune delle tecnologie principali che guidano la trasformazione digitale includono appunto il cloud computing.[27] Ma quali sono i vantaggi e le limitazioni di questa tecnologia?

4.1 Vantaggi del cloud computing

Il passaggio al cloud computing ha trasformato completamente il modo in cui lavoriamo, comunichiamo e collaboriamo, diventando rapidamente una necessità per rimanere competitivi nel mondo digitale di oggi. Di seguito parleremo dei principali vantaggi e svantaggi e del perché si dovrebbe considerare di passare ai servizi cloud.[28]

- Time to market più rapido: puoi avviare nuove istanze o eseguirne il ritiro in pochi secondi, consentendo agli sviluppatori di accelerare lo sviluppo con deployment rapidi. Il cloud computing supporta le nuove innovazioni semplificando l'esecuzione di test di nuove idee e la progettazione di nuove applicazioni senza limitazioni hardware o processi di approvvigionamento lenti.
- Scalabilità e flessibilità: Il cloud computing offre maggiore flessibilità alle aziende. Puoi scalare rapidamente le risorse e l'archiviazione per soddisfare le esigenze aziendali senza dover investire in infrastrutture fisiche. Le aziende non devono pagare o creare l'infrastruttura necessaria per supportare i loro livelli di carico più elevati. Allo stesso modo, possono fare lo scale down rapidamente se le risorse non vengono utilizzate.
- Risparmi sui costi: Qualunque modello di servizio cloud tu scelga, paghi solo per le risorse effettivamente utilizzate. In questo modo eviterai di sovraccaricare ed effettuare l'overprovisioning del tuo data center e permetterai ai tuoi team IT di risparmiare tempo prezioso da dedicare a un lavoro più strategico.
- Collaborazione più efficace: Il cloud ti consente di rendere i dati disponibili ovunque ti trovi, in qualsiasi momento. Anziché essere vincolati a una località o a un dispositivo

specifico, gli utenti possono accedere ai dati da qualsiasi parte del mondo e da qualsiasi dispositivo, purché abbiano una connessione a Internet.

- Sicurezza avanzata: Il cloud computing può di fatto rafforzare la tua sicurezza grazie alla profondità e all'ampiezza delle funzionalità di sicurezza, alla manutenzione automatica e alla gestione centralizzata, che implementano misure di sicurezza robuste, tra cui crittografia, gestione delle identità e degli accessi.
- Prevenzione della perdita di dati: I cloud provider offrono funzionalità di backup e ripristino di emergenza. L'archiviazione dei dati nel cloud anziché localmente può contribuire a prevenire la perdita di dati in caso di emergenza, come malfunzionamenti dell'hardware, minacce dannose o persino semplici errori degli utenti.
- Sostenibilità ambientale: I fornitori di servizi cloud si concentrano sempre di più sulla sostenibilità, incrementando l'efficienza energetica dei data center e scegliendo le fonti rinnovabili. Con i servizi cloud, le organizzazioni possono ridurre le loro emissioni di anidride carbonica e contribuire alla sostenibilità ambientale.[23]

4.1.1 Limitazioni del cloud computing

Uno degli svantaggi più comuni del cloud computing è che si basa su una connessione a Internet. L'informatica tradizionale utilizza una connessione cablata per accedere ai dati su server o dispositivi di archiviazione. Con il cloud computing, una connessione errata potrebbe impedirti di accedere alle informazioni o alle applicazioni di cui hai bisogno. Anche i principali fornitori di servizi cloud possono riscontrare tempi di inattività a causa di una calamità naturale o di prestazioni più lente causate da un problema tecnico imprevisto che potrebbe influire sulla connettività. Altre limitazioni potrebbero essere:[28]

- Rischio di vincoli al fornitore
- Un minor controllo sull'infrastruttura cloud sottostante
- Preoccupazioni relative a rischi per la sicurezza quali privacy dei dati e minacce online
- Complessità di integrazione con i sistemi esistenti
- Costi e spese imprevisti

A questo punto, è evidente che i vantaggi superano le limitazioni. La maggior parte delle aziende oggi non valuta se eseguire la migrazione al cloud, ma cosa dovrebbe migrare. Il cloud offre più flessibilità e affidabilità, migliora le prestazioni, l'efficienza e consente di ridurre i costi IT, migliorando inoltre l'innovazione. Questi vantaggi primari possono anche tradursi in altri vantaggi correlati che possono aiutare ad aumentare la produttività, supportare la forza lavoro da remoto e migliorare l'efficienza operativa. Inoltre, è importante ricordare che intraprendere il percorso verso il cloud non è necessariamente uno scenario del tipo "o tutto o niente". Ad esempio, molte aziende stanno scoprendo che l'adozione di un approccio ibrido può aiutare ad ampliare la capacità e le funzionalità dell'infrastruttura esistente.

4.2 Migrazione al cloud

La cosiddetta "Cloud Migration" è al centro del processo di trasformazione digitale delle aziende. Migrare verso il cloud vuol dire sostanzialmente muovere dati, risorse, applicazioni e i vari elementi di business in un ambiente tecnologico condiviso, scalabile e flessibile. Il principale obiettivo è quello di riuscire ad ospitare e valorizzare dati e applicazioni in un ambiente il più ottimale possibile per l'organizzazione, con notevoli benefici in termini di costi, performance e sicurezza.[29] Il cloud computing aiuta le aziende ad archiviare, elaborare e accedere ai dati ed è proprio per questo che abbiamo deciso di utilizzare questa tecnologia nella nostra web app il cui sviluppo e approfondimento saranno oggetto di analisi nei prossimi capitoli.

Capitolo 5

Requisiti e Architettura dell'Applicazione

La web app che ho sviluppato è stata progettata con l'obiettivo di gestire più aziende all'interno di un unico portale accessibile in locale dall'organizzazione. Lo scopo principale e il problema che risolve è quello di poter caricare e gestire file e documenti per ogni azienda registrata sul portale permettendo di sincronizzare i file caricati con un servizio di cloud storage esterno (Google Drive), in modo da:

- Garantire accessibilità ovunque.
- Mantenere i file aggiornati.
- Sfruttare le capacità di archiviazione e condivisione del cloud.
- Prevenire una possibile perdita di dati.

Per la realizzazione di questa applicazione abbiamo deciso di utilizzare un servizio cloud di tipo SaaS sfruttando quindi una piattaforma fornita interamente dal provider.

5.1 Requisiti

La web app deve implementare i seguenti requisiti:

- Pagina di registrazione: in questa pagina è possibile registrare un nuovo utente tramite email, con doppia verifica della password e con almeno 8 caratteri di lunghezza. Una volta registrati, viene inviata una mail di conferma all'indirizzo inserito che permette di abilitare l'account ed effettuare il login.
- Pagina di login: una pagina con la quale l'utente può autenticarsi tramite email (univoca) e password, quest'ultima crittografata all'interno del database tramite la funzione crittografica Bcrypt. Ogni utente possiede un ruolo che permette di svolgere o meno determinate funzioni all'interno della web app, così da garantire sicurezza e controllo degli accessi. Esistono due tipi di utenti:
  - ADMIN: possiede l'accesso completo alle funzionalità della web app, in particolare a quelle che riguardano la registrazione e la gestione delle aziende sul portale e la creazione di remote (directory) all'interno della piattaforma cloud.

  - USER: utente normale che sfrutta la funzione principale della web app, ovvero caricare e sincronizzare file e documenti delle aziende registrate

  Inoltre, a discrezione dell'utente è possibile abilitare (e disabilitare) l'autenticazione a due fattori (2FA) tramite Google Authenticator per garantire una sicurezza maggiore.

- HomePage: una volta eseguito il login con successo si viene reindirizzati alla homepage, in questa pagina ed in tutte le altre, ad eccezzione della login e registrazione, sono presenti delle "guardie" che rendono impossibile l'accesso, a meno che non si abbia effettuato l'autenticazione con successo reindirizzando l'utente alla pagina di login nel caso in cui l'utente non lo fosse. In questa pagina è possibile accedere alle funzioni legate al profilo, tramite una dropdown list identificata dall'email dell'utente, che sono:

  - Logout
  - Abilitazione della 2FA

  Se si è un utente ADMIN è possibile accedere alle funzioni di gestione che sono:

  - Registrare un nuovo utente
  - Registrare una nuova azienda
  - Creare un remote specifico per l'azienda all'interno di Google Drive (servizio che abbiamo scelto di utilizzare) nel quale poter sincronizzare i file caricati.

  Infine, è possibile accedere alla funzione principale di caricamento e sincronizzazione dei file che ci reindirizzerà alla pagina di gestione dedicata.

- Pagina di gestione file: questa pagina è composta da:

  - Una dropdown list che ci permette di selezionare una delle aziende registrate per le quali vogliamo caricare dei file.
  - Un bottone di sincronizzazione che svolge la funzione principale di mantenere sincronizzati i file in locale con quelli in cloud.
  - Due dataGrid (tabelle) che mostrano rispettivamente i file salvati in locale e i file salvati su cloud, così da darci un'idea della situazione generale.
  - Un bottone per l'upload dei file in locale.

  Inoltre è possibile visualizzare i file anche all'interno della web app.

- Una pagina per la registrazione delle aziende: in questa pagina è possibile registrare tramite apposito bottone nuove aziende inserendo: il nome, P.iva (univoca), numero di telefono, email ed eventuale url del sito. In questa pagina è anche presente una dataGrid che ci mostra tutte le aziende registrate al momento, permettendoci di modificarne i dati o eliminarle.
- Una pagina per la creazione di un remote (cartella dove sincronizzare i file per ogni azienda) su Google Drive: qui si potrà scegliere l'azienda per cui creare un remote di sincronizzazione, venendo poi guidati nella configurazione tramite RClone, un tool di cui parleremo in seguito.
- Una pagina per abilitare la 2FA: in questa pagina verrà mostrato un QR code univoco, generato con quickchart.io, che l'utente dovrà scannerizzare su Google Authenticator per poter avere il proprio autenticatore che fornirà il codice da utilizzare al momento del login avendo abilitato la 2FA.

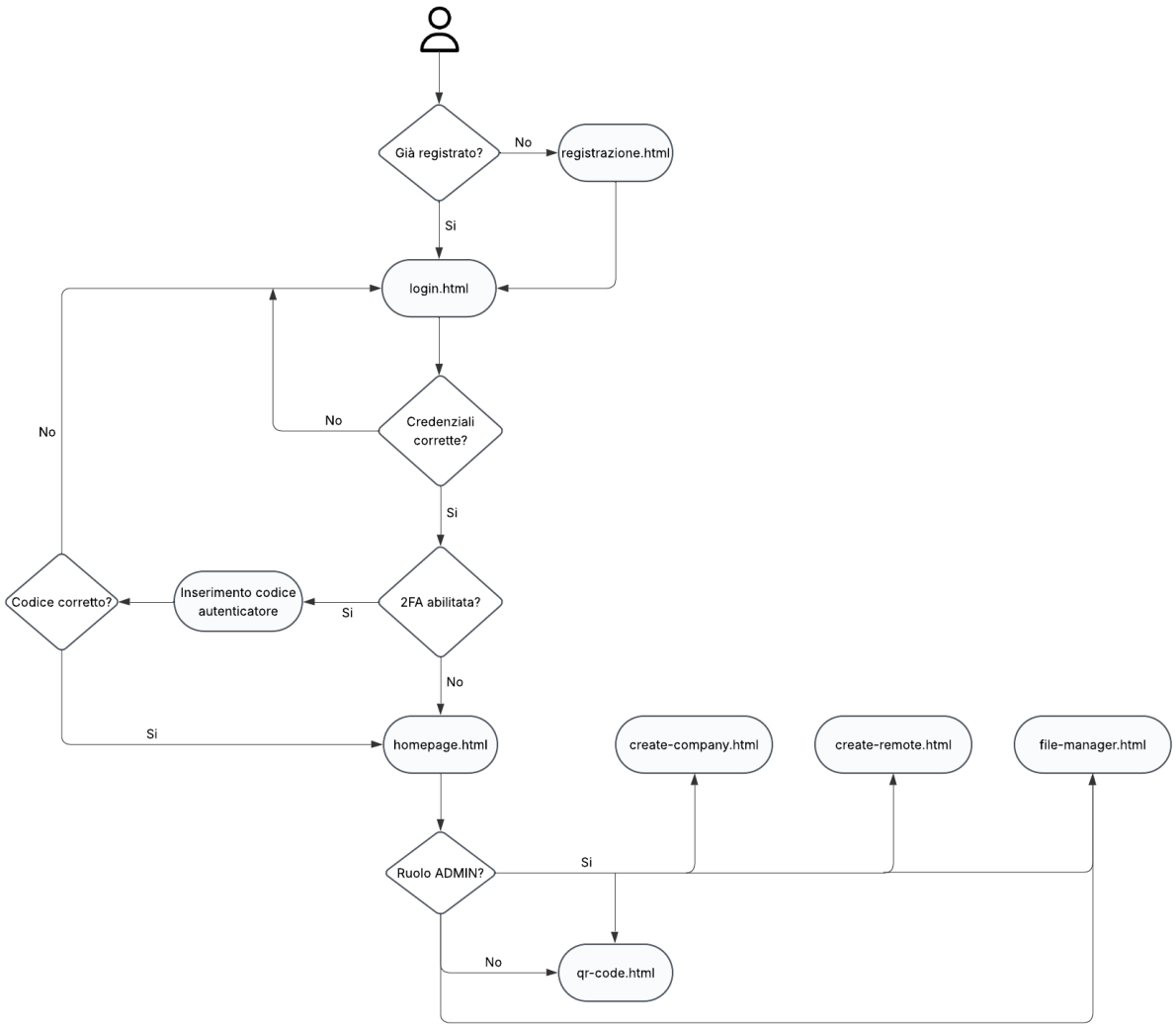


Figura 5.1: Pagine implementate

5.1.1 Back-end

Il back-end è stato implementato seguendo il pattern MVC che vedremo nel prossimo sottocapitolo insieme a tutte le tecnologie utilizzate, utilizzando il framework Spring Boot per implementare quanto segue:

- Entità:
  - Company
  - FileInfo
  - User
- Controller per gestire:
  - Registrazione
  - Homepage
  - Creazione e gestione delle aziende
  - Creazione e gestione dei remote
  - Caricamento e sincronizzazione dei file
  - Restituzione dei file locali e cloud
  - Creazione qr-code per 2FA
  - Varie configurazioni per la gestione della sicurezza al fine di gestire l'autenticazione degli utenti utilizzando Spring Boot Security, gestire la parte di invio dell'e-mail di conferma ogni volta che si registra un utente, implementare il login 2FA quando abilitato e creare un esecutore di comandi che permetta di eseguire i comandi RClone sul terminale in totale autonomia.

  Inoltre, alcuni controller sono stati creati per esporre delle API che restituiscono informazioni al front-end riguardo ai file e alle aziende.

5.1.2 Front-end

La parte di front-end deve implementare le seguenti pagine:

- registrazione.html
- login.html
- homepage.html
- create-company.html
- create-remote.html
- file-manager.html
- qr-code.html

5.1.3 Database

Per il database, che ci permette di mantenere e gestire tutti i dati relativi all'applicativo, abbiamo utilizzato un'immagine di mysql dockerizzata, quindi inserita all'interno di un container utilizzando l'applicativo Docker, permettendoci così di avviare e gestire il database in modo isolato e indipendente dall'ambiente di esecuzione dell'applicazione. Il database deve quindi implementare le seguenti tabelle:

- Users
- Authorities
- Company
- File

5.2 Architettura dell'applicazione

Un'applicazione web è un software applicativo eseguito su un server web e accessibile tramite browser da un utente. Una web app è, come visto nel capitolo precedente, costituita principalmente da:[30]

- Back-end: la parte che si occupa lato server di tutta la logica dell'applicativo e della comunicazione con il database.
- Front-end: la parte che si occupa lato client della presentazione dei dati all'utente garantendo l'interazione tra quest'ultimo e l'applicazione.
- Database: può essere opzionale ed è la parte che si occupa della gestione e della persistenza dei dati.

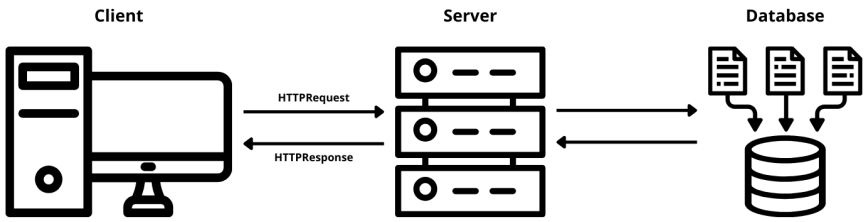


Figura 5.2: Architettura di un'applicazione web

È fondamentale costruire una buona architettura, la struttura portante della web app, affinché questi tre elementi siano organizzati e interagiscano nel modo corretto. Fortunatamente esistono dei design pattern che ci aiutano in questo, nel mio caso ho deciso di utilizzare il pattern MVC (Model View Controller).

5.2.1 Pattern MVC

Il Model-view-controller è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti e in applicazioni web, in grado di separare la logica di presentazione (front-end) dei dati dalla logica di business (back-end).[31] Questo pattern è caratterizzato da tre elementi principali:

- Model: fornisce i metodi necessari per accedere ai dati utili all'applicazione, nel nostro caso i model della nostra applicazione sono:
  - User
  - Company
  - FileInfo
- View: visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti, in particolare nella nostra web app si occuperà di mostrare e gestire le funzioni delle seguenti pagine:
  - registrazione.html
  - login.html
  - homepage.html
  - create-company.html
  - create-remote.html
  - file-manager.html
  - qr-code.html
- Controller: si occupa di implementare la logica vera e propria dell'applicazione, ricevere i comandi dell'utente (in genere attraverso la view) e attuarli modificando lo stato degli altri due componenti. Nel nostro caso i controller che abbiamo creato gestiscono:
  - Registrazione
  - Homepage
  - Creazione e gestione delle aziende
  - Creazione e gestione dei remote
  - Caricamento e sincronizzazione dei file
  - Restituzione dei file locali e cloud
  - Creazione qr-code per 2FA

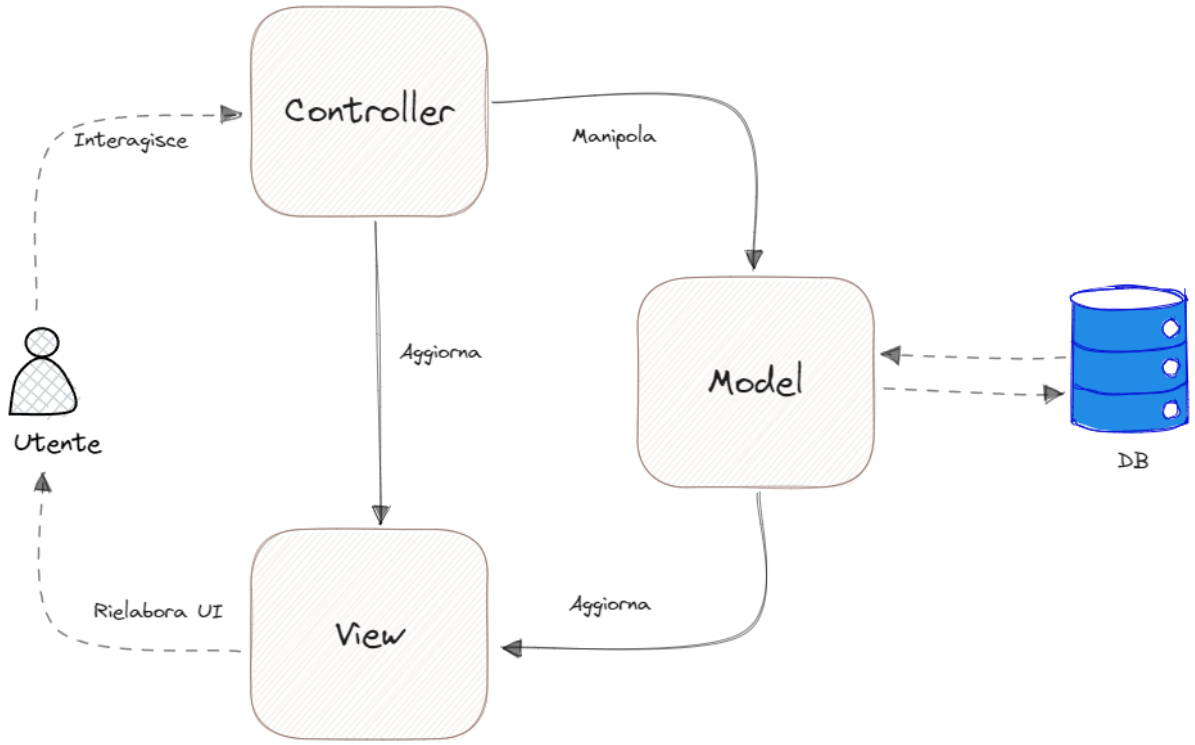


Figura 5.3: Interazione tra componenti del pattern MVC
[32]

5.3 Tecnologie utilizzate per front-end

Per lo sviluppo del front-end ho utilizzato le seguenti tecnologie:

- Angular
- Bootstrap

5.3.1 Angular

Angular è un framework open source per lo sviluppo di *single-page-application* [1] dinamiche, scalabili e performanti sviluppato da Google. Noto anche come Angular2 perché deriva dalla riscrittura completa di AngularJS con Typescript, un linguaggio di programmazione open source di Microsoft. Angular utilizza un'architettura basata sui componenti che permette agli sviluppatori di costruire interfacce e parti di interfaccia modulari e riutilizzabili.[34] Ogni componente è composto da:

- Un file HTML che specifica il template, ovvero quello che deve essere mostrato in pagina.
- Un file Typescript che definisce la logica e il funzionamento del componente.
- Un file CSS che definisce lo stile specifico per quel componente.

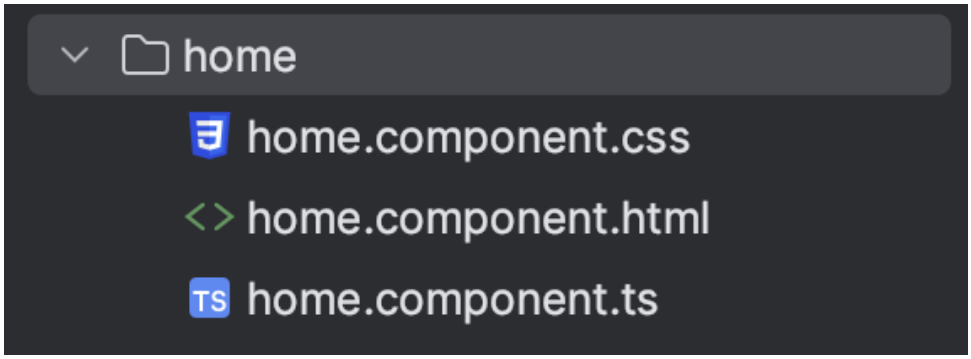


Figura 5.4: Esempio di componente Angular

Come abbiamo detto, Angular è scritto in TypeScript; questo linguaggio non è altro che un'estensione di Javascript che aggiunge elementi come tipi statici, interfacce, classi e la possibilità di definire funzioni anonime. Essendo quindi Typescript un'estensione, i programmi scritti in JavaScript possono essere eseguiti in TypeScript senza nessuna modifica. Altri elementi che caratterizzano Angular riguardano:

- Data Binding: Angular supporta il two-way data binding, ovvero la possibilità di mantenere sincronizzato il valore sottostante di un dato e la sua rappresentazione nell'interfaccia grafica.

[1] In informatica con Single-page application (SPA) si intende un'applicazione web o un sito web che può essere usato o consultato su una singola pagina web con l'obiettivo di fornire una esperienza utente più fluida e simile alle applicazioni desktop dei sistemi operativi tradizionali.[33]

- Dependency injection: Angular incorpora un sistema di Dependency injection che permette di gestire e iniettare facilmente le dipendenze dei componenti, promuovendo la modularità delle applicazioni.
- Direttive: Angular estende l'HTML attraverso le direttive, attributi HTML aggiuntivi che permettono di modificare il comportamento o l'aspetto degli elementi del *DOM* [2].
- Routing: Angular include un router che permette agli sviluppatori di definire e gestire gli stati delle applicazioni e i percorsi di navigazione, rendendo possibile costruire single page applications con strutture ad albero complesse.
- Angular CLI: Il progetto Angular include l'Angular CLI, uno strumento da linea di comando che facilita la creazione e lo sviluppo di applicazioni, alcuni comandi che include sono:
  - Il comando *ng new* che permette di creare una nuova applicazione Angular.
  - Il comando *ng generate* che permette di generare nuovi componenti con i quali è possibile comporre l'applicazione.
  - Il comando *ng serve* che permette di avviare l'applicazione per lo sviluppo locale.

[2] In informatica il Document Object Model è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti. [35]

5.3.2 Bootstrap

Bootstrap è un framework di sviluppo web open source progettato per semplificare la creazione di siti responsive e mobile-first, permettendo di adattare ogni singolo componente in modo automatico e dinamico alle dimensioni dello schermo col quale si sta visualizzando l'applicazione. Grazie alla sua struttura basata su HTML, CSS e JavaScript, Bootstrap fornisce componenti riutilizzabili e stili pronti all'uso che agevolano la progettazione di interfacce web moderne, riducendo il tempo necessario per scrivere codice da zero.[36] Il suo layout è basato su un sistema a griglia che utilizza container, righe e colonne per allineare i componenti in pagina; inoltre sfrutta sei responsive breakpoints che permettono ad ogni componente di adattarsi a qualsiasi dimensione di schermo.

Breakpoint	Class infix	Dimensions
Extra small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Figura 5.5: Responsive breakpoints in Bootstrap
[37]

Come abbiamo detto uno dei punti di forza di Bootstrap è quello di poter utilizzare componenti già pronti semplicemendo copiando e incollando il codice dalla documentazione nel nostro progetto, di seguito un esempio:

```
1  <div class="dropdown">
2    <button class="btn btn-secondary dropdown-toggle" type="button"
       data-bs-toggle="dropdown" aria-expanded="false">
3      Dropdown button
4    </button>
5    <ul class="dropdown-menu">
6      <li><a class="dropdown-item" href="#">Action</a></li>
7      <li><a class="dropdown-item" href="#">Another action</a></li>
8      <li><a class="dropdown-item" href="#">Something else here</a></
         li>
9    </ul>
10 </div>
```

Listing 5.1: Esempio di un componente Bootstrap (Dropdown)

5.4 Tecnologie utilizzate per back-end

Per l'implementazione del back-end sono state utilizzate diverse tecnologie:

- Java Spring Boot: per la gestione della logica lato server.
- Maven: un build automation tool
- Tomcat: un server di applicazioni Java
- Docker: per dockerizzare in un container il DB di Mysql
- JPA - Hibernate: per mappare oggetti Java in tabelle di un database relazionale
- RClone: tool per gestire la sincronizzazione dei file con le piattaforme cloud
- BCrypt: algoritmo per l'hashing delle password

5.4.1 Spring

Java Spring è un framework open source molto diffuso per la creazione di applicazioni autonome, adatte ad ambienti di produzione che vengono eseguite su JVM (Java Virtual Machine). Spring Framework prevede una funzione di inserimento delle dipendenze che consente agli oggetti di definire le proprie dipendenze che successivamente il container Spring provvederà ad inserire, questo permette agli sviluppatori di creare applicazioni modulari. Java Spring Boot invece, è uno strumento che semplifica e velocizza lo sviluppo di applicazioni web e microservizi con Spring Framework tramite tre funzionalità principali:[38]

- Configurazione automatica: le applicazioni vengono inizializzate con dipendenze preimpostate che non bisogna configurare manualmente. Grazie alla configurazione automatica, Spring Boot è in grado di configurare sia le impostazioni di Spring sia i pacchetti di terze parti. Questo consente di iniziare rapidamente a sviluppare applicazioni basate su Spring e a ridurre la possibilità di errori umani.
- Approccio categorico: Spring è in grado di scegliere i pacchetti da installare e i valori da predefiniti da utilizzare, piuttosto che lasciare all'utente il compito di prendere tutte queste decisioni e configurare tutto manualmente. Durante la creazione e configurazione del progetto è possibile selezionare le dipendenze necessarie attraverso un tool chiamato Spring Initializr (Figura 5.6, dove è stata aggiunta Spring Web come dipendenza), che permette tramite un modulo web di definire le esigenze del progetto.
- Applicazioni autonome: Spring Boot aiuta gli sviluppatori a creare applicazioni che sono semplicemente da eseguire. In particolare, ti consente di creare applicazioni autonome che vengono eseguite autonomamente integrando un server web come ad esempio Tomcat nell'applicazione durante il processo di inizializzazione, senza quindi dover fare affidamento su un server web esterno.

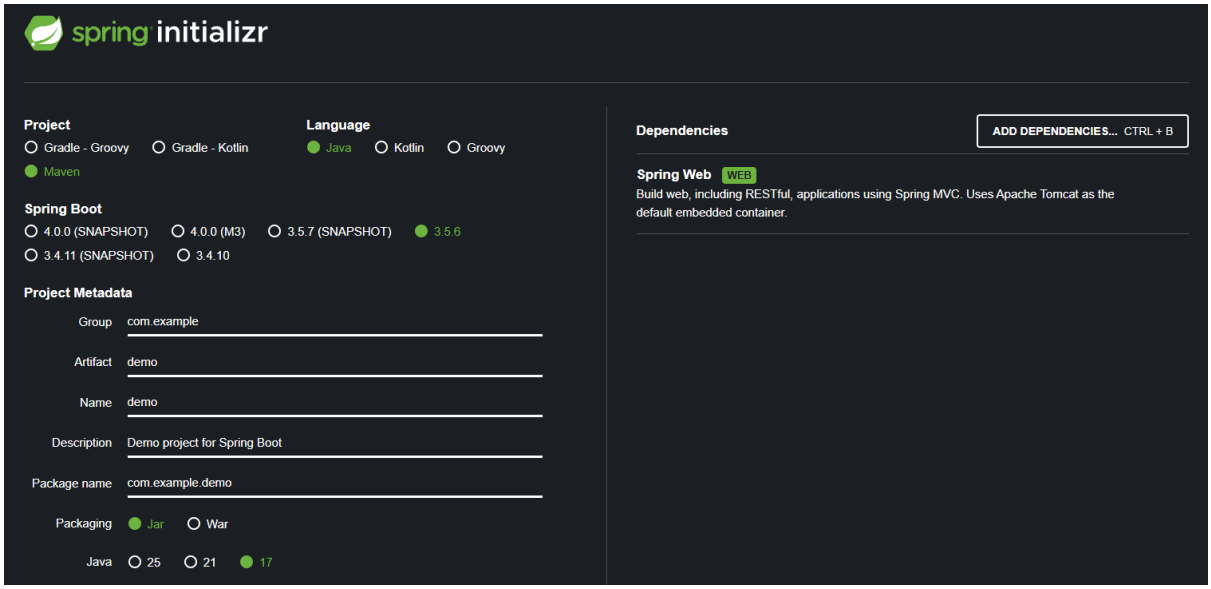


Figura 5.6: Spring initializr

5.4.2 Maven

Apache Maven è uno strumento di automazione build usato principalmente per i progetti Java. Rende più facile gestire e mantenere grandi progetti aiutando gli sviluppatori ad automatizzare il processo di build, test e distribuzione del software. Una delle caratteristiche principali di Maven è la sua capacità di gestire le dipendenze. Maven tiene traccia di tutte le librerie e di altre dipendenze di cui un progetto ha bisogno e le scarica automaticamente quando sono necessarie. Ciò rende facile per gli sviluppatori utilizzare librerie esterne nei loro progetti senza doverle scaricare e gestirle manualmente. Maven usa un approccio dichiarativo per specificare la build e le dipendenze del progetto utilizzando un file XML chiamato pom.xml (project object model).[39]

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.2</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>demo</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
  </properties>

  <dependencies>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

</project>
```

Listing 5.2: Esempio di pom.xml, nel quale sono state aggiunte le dipendenze web e database.

5.4.3 Tomcat

Apache Tomcat è un server web (nella forma di contenitore servlet) open source sviluppato da Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni web sviluppate in linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale e può essere utilizzata anche come contenitore servlet per framework come Spring.[40]. Nel nostro caso è stato utilizzato per separare la logica del back-end da quella del front-end. In questo modo abbiamo:

- Tomcat che gestisce esclusivamente le richieste API e la parte della logica di business del back-end.
- Il server front-end che serve esclusivamente l'app Angular, occupandosi della parte grafica e dell'interazione con l'utente.

Con questo approccio si hanno vantaggi come separazione chiara dei ruoli, maggiore scalabilità e gestione più semplice delle performance.

5.4.4 Docker

Docker è una piattaforma open source che consente agli sviluppatori di creare, implementare, eseguire, aggiornare e gestire i container che sono componenti standardizzati ed eseguibili che combinano il codice sorgente dell'applicazione con le librerie e le dipendenze del sistema operativo necessarie per eseguire tale codice in qualsiasi ambiente, semplificando lo sviluppo e l'implementazione di applicazioni distribuite. I container sono piccoli, veloci e portabili perché, a differenza di una virtual machine (VM), non hanno bisogno di includere un sistema operativo guest in ogni istanza, ma possono invece sfruttare le funzioni e le risorse del sistema operativo host.[41] Nella nostra app è stato utilizzato per dockerizzare l'istanza del db mysql tramite l'esecuzione di un file yaml (5.7) con il comando: *docker-compose up -d*

```
version: '3.8'

networks:
  default:

services:
  db:
    platform: linux/x86_64
    image: mysql:5.7
    container_name: book
    ports:
      - 3306:3306
    volumes:
      - "./.data/db:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD: pass
      MYSQL_DATABASE: book
```

Listing 5.3: File di configurazione docker-compose.yml.

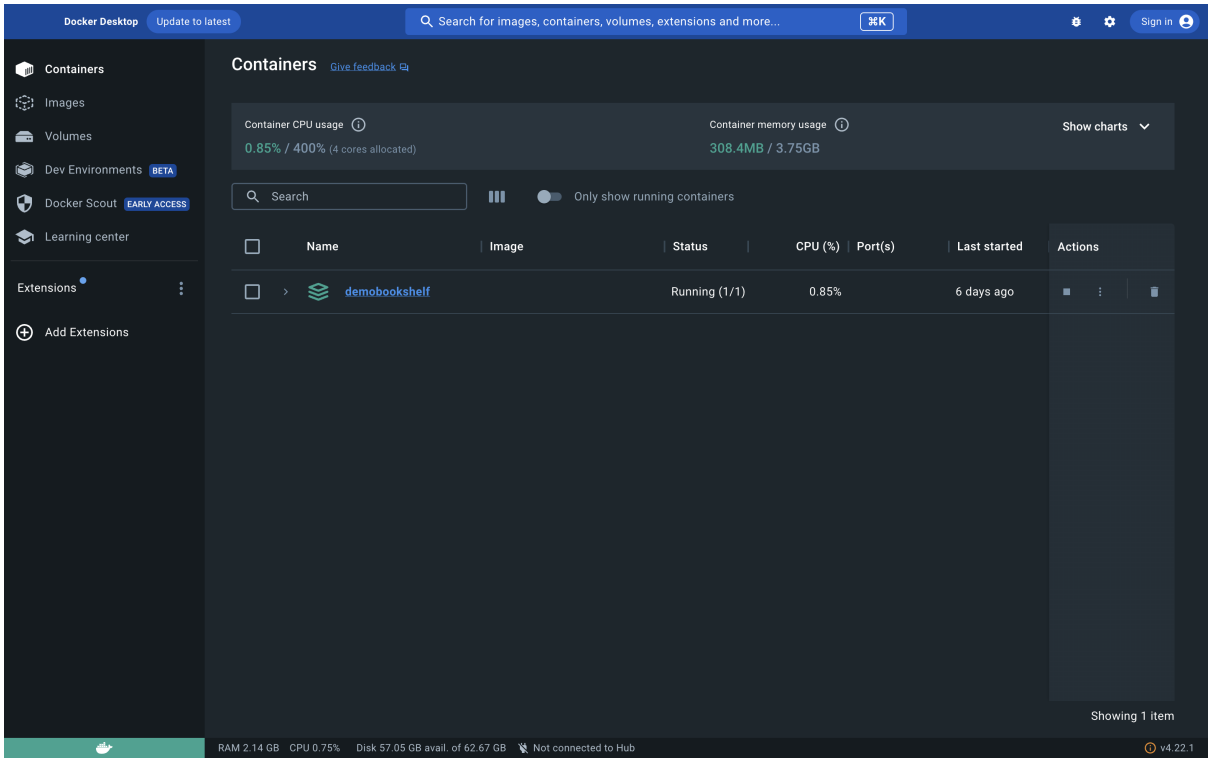


Figura 5.7: Docker dashboard con container in esecuzione

5.4.5 JPA - Hibernate

Java Persistence API (JPA) è un'interfaccia di programmazione che consente agli sviluppatori di lavorare con i database utilizzando Java. Viene usato per archiviare, recuperare e aggiornare i dati in un database. JPA si basa sulla tecnologia ORM (Object-Relational Mapping), può essere utilizzato per eseguire operazioni CRUD (creare, leggere, aggiornare ed eliminare) sui dati e fornisce una serie di API per interrogarli. JPA può essere implementato da vari framework ORM come Hibernate.[42] Hibernate è un ecosistema di librerie, la principale è Hibernate ORM, si tratta di un framework Java per mappare modelli di dominio orientati agli oggetti su un database relazionale. Sostanzialmente Hibernate viene usato per rendere persistenti i dati dall'ambiente Java al database. Hibernate implementa le specifiche JPA per la *persistenza dei dati*[3].[44]

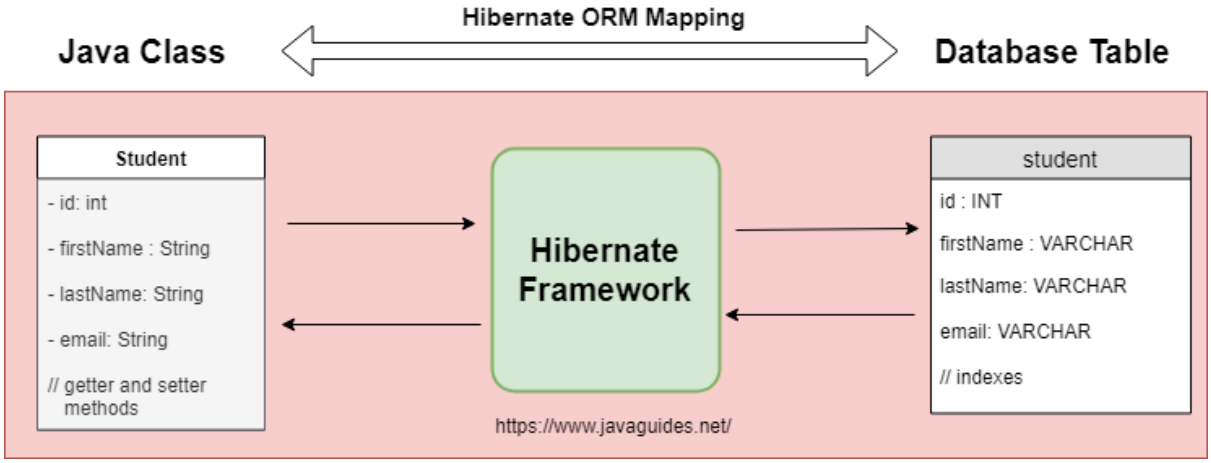


Figura 5.8: Funzionamento di Hibernate
[45]

ORM

L'Object-relational mapping (ORM) è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti (OOP) con sistemi RDBMS (Relational Database Management System). Un prodotto ORM fornisce, mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.[46]

[3] In informatica, il concetto di persistenza si riferisce alla caratteristica dei dati di un programma di sopravvivere all'esecuzione del programma stesso che li ha creati: senza questa capacità questi infatti verrebbero salvati solo in memoria Ram venendo dunque persi allo spegnimento del computer.[43]

5.4.6 RClone

RClone è un programma open source di sincronizzazione di file e gestione di directory che consente di gestire facilmente il backup e la sincronizzazione dei file tra il proprio dispositivo e una vasta gamma di servizi cloud, tra cui Google Drive, Amazon S3, Dropbox, Microsoft OneDrive e molti altri. RClone è noto per la sua flessibilità, efficienza e la capacità di eseguire complesse operazioni di gestione dei file attraverso una semplice interfaccia a riga di comando.[47]

```
1 rclone ls remote:path # lists a remote
2 rclone copy /local/path remote:path # copies /local/path to the
    remote
3 rclone sync --interactive /local/path remote:path # syncs /local/
    path to the remote
```

Listing 5.4: Esempio di comandi RClone.

5.4.7 BCrypt

Bcrypt è una *funzione hash crittografica*[4] progettata per l'hashing delle password e per la loro conservazione in sicurezza all'interno delle applicazioni, così da renderle meno vulnerabili agli attacchi informatici basati su dizionari. La funzione è stata concepita basandosi sul complesso algoritmo Blowfish cipher. Bcrypt esegue un elaborato processo di hashing, durante il quale la password di un utente viene trasformata in una serie di caratteri di lunghezza fissa utilizzando una funzione hash unidirezionale; questo significa che una volta che la password è stata "hashata", non può essere ricondotta alla sua forma originale. Ogni volta che l'utente accede al suo account, bcrypt rielabora la password e confronta il nuovo valore hash con la versione memorizzata nel sistema per verificare se le password corrispondono. A differenza di altri algoritmi di hashing, bcrypt aggiunge un elemento casuale, chiamato "salt", alla password, creando un hash unico che è quasi impossibile da decifrare con tentativi automatici negli attacchi a dizionario hash e brute force.[49]

[4]Una funzione crittografica di hash è un algoritmo matematico che mappa dei dati di lunghezza arbitraria in una stringa binaria di dimensione fissa chiamata valore di hash, questa funzione di hash è progettata per essere unidirezionale, ovvero una funzione difficile da invertire.Una funzione crittografica di hash ideale deve avere alcune proprietà fondamentali: deve identificare univocamente il messaggio, deve essere deterministico, in modo che lo stesso messaggio si traduca sempre nello stesso hash, deve essere semplice e veloce calcolare un valore hash da un qualunque tipo di dato e deve essere molto difficile o quasi impossibile generare un messaggio dal suo valore hash se non provando tutti i messaggi possibili.[48]

Capitolo 6

Sviluppo Della Web App

In questo capitolo vedremo come è stata sviluppata l'applicazione tramite la quale è possibile sincronizzare e mantenere aggiornati i file su diverse piattaforme cloud, usando le tecnologie spiegate nel capitolo precedente, partendo dalla struttura del database fino ad arrivare allo sviluppo del front-end.

6.1 Database

Per prima cosa mi sono occupato di costruire un diagramma ER (Entità-Relazione) sulla base delle specifiche che mi sono state assegnate per aiutarmi in seguito nella creazione vera e propria delle tabelle nel database.

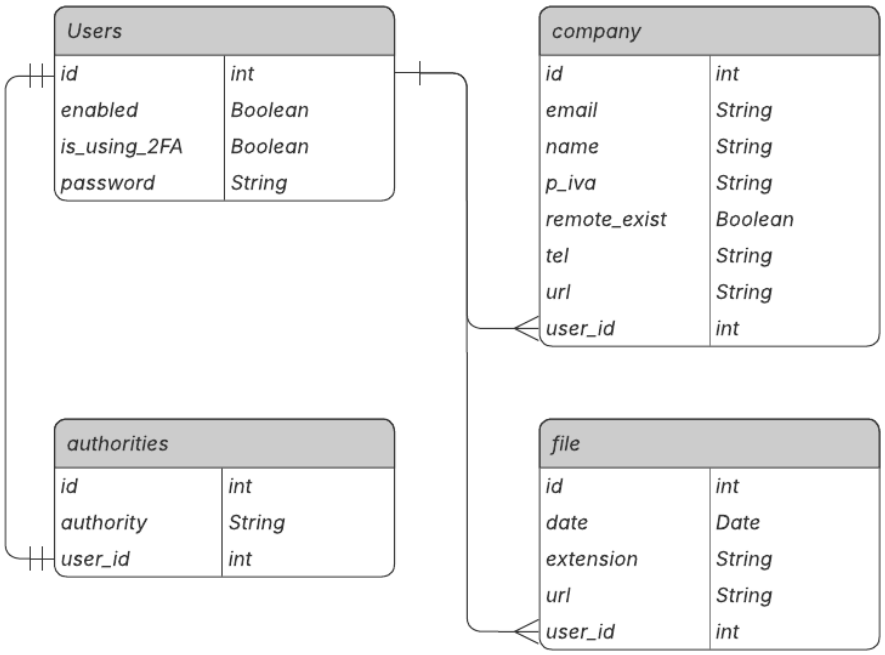


Figura 6.1: Diagramma ER per database

Dopodiché tramite il tool web Spring Initializr (riferimento alla figura 5.6) ho creato il mio progetto Spring importando le dipendenze necessarie per lo sviluppo del progetto.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.15</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.bookshelf2</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>demo</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>11</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.32</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webflux</artifactId>
    </dependency>

    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.10.1</version>
    </dependency>
    <dependency>
      <groupId>io.projectreactor.netty</groupId>
      <artifactId>reactor-netty-http</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.thymeleaf</groupId>
      <artifactId>thymeleaf-spring5</artifactId>
    </dependency>



    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <dependency>
      <groupId>org.thymeleaf.extras</groupId>
      <artifactId>thymeleaf-extras-springsecurity5</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-taglibs</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>org.hibernate.validator</groupId>
      <artifactId>hibernate-validator</artifactId>
```

```
88      </dependency>
89
90      <dependency>
91        <groupId>javax.mail</groupId>
92        <artifactId>mail</artifactId>
93        <version>1.4.7</version>
94      </dependency>
95
96      <dependency>
97        <groupId>org.springframework.boot</groupId>
98        <artifactId>spring-boot-starter-mail</artifactId>
99      </dependency>
100
101     <dependency>
102       <groupId>org.springframework</groupId>
103       <artifactId>spring-messaging</artifactId>
104     </dependency>
105
106     <dependency>
107       <groupId>org.springframework.boot</groupId>
108       <artifactId>spring-boot-starter-test</artifactId>
109       <scope>test</scope>
110     </dependency>
111
112     <dependency>
113       <groupId>org.jboss.aerogear</groupId>
114       <artifactId>aerogear-otp-java</artifactId>
115       <version>1.0.0</version>
116     </dependency>
117
118     <dependency>
119       <groupId>org.apache.tika</groupId>
120       <artifactId>tika-core</artifactId>
121       <version>0.7</version>
122     </dependency>
123   </dependencies>
124
125   <build>
126
127     <plugins>
128       <plugin>
129         <groupId>org.springframework.boot</groupId>
130         <artifactId>spring-boot-maven-plugin</artifactId>
131       </plugin>
132     </plugins>
133   </build>
134
135 </project>
```

Listing 6.1: pom.xml finale del progetto

A questo punto, avendo lo scheletro del progetto, ho creato il file *docker-compose.yml* per dockerizzare in un container l'immagine del db Mysql che ho utilizzato.

```
1  version: '3.8'
2
3  networks:
4    default:
5
6  services:
7    db:
8      platform: linux/x86_64
9      image: mysql:5.7
10     container_name: book
11     ports:
12       - 3306:3306
13     volumes:
14       - "./.data/db:/var/lib/mysql"
15     environment:
16       MYSQL_ROOT_PASSWORD: pass
17       MYSQL_DATABASE: book
```

Listing 6.2: File di configurazione docker-compose.yml.

Una volta creato il db e reso disponibile con docker, sono passato alla creazione delle classi delle mie entità in modo tale che JPA riuscisse a creare le rispettive tabelle nel database una volta configurato nelle *application.properties* del progetto.

```
spring.datasource.url=jdbc:mysql://localhost:3306/book
spring.datasource.username=root
spring.datasource.password=pass
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.
    MySQL5InnoDBDialect
```

Listing 6.3: Configurazione del db in application.properties.

6.2 Entità

Infatti con l'utilizzo di JPA, Hibernate e la configurazione *spring.jpa.hibernate.ddl-auto=create* le tabelle verranno create in automatico ad ogni avvio dell'applicazione grazie alle annotatzioni presenti nelle classi Java di ogni entità:

- @Entity: per definire la classe come entità da mappare su una tabella.
- @Table: per definire il nome della tabella.
- @Id, @GeneratedValue: per identificare la chiave primaria in modo autoincrementale.
- @OneToOne, @OneToMany: servono per identificare le eventuali relazioni tra tabelle per la gestione della chiave esterna.
- @JsonManagedReference: serve a risolvere i problemi di ricorsione infinita quando ci sono relazioni.
- @Transient: serve ad indicare che questo campo non andrà mappato nella tabella sul DB.

```
@Entity
@Table(name = "users")
@NamedQueries({@NamedQuery(name = User.USER_REPORT, query = User.USER_REPORT_JPQL)})
public class User {

    public static final String USER_REPORT = "User.userReport";
    public static final String USER_REPORT_JPQL = "select u.id,u.email,u.is_using_2FA,u.enabled from User u where u.email =
        ?1";

    @Transient
    private String secret;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotEmpty(message = "password cannot be empty.")
    @Column(name = "password")
    private String password;

    @Column(name = "is_using_2FA")
    private Boolean isUsing2FA = false;

    @Column(name = "enabled")
    private Boolean enabled = false;

    @JsonManagedReference
    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval=true,fetch = FetchType.LAZY)
    private List<Authorities> authorities = new ArrayList<>();

    @JsonManagedReference
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval=true,fetch = FetchType.LAZY)
```

```
32      private List<FileInfo> fileInfoList = new ArrayList<>();
33
34        @JsonManagedReference
35      @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval=true,fetch = FetchType.LAZY)
36      private List<Company> fileInfoList = new ArrayList<>();
37  }
```

Listing 6.4: Esempio classe user.

6.3 Repository e Service

In un'app Spring Boot strutturata con JPA è possibile riconoscere tre strati tipici:

- Repository: comunica col DB e ci permette di accedere ai dati tramite operazioni CRUD.
- Service: un'interfaccia che definisce le operazioni di business.
- ServiceImpl: implementa la logica di business sfruttando i repository.

6.3.1 Repository

Il repository, che solitamente è un'interfaccia utilizzata dai nostri service, è la parte che si occupa di comunicare con il database eseguendo le operazioni richieste dai service. Per identificare una classe o un'interfaccia come repository basta utilizzare semplicemente l'annotazione *@Repository*, inoltre grazie all'uso di JPA è possibile estenderla con la classe *JpaRepository<Entità,Tipo>* che fornisce in automatico tutte le operazioni CRUD senza la necessità di doverle implementare. Tra le parentesi angolate basterà specificare la classe dell'entità a cui ci stiamo riferendo e il tipo del suo id, di seguito un esempio del repository per la classe user:

```
1
2 @Repository
3 public interface UserRepository extends JpaRepository<User, Long> {
4
5     User userReport(String email);
6
7 }
```

Listing 6.5: Esempio repository user.

6.3.2 Service

In mezzo alla comunicazione tra controller e repository vi è l'interfaccia Service che si occupa di definire i metodi che vengono chiamati dai controller in base alle esigenze, di seguito un esempio del Service per la classe user:

```
1
2 public interface UserService {
3
4     User save(User user);
5
6     String findByEmail(String email);
7
8 }
```

Listing 6.6: Esempio repository user.

6.3.3 ServiceImpl

Infine abbiamo la classe ServiceImpl, creata con l'utilizzo dell'annotazione @Service, questa è l'implementazione vera e propria dei metodi sfruttando il nostro Repository iniettato come dipendenza tramite annotazione @Autowired, di seguito un esempio di ServiceImpl per la classe user:

```
1
2 @Service
3 public class UserServiceImpl implements UserService {
4
5     @Autowired
6     private UserRepository userRepository;
7
8     @Override
9     @Transactional
10    public User save(User user){
11        return userRepository.save(user);
12    }
13
14    @Override
15    public String findByEmail(String email) {
16        return userRepository.userReport(email);
17    }
18 }
```

Listing 6.7: Esempio ServiceImpl user.

Qui possiamo notare due metodi:

- save: è il metodo che ci permette di salvare un nuovo user in fase di registrazione richiamando direttamente uno dei metodi del repository.
- findByEmail: è un metodo che richiama la query custom (userReport) che abbiamo creato nella classe User che ci permette di ottenere tutti i dati di un singolo user cercandolo con la sua email univoca.

6.4 Controller

Un controller in un'applicazione web è il componente che si occupa di gestire le richieste dell'utente senza però occuparsi della logica di business. Può gestire richieste di URL, parametri, body HTTP (GET,POST,PUT,DELETE), mandare una risposta all'utente che può essere una pagina HTML o un file JSON oppure richiamare uno dei metodi del Service. In questo capitolo vedremo i controller principali dell'applicazione che permettono di eseguire le funzioni principali per le quali l'app è stata pensata. Quasi ogni controller di questa web app è annotato con @RestController, questo serve a specificare che il controller viene utilizzato per creare servizi Web che restituiscono dati JSON o XML ed è un controller con @ResponseBody incorporato.

6.4.1 Registration controller

Questo controller è composto da due metodi principali:

- registrazione: questo metodo è denotato da *@GetMapping*; questa notazione serve per gestire le richieste del browser. In questo caso, se nell'url è presente l'endpoint "/registration", il controller restituisce come risposta una stringa che verrà poi letta da Angular per mostrare la pagina corretta, in questo caso la pagina di registrazione.

- addUser: questo metodo è denotato da *@PostMapping*. Questa notazione serve per gestire richieste di tipo POST, quindi quando il nostro front-end invia dati al nostro back-end. In questo caso, infatti, il metodo riceve in input i parametri di registrazione di un nuovo utente: email, password e la conferma di quest'ultima. Ogni utente registrato ha il ruolo "USER" e solamente da DB è possibile assegnarsi il ruolo "ADMIN". Quando i parametri in input superano tutti i controlli, la password viene criptata tramite *user.setPassword(passwordEncoder.encode(user.getPassword()));* e l'utente viene registrato. Successivamente viene creato un evento *OnCreateAccountEvent* che vedremo in seguito.

```
1
2  @RestController
3  public class RegistrationController {
4
5      @Autowired
6      private UserService userService;
7
8      @Autowired
9      private AuthoritiesService authoritiesService;
10
11     @Autowired
12     private PasswordEncoder passwordEncoder;
13
14     @Autowired
15     private ApplicationEventPublisher applicationEventPublisher;
16
17     @Autowired
18     UserServiceImpl userServiceImpl;
19
20     @GetMapping("registration")
21     public String registrazione(){
22         return "registrazione";
23     }
24
25
26     @PostMapping("registration" )
27     public String addUser(@Valid @RequestPart("email") String email,
28                           @Valid @RequestPart("password") String password,
29                           @Valid @RequestPart("rePassword") String rePassword,
30                           BindingResult result, Map<String,Object> model,
31                           HttpServletRequest request) throws Exception {
```

```
32
33          User user = new User();
34          user.setEmail(email);
35          user.setPassword(password);
36          user.setMatchingPassword(rePassword);
37          String pass = user.getPassword();
38          String mPass = user.getMatchingPassword();
39
40          if(result.hasErrors()){
41              String string = "no-blank";
42
43              ObjectMapper objectMapper = new ObjectMapper();
44
45              String json = objectMapper.writeValueAsString(string);
46
47              System.out.println(json);
48              return json;
49          }
50          if(pass.length() < 8 && !pass.matches(".*[A-Z].*")) {
51              //model.put("invalidEmail", "La password deve essere lunga almeno 8 caratteri e contenere una lettera maiuscola")
                    ;
52              String string = "invalid-password";
53
54              ObjectMapper objectMapper = new ObjectMapper();
55
56              String json = objectMapper.writeValueAsString(string);
57
58              System.out.println(json);
59              return json;
60          }
61          else if(!pass.equals(mPass)) {
62              //model.put("invalidEmail","Password mismatch");
63              String string = "password-mismatch";
64
65              ObjectMapper objectMapper = new ObjectMapper();
66
67              String json = objectMapper.writeValueAsString(string);
68
69              System.out.println(json);
70              return json;
71          }else{
72              try{
73                  InternetAddress email = new InternetAddress(email);
74                  email.validate();
75              }catch(AddressException e){
76                  //model.put("invalidEmail","Format email not correct");
77                  String string = "invalid-email";
78
79                  ObjectMapper objectMapper = new ObjectMapper();
80
81                  String json = objectMapper.writeValueAsString(string);
82
83                  System.out.println(json);
84                  return json;
85              }
86              Authorities authorities = new Authorities(email, "ROLE_USER");
87              //user.getAuthorities().add(authorities); //unidirectional con joincolumn in user
88              user.addAuthorities(authorities); //bidirectional
89              user.setPassword(passwordEncoder.encode(user.getPassword()));
90              try {
91
92                  userService.save(user);
93              }catch (DataIntegrityViolationException e){
94                  System.out.println(e.getMessage());
95                  if(e.getMostSpecificCause().getMessage().contains("@")){
96                      //model.put("duplicateMail","Email already exist");
97                      String string = "duplicate-email";
98
99                      ObjectMapper objectMapper = new ObjectMapper();
100
101                     String json = objectMapper.writeValueAsString(string);
102
103                     System.out.println(json);
104                     return json;
105                 }
106
107             }
108         }
109         applicationEventPublisher.publishEvent(new OnCreateAccountEvent(user,request.getLocale(),"demo"));
110
111         String string = "registered";
112
113         ObjectMapper objectMapper = new ObjectMapper();
114
115         String json = objectMapper.writeValueAsString(string);
116
117         System.out.println(json);
118         return json;
119     }
120 }
```

Listing 6.8: RegistrationController.java.

6.4.2 Home controller

Questo controller semplicemente controlla, quando si accede alla pagina principale, che l'utente che sta provando ad accedervi è autenticato correttamente inviando un JSON di risposta che verrà gestito nel front-end da Angular.

```
1
2  @RestController
3  public class HomeController {
4
5      @Autowired
6      private UserService userService;
7
8      @GetMapping("")
9      public String home(Map<String,Object> model) throws JsonProcessingException {
10
11         SecurityContext context = SecurityContextHolder.getContext();
12         Authentication authentication = context.getAuthentication();
13         System.out.println(authentication);
14         Object principal = authentication.getPrincipal();
15
16         if(principal == "anonymousUser"){
17             model.put("role",authentication.getAuthorities().toString());
18         }else {
19             String email = authentication.getEmail();
20             model.put("email",email);
21             model.put("role",authentication.getAuthorities().toString());
22
23         }
24         String string = "";
25         if (authentication.isAuthenticated()) {
26              string = "auth";
27         }else{
28              string = "no-auth";
29         }
30
31
32         ObjectMapper objectMapper = new ObjectMapper();
33
34         String json = objectMapper.writeValueAsString(string);
35
36         System.out.println(json);
37
38         return json;
39     }
40 }
```

Listing 6.9: HomeController.java.

6.4.3 Create company controller

Anche questo controller, come il registration che abbiamo visto precedentemente, tramite il metodo *addCompany* si occupa semplicemente di ricevere i dati relativi all'azienda dal front-end, controllare eventuali errori e, in caso non ci fossero, aggiungere l'azienda nella relativa tabella sul DB.

```
1
2  @RestController
3  public class CreateCompanyController {
4
5      @Autowired
6      CompanyService companyService;
7
8      @PostMapping("/createCompany")
9      public String addCompany(@Valid @RequestPart("name") String name,
10                              @RequestPart("piva") String piva,
11                              @RequestPart("telefono") String telefono,
12                              @RequestPart("email") String email,
13                              @RequestPart("url") String url,
14                              BindingResult result, Map<String, Object> model,
15                              HttpServletRequest request) throws Exception {
16          Company company = new Company();
17          company.setName(name);
18          company.setpIva(piva);
19          company.setTel(telefono);
20          company.setEmail(email);
21          company.setUrl(url);
22          if (name.equals("null")) {
23              String string = "no-blank";
24
25              ObjectMapper objectMapper = new ObjectMapper();
26
27              String json = objectMapper.writeValueAsString(string);
28
29              return json;
30          }
31          if (!piva.isEmpty() && (piva.length() != 11)) {
32              String string = "invalid-piva";
33
34              ObjectMapper objectMapper = new ObjectMapper();
35
36              String json = objectMapper.writeValueAsString(string);
37
38              return json;
39          }
40
41          if(!email.equals("null")){
42              try{
43                  InternetAddress vemail = new InternetAddress(email);
44                  vemail.validate();
45              }catch(AddressException e){
46                  String string = "invalid-email";
47
48                  ObjectMapper objectMapper = new ObjectMapper();
49
50                  String json = objectMapper.writeValueAsString(string);
51
52                  return json;
53              }
54          }
55          try {
56              companyService.save(company);
57          } catch (DataIntegrityViolationException e) {
58
59                  String string = "duplicate-piva";
60
61                  ObjectMapper objectMapper = new ObjectMapper();
62
63                  String json = objectMapper.writeValueAsString(string);
64
65                  return json;
66              }
67          String string = "add-company";
68
69          ObjectMapper objectMapper = new ObjectMapper();
70
71          String json = objectMapper.writeValueAsString(string);
72
73          return json;
74          }
75      }
```

Listing 6.10: CreateCompanyController.java.

6.4.4 Create remote controller

Questo controller è fondamentale perché permette di eseguire il compito principale dell'app: quello di poter sincronizzare i file locali con una piattaforma cloud. Nello specifico questo controller permette di creare un remote (fondamentalmente una directory) per ogni azienda sulla piattaforma cloud Google Drive tramite il tool RClone. Il metodo createRemote prende in input dall'url tramite *@RequestParam* l'id dell'azienda per la quale si vuole creare un remote, recupera tramite quest'ultimo il nome dell'azienda dal DB che verrà usato come nome per la directory, ed esegue una serie di comandi RClone che, essendo un tool a riga di comando, devono essere eseguiti su una console. Per questo ho creato una classe util (che fornisce utilità), come viene detta in gergo, che permetta di eseguire i comandi che vengono mandati in input su una console.

- rclone config create: permette di creare il remote con il nome dell'azienda che viene passato.
- rclone authorize drive: apre una pagina internet che permette di autenticarsi con l'account aziendale su Google Drive.
- rclone config update: termina la configurazione impostando le variabili necessarie.
- rclone lsd: stampa il risultato, mostrando se la cartella sia stata creata o meno.

La classe è *"RcloneCommandExecutor"* e vedremo la sua implementazione in seguito.

```
1
2  @RestController
3  public class createRemoteController {
4      @Autowired
5      private RcloneCommandExecutor commandExecutor;
6
7      @Autowired
8      private CompanyService companyService;
9
10     @PostMapping("createRemote")
11     public String createRemote(@RequestParam("name") String idStr){
12         Company company = new Company();
13         Long id = Long.valueOf(idStr);
14         company = companyService.findById(id);
15         String name = company.getName();
16         System.out.println(name);
17         String codice="";
18         try {
19             String command = "rclone config create "+ name + " drive config_is_local=false";
20             String result = commandExecutor.getCommandOutput(command);
21             System.out.println("COMMAND " + result);
22         } catch (IOException | InterruptedException e) {
23             e.printStackTrace();
24         }
25         try {
26             String command = "rclone authorize drive";
27             String result = commandExecutor.getCommandOutput(command);
28             System.out.println("COMMAND " + result);
29             String[] arrOfStr = result.split("---");
30             result = arrOfStr[1];
31             arrOfStr = result.split(">");
32             result = arrOfStr[1];
33             result = result.replaceAll("<","").trim();
34             System.out.println("CODICE: "+result);
35             codice = result;
36
37         } catch (IOException | InterruptedException e) {
38             e.printStackTrace();
39         }
40
41         try {
42             String command = "rclone config update "+name+" --continue --state *oauth-islocal,teamdrive,, --result true";
43             String result = commandExecutor.getCommandOutput(command);
44             System.out.println("COMMAND " + result);
45         } catch (IOException | InterruptedException e) {
```

```
46              e.printStackTrace();
47          }
48
49          try {
50              String command = "rclone lsd "+ name + ":";
51              String result = commandExecutor.getCommandOutput(command);
52              System.out.println("COMMAND " + result);
53          } catch (IOException | InterruptedException e) {
54              e.printStackTrace();
55          }
56          company.setRemoteExist(true);
57          companyService.save(company);
58          return "remoteCreated";
59      }
60  }
```

Listing 6.11: createRemoteController.java.

6.4.5 File controller

Anche questo controller è essenziale per il funzionamento dell'app, in quanto si occupa di gestire tutte le operazioni CRUD riguardanti i file, in particolare:

- getLoadFile: recupera i file dal DB e li salva in un model che verrà inviato al front-end per popolare la data-grid dedicata.
- addFile: permette di salvare i file caricati dal front-end in una cartella locale e ne salva i dati sul DB.
- owFile: permette di sovrascrivere un file già esistente in locale e aggiornarne i dati su DB.
- deleteFile: permette di eliminare i file in locale e i relativi dati su DB.

Infine, la funzione fondamentale che permette la sincronizzazione dei file locali con il cloud: *doBisync*. Questa funzione legge i cambiamenti tra le due tabelle, file locali e file su cloud, e le aggiorna in modo tale da sincronizzarle tramite il comando RClone: *rclone bisync*, eseguito sempre grazie alla nostra classe *"RcloneCommandExecutor"*.

```
1
2   @RestController
3   public class FileController {
4
5       @Autowired
6       private UserService userService;
7
8       @Autowired
9       private FileInfoService fileInfoService;
10
11      @Autowired
12      private RcloneCommandExecutor commandExecutor;
13
14      @Autowired
15      private FileService fileService;
16
17      private final Path root = Paths.get("/Users/lc-service/Documents/demoBookShelf/src/main/webapp/WEB-INF/files");
18      private final Path path = Paths.get("/Users/lc-service/Documents/demoBookShelf/src/main/webapp/WEB-INF/temp");
19
20      //Carica la pagina con i file
21      @GetMapping("/loadFile")
22      public String getLoadFile(Model model, HttpServletRequest request) throws IOException {
23          List<FileInfo> fileList;
24          fileList = fileInfoService.findAllFile();
25          model.addAttribute("files", fileList);
26          return "loadFile";
27      }
28      //Upload di un file
29      @PostMapping("/loadFile")
30      @CrossOrigin(origins = "http://localhost:4200",methods = RequestMethod.POST)
31      public String addFile(@RequestParam(value = "file",required = false) MultipartFile file, Map<String, Object> model)
            throws IOException {
32          if (file.getOriginalFilename().contains(" ")) {
33              String string = "file-manager-invalid";
34              ObjectMapper objectMapper = new ObjectMapper();
35              String json = objectMapper.writeValueAsString(string);
36              return json;
37          }
38          SecurityContext context = SecurityContextHolder.getContext();
39          Authentication authentication = context.getAuthentication();
40          String username = authentication.getName();
41          User user = userService.findUser(username);
42          String filename = file.getOriginalFilename().toString();
43          String exstension = "";
44          int lastDotIndex = filename.lastIndexOf('.');
45          if (lastDotIndex > 0) {
46              exstension = filename.substring(lastDotIndex + 1);
47          }
48          LocalDateTime currentDateTime = LocalDateTime.now();
49
50          DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
51          String dateModified = currentDateTime.format(formatter);
52
53          FileInfo fileInfo = new FileInfo();
54          fileInfo.setUrl(root.toString());
55          fileInfo.setName(filename);
```

```
56          fileInfo.setDate(dateModified);
57          fileInfo.setExstension(exstension);
58          fileInfo.setUser(user);
59
60          try {
61              fileService.save(file);
62              fileInfoService.save(fileInfo);
63          } catch (Exception e) {
64
65              fileService.save(file, path);
66              String string = "file-manager-ow";
67              ObjectMapper objectMapper = new ObjectMapper();
68              String json = objectMapper.writeValueAsString(string);
69              return json;
70          }
71
72          String string = "file-manager";
73
74          ObjectMapper objectMapper = new ObjectMapper();
75
76          String json = objectMapper.writeValueAsString(string);
77
78          System.out.println(json);
79          return json;
80      }
81
82      //sovrascrivi il file
83      @GetMapping("overwrite/{ow}")
84      public String owFile(@PathVariable(value = "ow") boolean ow) throws IOException {
85          System.out.println("overwrite");
86          if (!ow) {
87              fileService.deleteAll(path);
88              String string = "file-manager";
89
90              ObjectMapper objectMapper = new ObjectMapper();
91
92              String json = objectMapper.writeValueAsString(string);
93
94              System.out.println(json);
95              return json;
96          } else {
97
98              File tempFile = new File("/Users/baalza/Desktop/demoBookShelf/src/main/webapp/WEB-INF/temp");
99              File[] files = tempFile.listFiles();
100             String filename = files[0].getName();
101             fileService.delete(filename);
102             LocalDateTime currentDateTime = LocalDateTime.now();
103             DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
104             String dateModified = currentDateTime.format(formatter);
105
106             FileInfo fileInfo = fileInfoService.findFile(filename);
107             fileInfo.setDate(dateModified);
108             fileInfoService.save(fileInfo);
109             Path sourceFilePath = Path.of("/Users/baalza/Desktop/demoBookShelf/src/main/webapp/WEB-INF/temp", filename);
110
111             try {
112                 // Sposta il file nella directory di destinazione
113                 Files.copy(sourceFilePath, root.resolve(filename), StandardCopyOption.REPLACE_EXISTING);
114                 fileService.deleteAll(path);
115                 /*String command = "rclone copy " + "/Users/baalza/Desktop/demoBookShelf/src/main/webapp/WEB-INF/files/" +
                        filename + " BookShelfRemote:FileBookShelf";
116                 String result = commandExecutor.getCommandOutput(command);
117                 System.out.println("COMMAND " + result);*/
118                 System.out.println("File spostato con successo.");
119             } catch (IOException e) {
120                 System.err.println("Errore durante lo spostamento del file: " + e.getMessage());
121             }
122             String string = "file-manager-ow-true";
123
124             ObjectMapper objectMapper = new ObjectMapper();
125
126             String json = objectMapper.writeValueAsString(string);
127
128             System.out.println(json);
129             return json;
130         }
131     }
132
133     //delete di un file
134     @GetMapping("deleteFile/{fileName}")
135     public String deleteFile(@PathVariable(value = "fileName",required = false) String fileName) throws
            JsonProcessingException {
136
137         fileInfoService.delete(fileName);
138         fileService.delete(fileName);
139         String string = "file-manager";
140
141         ObjectMapper objectMapper = new ObjectMapper();
142
143         String json = objectMapper.writeValueAsString(string);
144
145         return json;
146
147     }
148
```

```
149     @GetMapping("/execute")
150     public String executeCommand(Model model) {
151         try {
152             String command = "rclone lsd BookShelfRemote:";
153             String result = commandExecutor.getCommandOutput(command);
154             System.out.println("COMMAND " + result);
155             model.addAttribute("output", result);
156         } catch (IOException | InterruptedException e) {
157             e.printStackTrace();
158             model.addAttribute("error", "Errore durante l'esecuzione del comando.");
159         }
160         return "index";
161     }
162
163     @GetMapping("bisync")
164     public String doBisync(@RequestParam(name = "ow",required = false) boolean ow,@RequestParam(name = "add",required = false
            ) boolean add,@RequestParam(name = "delete",required = false) boolean delete) throws JsonProcessingException {
165         if(!ow && !add && !delete){
166             List<FileInfo> list = fileInfoService.findAllFile();
167             if(!list.isEmpty()) {
168                 try {
169                     String command = "rclone bisync " + root + " BookShelfRemote:FileBookShelf --delete-before --force --
                            verbose";
170                     String result = commandExecutor.getCommandOutput(command);
171                     System.out.println("COMMAND " + result);
172
173                 } catch (IOException | InterruptedException e) {
174                     e.printStackTrace();
175                 }
176                 try {
177                     String command = "rclone bisync " + root + " BookShelfRemote:FileBookShelf --resync --verbose";
178                     String result = commandExecutor.getCommandOutput(command);
179                     System.out.println("COMMAND " + result);
180
181                 } catch (IOException | InterruptedException e) {
182                     e.printStackTrace();
183                 }
184             }else{
185                 try {
186                     String command = "rclone delete BookShelfRemote:FileBookShelf --verbose";
187                     String result = commandExecutor.getCommandOutput(command);
188                     System.out.println("COMMAND " + result);
189                 } catch (IOException | InterruptedException e) {
190                     e.printStackTrace();
191                 }
192             }
193         }else if(ow || add){
194             try {
195                 String command = "rclone bisync /Users/baalza/Desktop/demoBookShelf/src/main/webapp/WEB-INF/files
                        BookShelfRemote:FileBookShelf --resync --verbose";
196                 String result = commandExecutor.getCommandOutput(command);
197                 System.out.println("COMMAND " + result);
198             } catch (IOException | InterruptedException e) {
199                 e.printStackTrace();
200             }
201         }else if(delete){
202             List<FileInfo> list = fileInfoService.findAllFile();
203             if(!list.isEmpty()) {
204                 try {
205                     String command = "rclone bisync " + root + " BookShelfRemote:FileBookShelf --delete-before --force --
                            verbose";
206                     String result = commandExecutor.getCommandOutput(command);
207                     System.out.println("COMMAND " + result);
208
209                 } catch (IOException | InterruptedException e) {
210                     e.printStackTrace();
211                 }
212             }else{
213                 try {
214                     String command = "rclone delete BookShelfRemote:FileBookShelf --verbose";
215                     String result = commandExecutor.getCommandOutput(command);
216                     System.out.println("COMMAND " + result);
217                 } catch (IOException | InterruptedException e) {
218                     e.printStackTrace();
219                 }
220             }
221         }
222
223         String string = "file-manager";
224
225         ObjectMapper objectMapper = new ObjectMapper();
226
227         String json = objectMapper.writeValueAsString(string);
228
229         System.out.println(json);
230         return json;
231     }
232 }
```

Listing 6.12: FileController.java.

6.4.6 Rest file controller

Questo controller è mappato su due endpoint che restituiscono rispettivamente i file locali e i file presenti su Google Drive.

```
1
2  @RestController
3  public class restFileController {
4
5      @Autowired
6      private FileInfoService fileInfoService;
7
8      @Autowired
9      private RcloneCommandExecutor commandExecutor;
10
11     @GetMapping("restFiles")
12     @CrossOrigin(origins = "http://localhost:4200")
13     public List<FileInfo> restFiles() {
14         List<FileInfo> files = fileInfoService.findAllFile();
15         Collections.reverse(files);
16         return files;
17     }
18
19     @GetMapping("restFiles-drive")
20     @CrossOrigin(origins = "http://localhost:4200")
21     public List<FileInfo> restFilesDrive() throws IOException, InterruptedException {
22         List <FileInfo> fileList = new ArrayList<>();
23         String command = "rclone lsjson BookShelfRemote:FileBookShelf";
24         String result = commandExecutor.getCommandOutput(command);
25         JsonArray jsonArray = new Gson().fromJson(result, JsonArray.class);
26         for (int i = 0; i < jsonArray.size(); i++) {
27             JsonObject jsonObject = jsonArray.get(i).getAsJsonObject();
28             String name = jsonObject.get("Name").getAsString();
29             String modTime = jsonObject.get("ModTime").getAsString();
30             DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
31             String cleanedDateString = modTime.substring(0, modTime.indexOf('.'));
32             cleanedDateString = cleanedDateString.replace("T"," ");
33             LocalDateTime localDateTime = LocalDateTime.parse(cleanedDateString, formatter);
34             Duration duration = Duration.ofHours(2);
35             LocalDateTime newDateTime = localDateTime.plus(duration);
36             String dateModified = newDateTime.format(formatter);
37             FileInfo file = new FileInfo(name,dateModified);
38             fileList.add(file);
39             System.out.println("Name: " + name);
40             System.out.println("ModTime: " + dateModified);
41             System.out.println("Formatted Date: "+dateModified);
42         }
43         return fileList;
44     }
45 }
```

Listing 6.13: GenerateQRcodeController.java.

6.4.7 Generate QR-code controller

Questo controller permette di abilitare l'autenticazione 2FA a discrezione dell'utente. Si compone di due metodi:

- generateQRUrl: questo metodo partendo da un URL base di quickchart.io per qr-code costruisce l'url che permette di essere scansionata con Google Authenticator e impostare il 2FA. L'url base viene combinata con una stringa segreta generata per ogni utente al momento del processo di abilitazione della 2FA.
- getQrcode: questo metodo controlla che l'utente sia autenticato, in caso positivo aggiorna il campo isUsing2FA, se true, lo aggiorna e richiama il metodo *generateQRUrl* per generare l'url che viene passata al front-end, il quale mostrerà il QR code da scansionare all'utente.

```
1
2  @RestController
3  public class GenerateQRcodeController {
4
5      @Autowired
6      private ApplicationContext applicationContext;
7
8      @Autowired
9      private UserService userService;
10     public static String QR_PREFIX =
11             "https://quickchart.io/chart?chs=200x200&chld=M%7C0&cht=qr&chl=";
12     @Value("${spring.application.name}")
13     public String APP_NAME;
14
15     public String generateQRUrl(User user) throws UnsupportedEncodingException {
16
17         String SK = user.getSecret().toUpperCase().toString().trim();
18         return    QR_PREFIX +URLEncoder.encode(String.format(
19                         "otpauth://totp/%s:%s?secret=%s&issuer=%s",
20                         APP_NAME, user.getUsername(), SK, APP_NAME),
21                 "UTF-8");
22     }
23
24     @GetMapping("qrcode")
25     @CrossOrigin(origins = "http://localhost:4200")
26     public String getQrcode(Map<String, Object> model) throws UnsupportedEncodingException, JsonProcessingException {
27         SecurityContext context = SecurityContextHolder.getContext();
28         Authentication authentication = context.getAuthentication();
29         Collection<? extends GrantedAuthority> authorities=authentication.getAuthorities();
30         String username = authentication.getName();
31         User user = userService.findUser(username);
32         user.setUsing2FA(true);
33         Authentication auth = new UsernamePasswordAuthenticationToken(
34                 user, user.getPassword(), authorities);
35         SecurityContextHolder.getContext().setAuthentication(auth);
36
37         userService.save(user);
38         String QRurl = generateQRUrl(user);
39
40         ObjectMapper objectMapper = new ObjectMapper();
41
42         String json = objectMapper.writeValueAsString(QRurl);
43
44         System.out.println(json);
45
46         return json;
47     }
48 }
```

Listing 6.14: GenerateQRcodeController.java.

6.5 Configurazioni

In questa sezione andremo ad analizzare le varie configurazioni implementate per la gestione della sicurezza al fine di gestire l'autenticazione degli utenti utilizzando Spring Boot Security, gestire la parte di invio e-mail di conferma ogni volta che un utente viene registrato, implementare il 2FA login quando abilitato e creare un esecutore di comandi che permetta di eseguire i comandi RClone su terminale in totale autonomia.

6.5.1 Configurazione login e 2FA

Questa classe implementa Spring Security per poter gestire l'autenticazione dell'utente. È stata annotata con *@Configuration* per indicare che è una classe di configurazione e *@EnableWeb-Security* per abilitare la configurazione di sicurezza web di Spring Security. In particolare nel metodo *"configure"* possiamo notare alcuni elementi importanti:

- antMatchers: servono a stabilire le regole di accesso a determinati endpoint, ad esempio in base al ruolo di un utente autenticato.
- loginPage e loginProcessingUrl: servono a stabilire quale pagina e quale URL saranno dedicate alla login, operazione gestita da Spring Security insieme alla logout.
- addFilterAfter: aggiunge un ulteriore filtro successivo alla login per verificare se l'utente ha abilitato il 2FA che in caso positivo verrà gestito da *"twoFactorAuthenticationProvider"*.
- passwordEncoder: Questo metodo utilizza BCrypt per l'hashing delle password.

```
1
2  @Configuration
3  @EnableWebSecurity
4  @ComponentScan("com.bookshelf2.demo")
5  public class SecurityConfig extends WebSecurityConfigurerAdapter {
6
7      @Autowired
8      private DataSource dataSource;
9
10     @Autowired
11     private BookshelfUserDetailsContextMapper ctxMapper;
12
13     @Autowired
14     AuthenticationProvider twoFactorAuthenticationProvider;
15
16     @Autowired
17     UserService userService;
18
19     @Override
20     @CrossOrigin(origins = "http://localhost:4200", methods = RequestMethod.POST)
21     protected void configure(final HttpSecurity http) throws Exception {
22
23         http
24                 .cors().and()
25                 .csrf(AbstractHttpConfigurer::disable)
26                 .sessionManagement()
27                 .sessionFixation()
28                 .migrateSession()
29                 .and()
30                 .authorizeRequests()
31                 .antMatchers("/anonymus*").anonymous() //role anonymus
32                 .antMatchers("/login*").permitAll()
33                 .antMatchers("/loadFile").permitAll()
34                 .antMatchers("/restFiles").authenticated()
35                 .antMatchers("/createRemote").hasRole("ADMIN")
36                 .antMatchers("/createCompany").hasRole("ADMIN")
37                 .antMatchers("/oauth2/authorization/google").permitAll()
38                 .antMatchers("/login/oauth2/code/google").permitAll()*/
39                 .antMatchers("/api/**").authenticated()
40                 .antMatchers("/registration*").permitAll()
41                 .antMatchers("/static/**").permitAll() //resources
```

```
42                  .antMatchers("/addAuthors").hasRole("USER")
43                  .antMatchers("/addBooks").hasRole("USER")
44                  .antMatchers("/*").permitAll()
45                  .anyRequest().permitAll()
46                  .and()
47                  .formLogin()
48                  .loginPage("http://localhost:4200/demo/login")
49                  .loginProcessingUrl("/perform_login")
50
51                  .successHandler((request, response, authentication) -> {
52                      Cookie customCookie = new Cookie("Authenticated", "true");
53                      customCookie.setMaxAge(86400); // Durata in secondi
54                      customCookie.setPath("/demo"); // Imposta il percorso del cookie
55                      customCookie.setHttpOnly(false);
56                      customCookie.setDomain("localhost");
57                      response.setContentType("application/json");
58                      UserDetails userDetails = (UserDetails) authentication.getPrincipal();
59                      String role = authentication.getAuthorities().iterator().next().getAuthority();
60                      System.out.println("ruoli"+role);
61                      boolean enable = userService.findUser(userDetails.getUsername()).getUsing2FA();
62                      String email = userDetails.getUsername();
63                      Map<String, Object> responseData = new HashMap<>();
64                      responseData.put("error", false);
65                      responseData.put("2fa", enable);
66                      responseData.put("email",email);
67                      responseData.put("role",role);
68                      responseData.put("cookie", customCookie);
69                      //responseData.put("session",);
70                      response.getWriter().write(new ObjectMapper().writeValueAsString(responseData));
71                  })
72                  .failureHandler((request, response, exception) -> {
73                      response.setContentType("application/json");
74                      response.getWriter().write("{ \"error\": \"true\" }");
75                  })
76                  .permitAll()
77                  .and()
78                  .and()
79                  .rememberMe()
80                  .key("superSecretKey")
81                  .tokenValiditySeconds(18000) //5 ore
82                  .tokenRepository(tokenRepository())
83                  .and()
84                  .logout()
85                  .logoutSuccessUrl("/")
86                  .logoutSuccessHandler((request, response, authentication) -> {
87                      System.out.println("logout");
88                      Cookie customCookie = new Cookie("Authenticated", "true");
89                      customCookie.setMaxAge(86400); // Durata in secondi
90                      customCookie.setPath("/demo"); // Imposta il percorso del cookie
91                      customCookie.setHttpOnly(false);
92                      customCookie.setDomain("localhost");
93                      response.setContentType("application/json");
94                      Map<String, Object> responseData = new HashMap<>();
95                      responseData.put("error", false);
96                      responseData.put("cookie", customCookie);
97                      response.getWriter().write(new ObjectMapper().writeValueAsString(responseData));
98                  })
99                  .logoutRequestMatcher(new AntPathRequestMatcher("/perform_logout", "GET"))
100                 .invalidateHttpSession(true)
101                 .deleteCookies("JSESSIONID")
102                 .permitAll()
103                 .and()
104
105                 .addFilterAfter(new TwoFactorAuthenticationFilter("/2fa-login", userService, corsConfigurationSource()),
                        DefaultLoginPageGeneratingFilter.class)
106                 .authenticationProvider(twoFactorAuthenticationProvider); // Configura la 2FA
107
108     }
109
110     @Bean
111     public PersistentTokenRepository tokenRepository() {
112         JdbcTokenRepositoryImpl token = new JdbcTokenRepositoryImpl();
113         token.setDataSource(dataSource);
114         return token;
115     }
116
117     @Override
118     @CrossOrigin(origins = "http://localhost:4200", methods = RequestMethod.POST)
119     protected void configure(final AuthenticationManagerBuilder auth) throws Exception {
120
121         auth.jdbcAuthentication().dataSource(dataSource).passwordEncoder(passwordEncoder())
122                 .and().authenticationProvider(twoFactorAuthenticationProvider);
123     }
124
125     @Bean
126     public PasswordEncoder passwordEncoder() {
127         return new BCryptPasswordEncoder();
128     }
129 }
```

Listing 6.15: Classe SecurityConfig.java.

6.5.2 Configurazione 2FA

Questa classe si occupa di validare l'OTP generato dall'autenticatore e inserito dall'utente in fase di login con 2FA abilitata. Il metodo *"authenticate"* recupera l'OTP e la chiave segreta dell'utente, la chiave segreta viene utilizzata per creare l'oggetto *"TOTP"* che a sua volta servirà per validare il codice OTP.

```
1
2  @Component
3  public class TwoFactorAuthenticationProvider implements AuthenticationProvider {
4  private UserService userService;
5
6      public TwoFactorAuthenticationProvider(UserService userService) {
7          this.userService=userService;
8      }
9
10     @Override
11     public Authentication authenticate(Authentication authentication) throws AuthenticationException {
12         if (!(authentication instanceof TwoFactorAuthenticationToken)) {
13             return null;
14         }
15
16         TwoFactorAuthenticationToken twoFactorAuthentication = (TwoFactorAuthenticationToken) authentication;
17         String otpCode = twoFactorAuthentication.getOtpCode();
18         SecurityContext context = SecurityContextHolder.getContext();
19         Authentication auth = context.getAuthentication();
20         String username = auth.getName();
21         System.out.println("TOTP "+otpCode);
22         User user = userService.findUser(username);
23         String SK = user.getSecret().toString().toUpperCase().trim();
24         boolean isOtpValid = validateOtpCode(otpCode);
25         Totp totp = new Totp(SK);
26         System.out.println("SK: " + SK + " " + totp.verify(otpCode));
27         if (isOtpValid && totp.verify(otpCode)) {
28             System.out.println("Codice valido");
29             // Restituisci un'istanza di Authentication riuscita
30             return authentication;
31         } else {
32             System.out.println("Codice non valido");
33             authentication.setAuthenticated(false);
34             return authentication;
35
36         }
37     }
38
39     @Override
40     public boolean supports(Class<?> authentication) {
41         return TwoFactorAuthenticationToken.class.isAssignableFrom(authentication);
42     }
43
44     private boolean validateOtpCode(String otpCode) {
45         try {
46             Long.parseLong(otpCode);
47         } catch (NumberFormatException e) {
48             return false;
49         }
50         return true;
51     }
52 }
```

Listing 6.16: Classe TwoFactorAuthenticationProvider.java.

6.5.3 Mail Sender

Questa classe implementa l'interfaccia *ApplicationListener*, un componente che ascolta eventi pubblicati nell'ApplicationContext (contenitore Spring), nel nostro caso l'evento è: *OnCreateAccountEvent*, triggerato dalla registrazione di un nuovo utente come abbiamo visto precedentemente (ref 6.4.1). Qui sfruttiamo l'interfaccia *JavaMailSender* e la dipendenza *spring-boot-starter-mail* per popolare i campi di una vera e propria e-mail da inviare all'utente tramite il suo indirizzo, inserito in fase di registrazione e recuperato dall'evento creato.

```
1
2  @Component
3  public class UserListener implements ApplicationListener<OnCreateAccountEvent> {
4      private String serverUrl = "http://localhost:8080/";
5
6      @Autowired
7      private JavaMailSender mailSender;
8
9      @Autowired
10     private UserService service;
11
12     @Autowired
13     private MessageSource messages;
14
15     @Autowired
16     private UserService userService;
17
18     @Override
19     public void onApplicationEvent(OnCreateAccountEvent event) {
20         this.confirmRegistration(event);
21     }
22
23     private void confirmRegistration(OnCreateAccountEvent event) {
24         User user = event.getUser();
25         String token = UUID.randomUUID().toString();
26         VerificationToken verificationToken = new VerificationToken();
27         Date expiryDate;
28         expiryDate = verificationToken.calculateExpiryDate(verificationToken.EXPIRATION);
29         System.out.println("EXPIRATON "+expiryDate);
30         service.createVerificationToken(user, user.getUsername(), token, expiryDate);
31         String recipientAddress = user.getUsername();
32         String subject = "Registration Confirmation BookShelf";
33         String confirmationUrl = event.getAppUrl() + "/account-confirmation?token=" + token;
34         String message = "Please confirm:";
35         SimpleMailMessage email = new SimpleMailMessage();
36         email.setTo(recipientAddress);
37         email.setSubject(subject);
38         String htmlContent = message + "http://localhost:4200/" + confirmationUrl + "";
39         email.setText(htmlContent);
40         mailSender.send(email);
41     }
42 }
```

Listing 6.17: Classe UserListener.java.

6.5.4 Esecutore dei comandi RClone

Questa classe permette di eseguire comandi del tool RClone, un tool a riga di comando, direttamente sulla console. Prende in input il comando come stringa, lo adatta e lo esegue su console tramite *InputStreamReader*.

```
1
2  @Service
3  public class RcloneCommandExecutor {
4
5      public static String getCommandOutput(String command) throws IOException, InterruptedException {
6          ProcessBuilder processBuilder = new ProcessBuilder(command.split("\\s+"));
7
8          String strArr[];
9          strArr=command.split("\\s+");
10         for(int i =0; i<strArr.length;i++){
11             System.out.println(strArr[i]);
12         }
13
14         processBuilder.redirectErrorStream(true);
15         Process process = processBuilder.start();
16
17         BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
18         StringBuilder output = new StringBuilder();
19
20         String line;
21         while ((line = reader.readLine()) != null) {
22             output.append(line).append("\n");
23         }
24
25         int exitCode = process.waitFor();
26         if (exitCode == 0) {
27             return output.toString();
28         } else {
29             throw new IOException("Errore durante l'esecuzione del comando: " + command);
30         }
31     }
32 }
```

Listing 6.18: Classe RcloneCommandExecutor.java.

6.6 Front-end

Il front-end è stato sviluppato in Angular e, come precedentemente discusso, è qui che abbiamo gestito la possibilità di svolgere determinate funzioni in base al ruolo dell'utente autenticato. Questo è stato possibile grazie all'uso delle *Guard*, un servizio che implementa un'interfaccia speciale per decidere se permettere o meno la navigazione verso una rotta oppure mostrare o no determinati elementi. Di seguito un esempio della dashboard presente in homepage tramite la quale le funzioni *createCompany* e *createRemote* sono disponibili solamente per un utente ADMIN.

```
1
2  <footer>
3    <nav class="navbar navbar-expand-lg navbar-light bg-light sticky-bottom">
4      <div class="container-fluid">
5
6        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNavDropdown" aria-
             controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">
7          <span class="navbar-toggler-icon"></span>
8        </button>
9        <div class="collapse navbar-collapse" id="navbarNavDropdown">
10         <ul class="navbar-nav">
11           <li class="nav-item">
12             <a class="nav-link toasts active" aria-current="page" (click)="open(staticBackdrop)"> <fa-icon [icon]="faBars"></
                   fa-icon></a>
13           </li>
14         </ul>
15       </div>
16     </div>
17   </nav>
18 </footer>
19 <ng-template #staticBackdrop let-content>
20   <div class="offcanvas-header">
21     <h1 class="offcanvas-title" id="offcanvasBottomLabel"></h1>
22     <button type="button" class="btn-close" data-bs-dismiss="offcanvas" aria-label="Close" (click)="close()"></button>
23   </div>
24   <div class="offcanvas-body small">
25     <div class="container-fluid">
26       <div class="row users">
27         <h2>Utenti  <fa-icon class="user-canvas" [icon]="faUser"></fa-icon></h2>
28         <div class="col-auto mb-2 text-center">
29           <a class="nav-link canvas " aria-current="page" routerLink="/demo/registrazione">Registrazione</a>
30         </div>
31       </div>
32       <div class="row users pt-2">
33         <h2>Gestione  <fa-icon class="user-canvas" [icon]="faGear"></fa-icon></h2>
34         <div class="col-auto mb-2">
35           <a routerLink="/demo/company" *ngIf="role === 'ROLE_ADMIN'">
36             <button type="button" class="btn btn-primary">Aziende</button>
37           </a>
38         </div>
39         <div class="col-auto mb-2">
40           <a routerLink="/demo/create-remote" *ngIf="role === 'ROLE_ADMIN'">
41             <button type="button" class="btn btn-primary">Crea Remote</button>
42           </a>
43         </div>
44       </div>
45     </div>
46   </div>
47
48 </ng-template>
```

Listing 6.19: footer-nav.component.html.

Un'altra parte fondamentale della nostra applicazione riguarda l'utilizzo della *dataGrid*, un componente che serve a mostrare dati tabellari in modo interattivo, con funzionalità avanzate come ordinamento, paginazione, filtro, editing, ecc. Compongono la pagina principale della web app, nella quale sono presenti due dataGrid che mostrano rispettivamente i file archiviati in locale e i file archiviati su piattaforma cloud. Di seguito un esempio della tabella per la gestione dei file locali:

```

<ag-grid-angular
  style="width: 100%; height: 400px;"
  class="ag-theme-alpine"
  [columnDefs]="columnDefs"
  [defaultColDef]="defaultColDef"
  [rowData]="rowData$ | async"
  [rowSelection]="'multiple'"
  [animateRows]="true"
  (gridReady)="onGridReady($event)"
></ag-grid-angular>
```

Listing 6.20: data-grid-fs.component.html.

```

@Component({
  selector: 'bookshelf-data-grid-fs',
  templateUrl: './data-grid-fs.component.html',
  styleUrls: ['./data-grid-fs.component.css']
})
export class DataGridFsComponent {

  modalReference: NgbActiveModal | undefined;
  component = DeleteModalComponent;
  ngbModalOptions: NgbModalOptions = {
    backdrop: 'static',
    keyboard: false
  };
  public open(modal: any,field:any): void {
    const active = this.modalService.open(modal, this.ngbModalOptions);
    active.componentInstance.field = field;
  }
  public columnDefs: ColDef[] = [
    { headerName: "Filename",field: 'name', comparator:(valueA,valueB,nodeA,nodeB, isDescending)=> {
        if(valueA == valueB) return 0;
        return(valueA> valueB) ? 1 : -1;
      }},
    { headerName: "Date",field: 'date',comparator:(valueA,valueB,nodeA,nodeB, isDescending)=> {
        if(valueA == valueB) return 0;
        return(valueA> valueB) ? 1 : -1;
      }},
    { headerName: "Action",field: 'name', cellRenderer: TrashButtonComponent,cellRendererParams: {
        clicked: (field: any) => {
          this.open(this.component,field);
          //alert(`${field} was clicked`);
        }
      },
    }
  ];

  public defaultColDef: ColDef = {
    sortable: true,
    filter: true,
  };

  public rowData$!: Observable<any[]>;
  @ViewChild(AgGridAngular) agGrid!: AgGridAngular;
  constructor(private http: HttpClient,private modalService: NgbModal) {}
  onGridReady(params: GridReadyEvent) {
    const options = { headers:{}, withCredentials : true };
    this.rowData$ = this.http
      .get<any[]>('http://localhost:8080/demo/restFiles',options);
  }
}
```

Listing 6.21: data-grid-fs.component.ts.

Nella logica di questa pagina, oltre ai metodi di definizione per mappare i dati nelle colonne, possiamo evidenziare il metodo *onGridReady* che chiama il back-end all'endpoint /restFiles il cui controller restituisce tutti i file salvati in locale sotto forma di lista.

Infine, come ultima pagina, andremo ad analizzare *file-manager.component.html* al fine di mostrare le potenzialità del framework Angular. Infatti, come già discusso, Angular ci permette di suddividere la nostra pagina in blocchi (i component) che ci permettono di rendere le nostre pagine più compatte evitando di dover riscrivere codice che può essere utilizzato in più punti e gestirne la logica separatamente. Nello specifico, possiamo notare l'utilizzo dei componenti: <*bookshelf-data-grid-fs*> e <*bookshelf-data-grid-gd*> che sono le nostre dataGrid che mostrano i file, utilizzati semplicemente come tag HTML specificando il nome del selettore impostato nella relativa classe .ts.

```
1
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5    <meta charset="UTF-8">
6    <!--<noscript>
7      <meta http-equiv=refresh content='0; url=http://localhost:8080/demo/error?js=disabled'>
8    </noscript>-->
9    <link
10     href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
11     rel="stylesheet"
12     integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
13     crossorigin="anonymous"
14   />
15 </head>
16 <body>
17 <bookshelf-navbar></bookshelf-navbar>
18 <div class="container-fluid">
19   <div class="container">
20     <div class="col-auto text-center align-items-center pt-2">
21       <h1 class="pt-5">Lists of Files</h1>
22     </div>
23     <div class="row justify-content-center text-center pt-2">
24       <select class="form-select" aria-label="Default select example" >
25         <option value="" disabled selected>Seleziona un'Azienda</option>
26         <option *ngFor="let object of resp" [value]="object.id">{{ object.name }}</option>
27       </select>
28       <div class="col-2 pt-3">
29         <button #bisyncB id="sync" class="btn" type="button" (click)="bisync()"> <fa-icon [icon]="faArrowsRotate"></fa-icon><
               /button>
30       </div>
31     </div>
32       <div *ngIf="isLoading" class="overlay">
33         <mat-spinner></mat-spinner>
34       </div>
35   </div>
36 </div>
37 <div class="container-fluid">
38   <div class="row text-center pt-5">
39     <div class="col-sm-12 col-md-12 col-lg-6">
40       <h1 class="pt-2">FileSystem List</h1>
41       <!-- AG Grid Angular Component for local files-->
42       <bookshelf-data-grid-fs></bookshelf-data-grid-fs>
43     </div>
44     <div class="col-sm-12 col-md-12 col-lg-6">
45       <h1 class="pt-2">GoogleDrive List</h1>
46       <!-- AG Grid Angular Component for drive files-->
47       <bookshelf-data-grid-gd></bookshelf-data-grid-gd>
48     </div>
49   </div>
50   <div class="row mt-3 justify-content-center text-center">
51     <p *ngIf="isInvalid" class="error mt-2">Nome del file non valido,rinominare il file o selezionarne un altro.</p>
52     <button type="button" class="btn btn-primary mt-3" (click)="open(staticBackdrop)">
53       Upload File
54     </button>
55   </div>
56 </div>
57 <!-- Modal -->
58 <ng-template #staticBackdrop let-modal>
59   <div class="">
60       <div class="modal-header justify-content-center text-center">
61         <h5 class="modal-title" id="staticBackdropLabel">Upload File</h5>
62         <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
63       </div>
64       <div class="modal-body">
65         <form [formGroup]="fileForm" (submit)="onSubmit()" class="row g-3 justify-content-center" enctype="multipart/form-
               data">
66           <div class="col-auto">
```

```
67              <div class="mb-3">
68                <label for="formFile" class="form-label">Default file input example</label>
69                <input class="form-control" type="file" id="formFile" name="file" formControlName="file" (change)="loadFile(
                      $event)">
70              </div>
71            </div>
72            <div class="row justify-content-center">
73              <div class="col-auto">
74                <button id="submitFile" type="submit" value="submit" class="btn btn-primary mb-1">Upload File</button>
75                <a
76                  href="#"
77                  data-bs-toggle="tooltip"
78                  data-bs-title="Seleziona un file per fare l'upload"
79                ><i class="fa-solid fa-question fa-lg ms-3"></i></a>
80              </div>
81            </div>
82          </form>
83        </div>
84        <div class="modal-footer">
85          <button type="button" class="btn btn-secondary" (click)="close(staticBackdrop)">Close</button>
86        </div>
87      </div>
88
89  </ng-template>
90  <script
91    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"
92    integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN"
93    crossorigin="anonymous">
94  </script>
95  </body>
96  </html>
```

Listing 6.22: file-manager.component.html.

6.7 Demo Web App

L'utente esegue le seguenti operazioni:

Effettua la registrazione di un utente

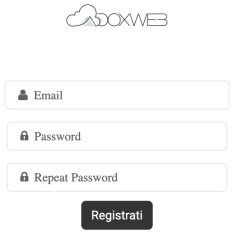


Figura 6.2: Pagina di registrazione

Effettua il login

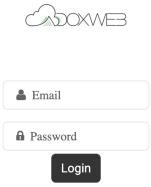


Figura 6.3: Pagina di login

Effettua login 2FA

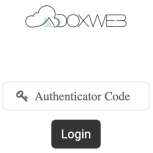


Figura 6.4: Pagina di login con 2FA

Visualizza la homepage

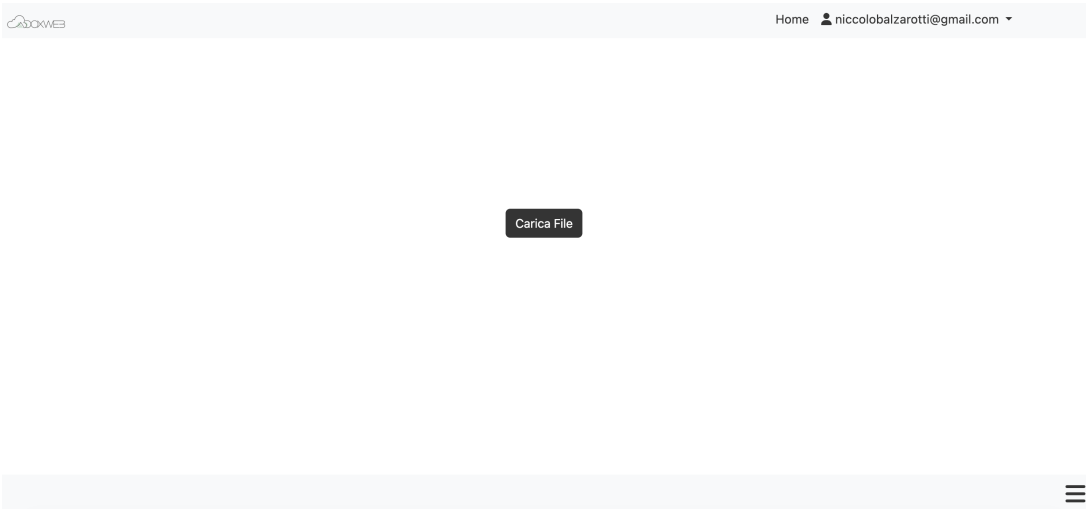


Figura 6.5: Home page

Apre la sezione profilo

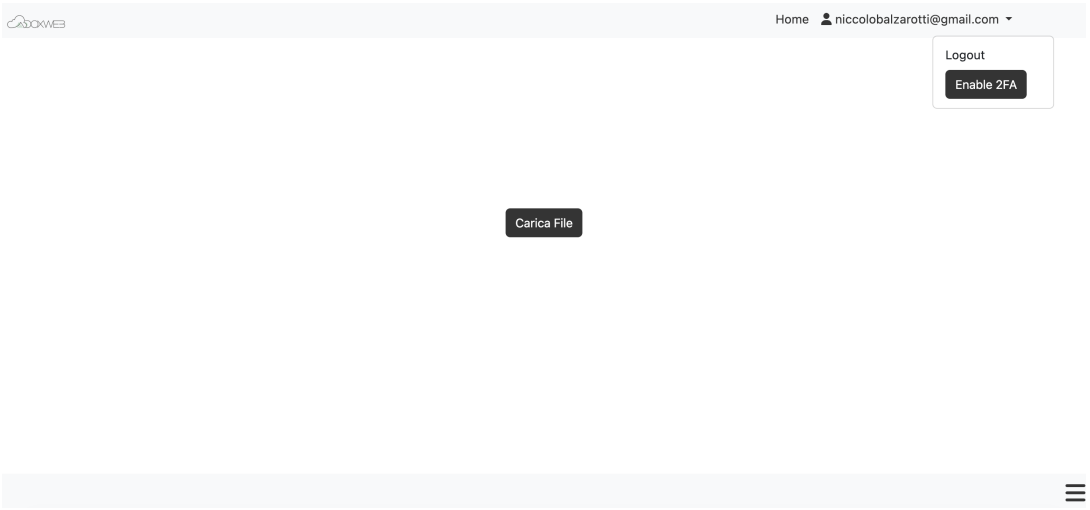


Figura 6.6: Dropdown profilo

Apre l'abilitazione 2FA

Figura 6.7: Pagina qr-code

Apre la dashboard ADMIN

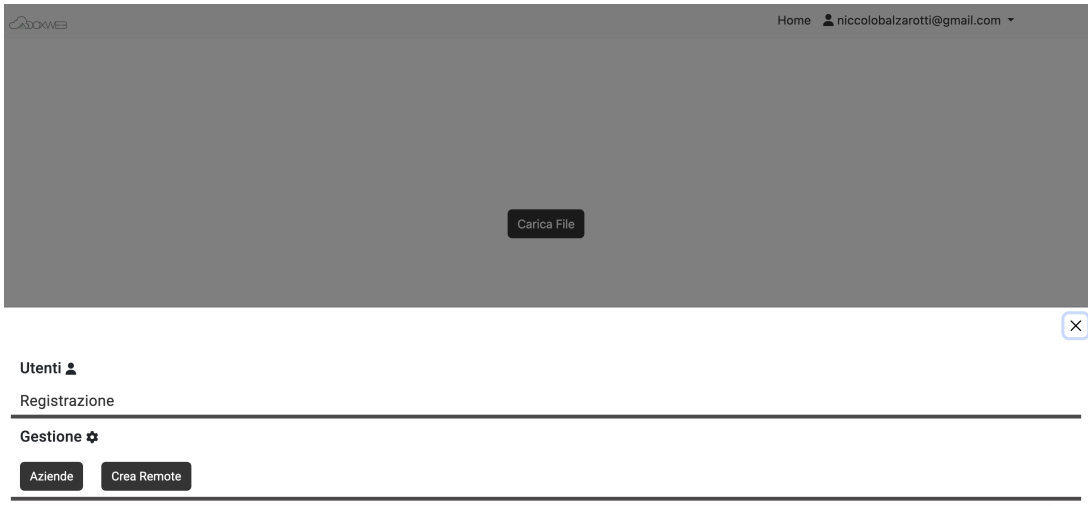


Figura 6.8: Funzioni ADMIN

Inserisce un'azienda

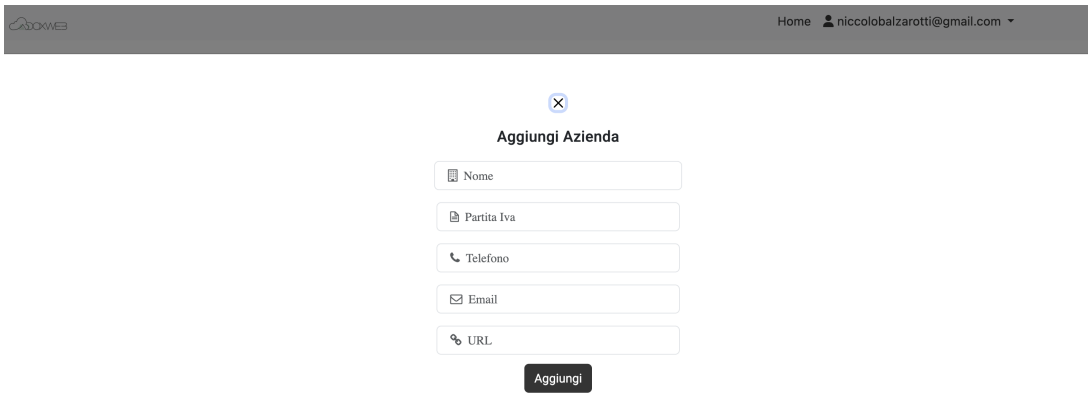


Figura 6.9: Modal creazione azienda

Datagrid aziende

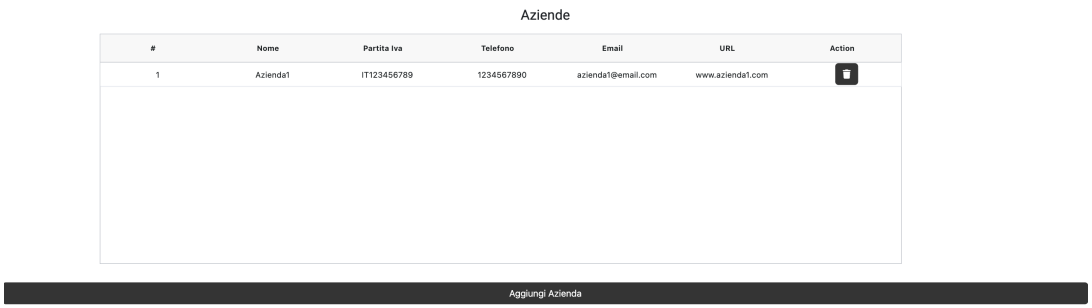


Figura 6.10: Pagina create-company

Crea un remote per l'azienda inserita

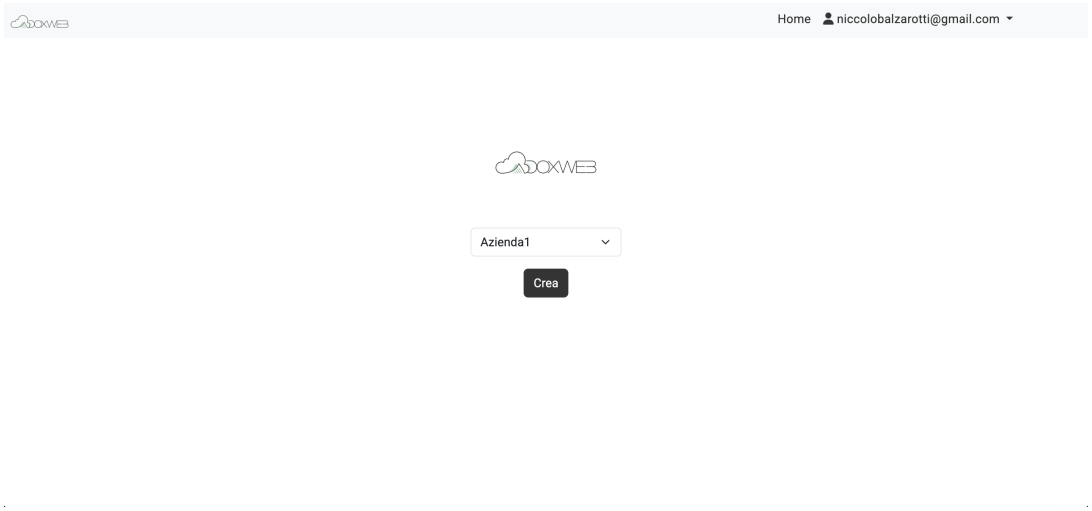


Figura 6.11: Pagina create-remote

File-manager

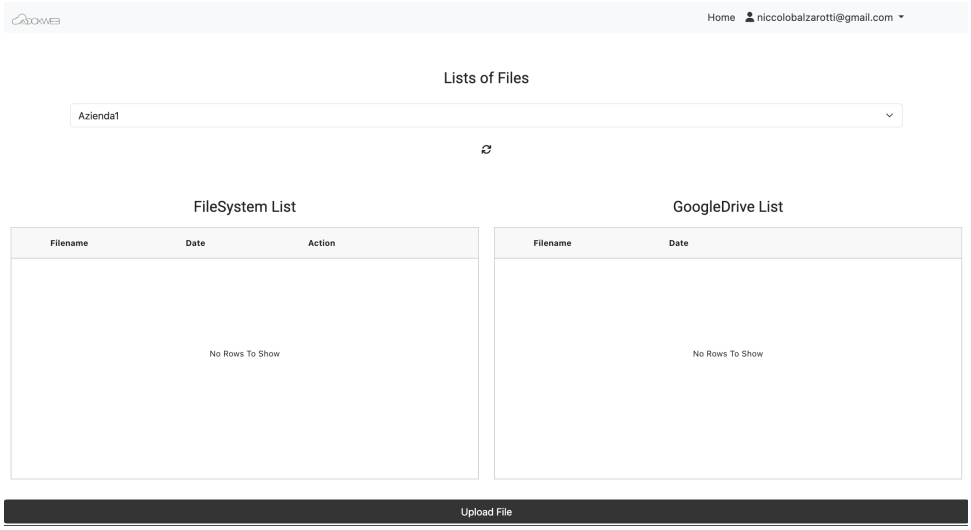


Figura 6.12: Pagina file-manager

Esegue l'upload di un file

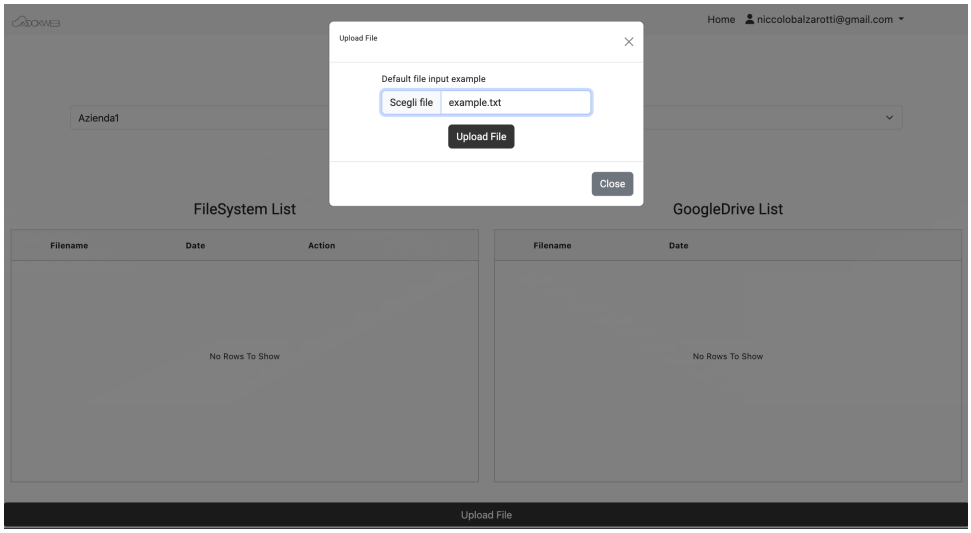


Figura 6.13: Modal upload file

File-manager

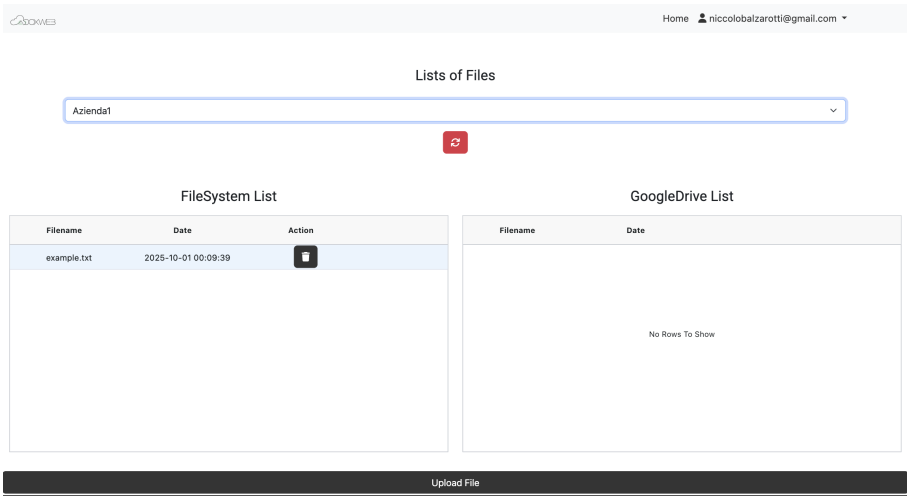


Figura 6.14: File caricato correttamente

Esegue la funzione di sincronizzazione

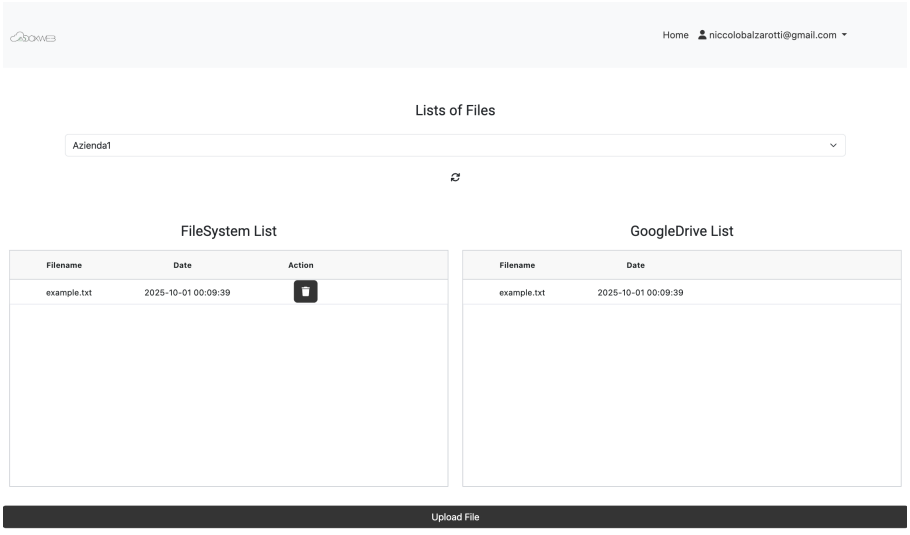


Figura 6.15: Pagina file-manager

Mostra l'avvenuta sincronizzazione su Google Drive

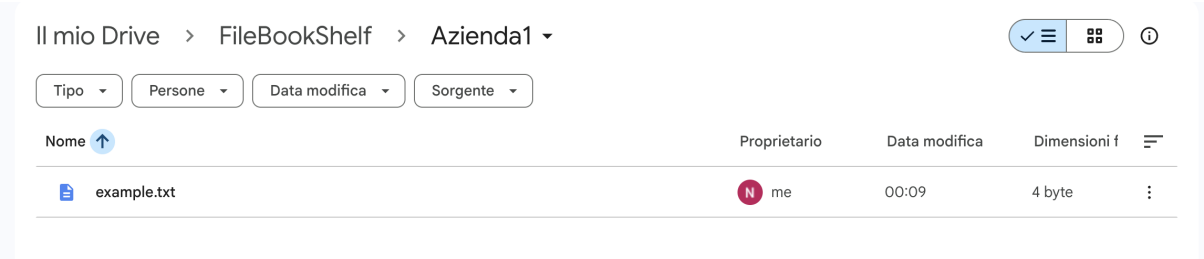


Figura 6.16: Remote Google Drive

Postfazione

Lavorare a questa tesi mi ha permesso di approfondire il mondo dello sviluppo web moderno, affrontando temi come la progettazione di una web app e l'integrazione con servizi cloud. Lo sviluppo e la progettazione di questa web app hanno richiesto un approccio full-stack, dall'architettura software all'implementazione del back-end e del front-end fino ad occuparmi della gestione dei dati e della loro integrazione con il cloud. Lo stage di lavoro che ho svolto è stato fondamentale per poter conoscere ed utilizzare strumenti e tecnologie utilizzati in contesti reali, permettendomi di mettere in pratica le conoscenze acquisite durante il percorso di studi e arricchendole con il lavoro pratico svolto in un contesto lavorativo reale, oltre che essere stata un'esperienza estremamente formativa che mi ha insegnato ad affrontare le sfide tecniche che mi si sono poste nell'utilizzo di nuove tecnologie, ma anche a saper lavorare in un contesto professionale, dinamico e innovativo. Ci tengo a ringraziare il mio relatore per tutte le linee guida e consigli che mi ha fornito, al fine di rendere la scrittura di questa tesi possibile.

Bibliografia

[1] PSN. L'evoluzione del cloud computing dagli anni '60 fino a oggi. URL https://www.polostrategiconazionale.it/media/news/evoluzione-del-cloud.

[2] Queen Elizabeth II. Elizabeth ii: la prima regnante a inviare una e-mail. URL https://www.rainews.it/articoli/2022/09/la-regina-elisabetta-fu-prima-regnante-a-inviare-una-e-mail-b8e46acc-1480-43b1-bfdc-06adf655e8f2.html.

[3] Compaq. Internet solutions division strategy for cloud computing. URL https://s3.amazonaws.com/files.technologyreview.com/p/pub/legacy/compaq_cst_1996_0.pdf.

[4] Wikipedia. On demand (informatica), . URL https://it.wikipedia.org/wiki/On_demand_(informatica).

[5] NIST. The nist definition of cloud computing. URL https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf.

[6] Wikipedia. Cloud computing, . URL https://it.wikipedia.org/wiki/Cloud_computing.

[7] Wikipedia. Pay per use, . URL https://it.wikipedia.org/wiki/PPU.

[8] Parlamento Europeo E Del Consiglio. Direttiva (ue) 2022/2555 del parlamento europeo e del consiglio del 14 dicembre 2022. URL https://eur-lex.europa.eu/legal-content/IT/TXT/PDF/?uri=CELEX:32022L2555.

[9] Atlassian. Che cos'è il cloud computing? una panoramica del cloud. URL https://www.atlassian.com/it/microservices/cloud-computing.

[10] Google Cloud. Paas, iaas, saas e caas: in che cosa differiscono? URL https://cloud.google.com/learn/paas-vs-iaas-vs-saas?hl=it.

[11] Red Hat. Tipologie di cloud computing, . URL https://www.redhat.com/it/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud.

[12] IBM. Cos'è un'api?, . URL https://www.ibm.com/it-it/think/topics/api.

[13] Wikipedia. Middleware, . URL https://it.wikipedia.org/wiki/Middleware.

[14] Wikipedia. Devops, . URL https://it.wikipedia.org/wiki/DevOps.

[15] Red Hat. Cosa sono i container?, . URL https://www.redhat.com/it/topics/containers.

[16] IBM. Cos'è il multi-tenant (o multitenancy)?, . URL https://www.ibm.com/it-it/topics/multi-tenant.

[17] IBM. Cos'è un server bare metal?, . URL https://www.ibm.com/it-it/think/topics/bare-metal-dedicated-servers.

[18] Wikipedia. Firewall, . URL https://it.wikipedia.org/wiki/Firewall.

[19] Akamai. Che cos'è l'architettura cloud? URL https://www.akamai.com/it/blog/cloud/what-is-cloud-architecture.

[20] RedHat. Cos'è l'architettura cloud? URL https://www.redhat.com/it/topics/cloud-computing/what-is-cloud-architecture.

[21] IBM. Cos'è l'architettura cloud?, . URL https://www.ibm.com/it-it/think/topics/cloud-architecture.

[22] HPE. Componenti dell'architettura cloud, . URL https://www.hpe.com/it/it/what-is/cloud-computing.html.

[23] HPE. Cos'è l'architettura cloud?, . URL https://www.hpe.com/it/it/what-is/cloud-architecture.html.

[24] IBM. Cosa sono il backup e il disaster recovery?, . URL https://www.ibm.com/it-it/think/topics/backup-disaster-recovery.

[25] IBM. Cos'è una cdn (content delivery network)?, . URL https://www.ibm.com/it-it/topics/content-delivery-networks.

[26] IBM. Cos'è l'sdn?, . URL https://www.ibm.com/it-it/topics/sdn.

[27] Google. Che cos'è la trasformazione digitale?, . URL https://cloud.google.com/learn/what-is-digital-transformation?hl=it.

[28] Google. Vantaggi e svantaggi del cloud computing, . URL https://cloud.google.com/learn/advantages-of-cloud-computing?hl=it.

[29] Osservatori Digital Innovation del Politecnico di Milano. Cloud computing: cos'è, come funziona e vantaggi. URL https://www.osservatori.net/blog/cloud-transformation/cloud-computing-significato-vantaggi/#Quali_sono_i_vantaggi_del_Cloud_Computing.

[30] Aulab. Cos'è una web app? URL https://aulab.it/blog/cose-una-web-app.

[31] Wikipedia. Model-view-controller, . URL https://it.wikipedia.org/wiki/Model-view-controller.

[32] Aulab. Il pattern mvc. URL https://aulab.it/guide-avanzate/il-pattern-mvc.

[33] Wikipedia. Single-page application, . URL https://it.wikipedia.org/wiki/Single-page_application.

[34] Wikipedia. Angular, . URL https://it.wikipedia.org/wiki/Angular.

[35] Wikipedia. Document object model, . URL https://it.wikipedia.org/wiki/Document_Object_Model.

[36] Creativemotion. Bootstrap: cos'è e come utilizzarlo per creare siti web responsive. URL https://www.creativemotions.it/cose-bootstrap/.

[37] Bootstrap. Breakpoints. URL https://getbootstrap.com/docs/5.3/layout/breakpoints/.

[38] IBM. Cos'è java spring boot?, . URL https://www.ibm.com/it-it/topics/java-spring-boot.

[39] GeekAndJob. Cos'è maven, . URL https://www.geekandjob.com/wiki/maven.

[40] Wikipedia. Apache tomcat, . URL https://it.wikipedia.org/wiki/Apache_Tomcat.

[41] IBM. Cos'è docker?, . URL https://www.ibm.com/it-it/think/topics/docker.

[42] GeekAndJob. Cos'è jpa, . URL https://www.geekandjob.com/wiki/jpa.

[43] Wikipedia. Persistenza (informatica), . URL https://it.wikipedia.org/wiki/Persistenza_(informatica).

[44] GeekAndJob. Cos'è hibernatecos'è hibernate, . URL https://www.geekandjob.com/wiki/hibernate.

[45] DZone. What is the difference between hibernate and spring data jpa? URL https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1.

[46] Wikipedia. Object-relational mapping, . URL https://it.wikipedia.org/wiki/Object-relational_mapping.

[47] ManagedServer. Rclone, cos'è, a cosa serve e come si integra nella tua strategia di backup?rclone, cos'è, a cosa serve e come si integra nella tua strategia di backup? URL https://managedserver.it/rclone-cose-a-cosa-serve-e-come-si-integra-nella-tua-strategia-di-backup/.

[48] Wikipedia. Funzione crittografica di hashfunzione crittografica di hash, . URL https://it.wikipedia.org/wiki/Funzione_crittografica_di_hash.

[49] Stolas informatica. Bcrypt: cos'è e come funziona? URL https://stolasinformatica.eu/sicurezza/bcrypt-cosa-come-funziona-guida/.