📘 Documento di Progettazione – *Quaderno*

Date:	14 set 2025
Author:	Luca De Castro Luca De Castro
Version:	1.0

+

📘 **Documento di Progettazione – Quaderno** **1**

1. Introduzione 2
2. Architettura ad alto livello 3
    2.1 Panoramica 3
    2.2 API Gateway 4
    2.3 Note Service 4
    2.4 Frontend 4
3. Modello dei dati 5
    3.1 Persistenza principale (SQL – PostgreSQL) 5
    3.2 Persistenza aggiuntiva (NoSQL - Elasticsearch) 6
    3.3 Cache condivisa (Redis) 6
4. Razionali progettuali ed architetturali e pipeline di CI/CD 7
5. Conclusioni 8

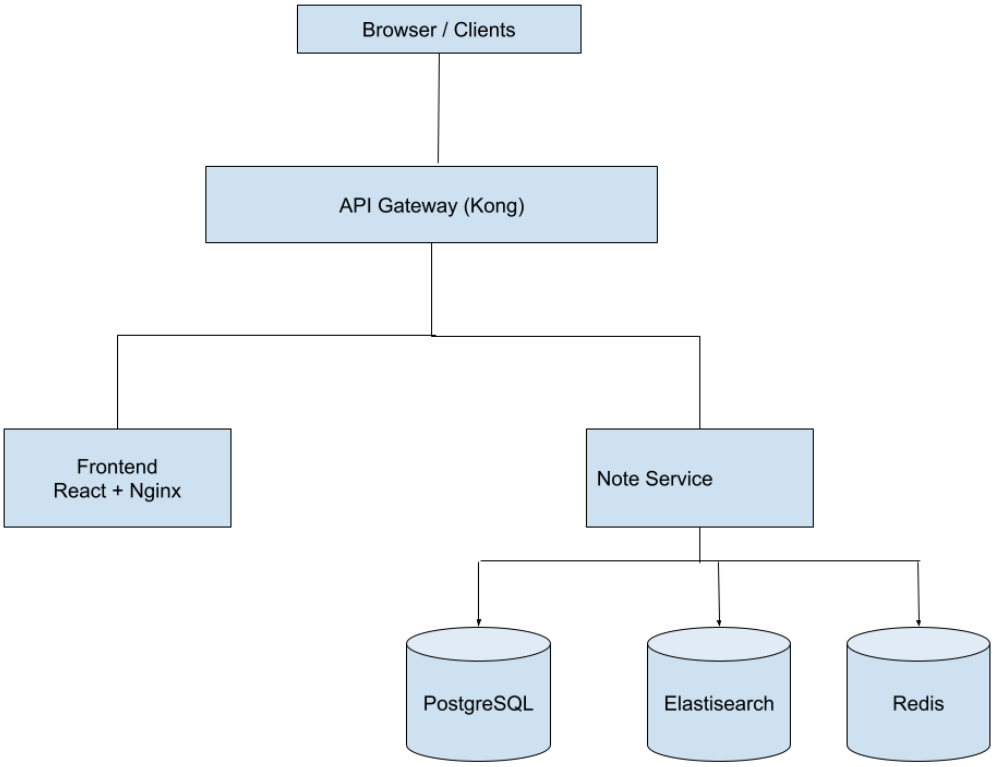
1. Introduzione

Quaderno è una web application per la gestione e la condivisione di note digitali, progettata per essere scalabile, sicura e collaborativa.
L'applicazione adotta un'architettura a microservizi, un sistema di autenticazione e meccanismi di ricerca avanzata, con l'obiettivo di supportare un'ampia base di utenti.

L'idea alla base del progetto è semplice ma allo stesso tempo molto efficace. Porre davanti a tutto in ingress, un APIGateway (es. Kong) che si occupa di gestire le chiamate in ingresso, cosi da poter gestire meglio traffico, limitazioni ed un eventuale autenticazione nel caso in cui si decide di utilizzare un sistema di gestione esterno delle utenze (es. Keycloak). Al momento visto la poca disponibilità di tempistiche, causa consegna ristretta ed impegni lavorativi, si è penssato utilizzare un APIGateway e di implementare la gestione utenze in autonomia, all'interno del microservizio che si occupa anche delle note, così da gestire l'autenticazione di Spring boot attraverso l'utilizzo di un JWT. Inoltre per veloccizare l'implementazione verrà utilizzato Jhipster, un generato di applicazioni, che ci viene incontro benissimo al nostro caso d'uso, velocizzando l'impostanzione del progetto e generando gran parte del codice, così da poter dedicare maggior tempo alla fase di analisi.

Fatta una panoramica generale andiamo ad analizzare nel dettaglio quando descritto in precedenza.

2. Architettura ad alto livello



2.1 Panoramica

L'applicazione Quaderno è stata progettata seguendo un'architettura a microservizi orchestrata all'interno di un cluster Kubernetes. L'utente accede al sistema tramite browser e interagisce con la Web Application sviluppata in ReactJS. Tutto il traffico esterno viene gestito da Kong, configurato come Ingress Controller e API Gateway, che rappresenta l'unico punto di accesso al cluster. Le richieste indirizzate alle API, identificate dal prefisso /api/***, vengono instradate dal Gateway al NoteService, il componente principale che attualmente gestisce sia la logica applicativa relativa alle note sia la gestione delle utenze. Le richieste alle risorse statiche e all'interfaccia web vengono invece servite dal servizio dedicato al frontend, che distribuisce la build di React tramite un container Nginx.

Per ragioni legate alle tempistiche di sviluppo si è scelto di accorpare nel NoteService anche le funzionalità di gestione utenti, inclusa l'autenticazione basata su JWT. Tuttavia, in una possibile evoluzione dell'architettura, tali responsabilità potrebbero essere separate delegando l'autenticazione e l'autorizzazione a un Identity Provider conforme a OIDC, come Keycloak, e introducendo un microservizio dedicato allo user management che funga da

intermediario tra l'applicazione e l'Identity Provider. Questa soluzione garantirebbe una maggiore modularità, una gestione più sicura e standardizzata delle identità e faciliterebbe l'integrazione con sistemi esterni, mantenendo il NoteService focalizzato unicamente sulla logica applicativa delle note.

2.2 API Gateway

L'API Gateway, realizzato con Kong e integrato nativamente in Kubernetes come Ingress Controller, rappresenta l'elemento centrale di mediazione tra i client esterni e i servizi interni. Ogni richiesta HTTP o HTTPS proveniente dal browser passa attraverso Kong, che la valida, la traccia e la reindirizza al componente corretto. Per le chiamate che riguardano le API, Kong inoltra il traffico al NoteService, mentre per le richieste relative alle risorse statiche si occupa di consegnare i contenuti del frontend React.

Il Gateway gestisce la sicurezza applicativa attraverso la validazione dei token JWT inclusi nelle richieste, assicurando che soltanto gli utenti autenticati possano accedere alle risorse protette. Inoltre, fornisce un livello di controllo centralizzato su policy di traffico come rate limiting, logging e metriche, facilitando l'osservabilità e la gestione degli accessi. In questo modo il Gateway costituisce un entry point unico, semplificando la governance dell'infrastruttura e riducendo la complessità per il team DevOps che si occupa di CI/CD e monitoraggio

2.3 Note Service

Il NoteService, sviluppato in Spring Boot e distribuito come microservizio all'interno del cluster Kubernetes, rappresenta il cuore logico dell'applicazione. Questo componente espone un insieme di API REST attraverso cui vengono gestite le funzionalità principali del sistema: la registrazione e l'autenticazione degli utenti, la creazione, lettura, aggiornamento e cancellazione delle note, l'assegnazione dei tag e la condivisione dei contenuti con altri utenti. La sicurezza è garantita tramite l'uso di token JWT, che vengono validati sia a livello di Gateway sia a livello applicativo.

Per quanto riguarda la gestione dei dati, il NoteService interagisce con diversi sistemi di persistenza. PostgreSQL è utilizzato come database relazionale principale e conserva tutte le informazioni strutturate relative a utenti, note, tag e condivisioni. Redis viene impiegato come cache condivisa per alleggerire il carico del database e velocizzare operazioni frequenti, come la validazione delle sessioni o il recupero rapido di dati già consultati. Elasticsearch è invece utilizzato per indicizzare i contenuti delle note e dei tag, consentendo ricerche full-text efficienti e performanti. Grazie a questa combinazione, il NoteService offre al tempo stesso consistenza dei dati e alte prestazioni nelle operazioni di consultazione.

2.4 Frontend

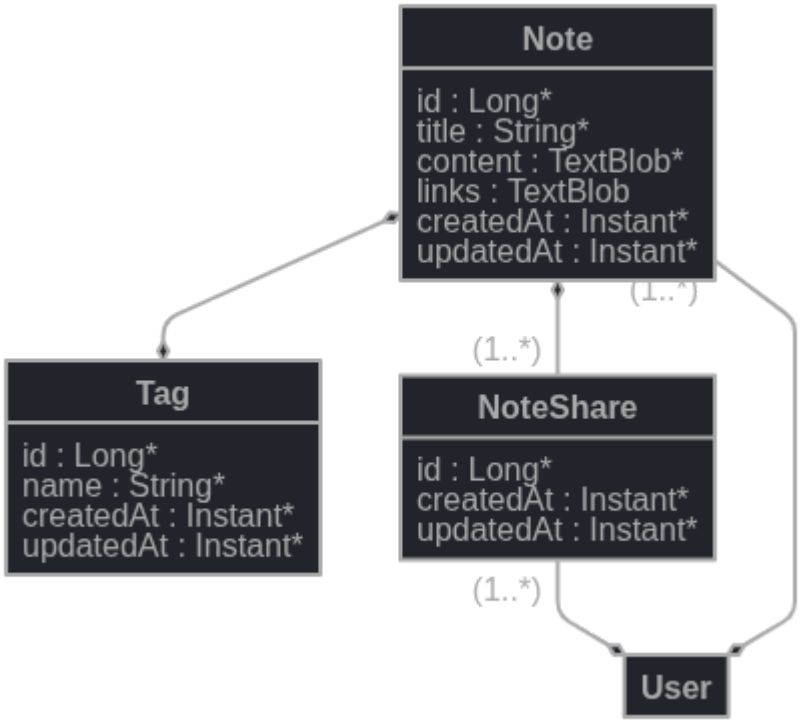
Il frontend dell'applicazione Quaderno è stato realizzato in ReactJS e fornisce l'interfaccia utente attraverso cui vengono consumati i servizi del backend. Una volta compilato, il progetto React viene distribuito come set di file statici serviti da un container Nginx, esposto

come Deployment nel cluster Kubernetes. Il traffico verso il frontend è incanalato anch'esso tramite Kong, che si occupa di consegnare al browser le risorse necessarie per caricare l'applicazione.

Il frontend offre all'utente un'esperienza interattiva e intuitiva, permettendo di eseguire il login, visualizzare la lista delle note, distinguere le note personali da quelle condivise, creare nuove note, aggiungere tag e condividere contenuti con altri utenti. Le operazioni sul frontend si traducono in chiamate API verso il NoteService, garantendo così una chiara separazione tra livello di presentazione e livello logico-applicativo. Il fatto che anche le risorse statiche siano servite dietro l'API Gateway rende l'architettura più uniforme, riduce la superficie di esposizione dei servizi interni e semplifica notevolmente il lavoro di deploy e manutenzione all'interno del cluster Kubernetes.

3. Modello dei dati

3.1 Persistenza principale (SQL – PostgreSQL)



Lo schema mostrato nell'immagine è un JDL, una notazione UML adattata a JHipster, pensata per essere importata direttamente nel framework al fine di generare automaticamente le relative classi e i test associati.

Per garantire la consistenza e l'integrità dei dati, l'applicazione Quaderno utilizza **PostgreSQL** come database relazionale principale. Questo livello di persistenza conserva le informazioni strutturate relative agli utenti, alle note, ai tag e alle condivisioni. Il modello dati è stato progettato per riflettere le esigenze applicative, assicurando coerenza nelle relazioni e supportando i vincoli di integrità referenziale.

La tabella **User** rappresenta l'entità fondamentale per la gestione delle identità applicative. Contiene informazioni quali identificativo univoco, username, email, hash della password e metadati relativi alla creazione e all'aggiornamento del record.

La tabella **Note** conserva i contenuti testuali prodotti dagli utenti. Ogni nota è caratterizzata da un identificativo, un titolo, il corpo testuale, l'utente proprietario e i riferimenti temporali relativi alla creazione e all'ultima modifica. Le note sono collegate ai tag attraverso una relazione molti-a-molti.

La tabella **Tag** definisce i descrittori tematici associabili alle note. Ogni tag è identificato in modo univoco ed è collegato a una o più note tramite la tabella di relazione **NoteTag** (non presente nel JDL perché creata automaticamente dalla relazione *ManyToMany*), che associa note e tag mantenendo una struttura relazionale normalizzata.

Infine, la tabella **NoteShare** consente di rappresentare le relazioni di condivisione tra note e utenti. Ciascun record indica quale nota è stata condivisa, con quale utente destinatario e quando la condivisione è stata creata. In questo modo si supporta il requisito funzionale che prevede la visualizzazione in sola lettura da parte dei destinatari diversi dall'owner.

3.2 Persistenza aggiuntiva (NoSQL - Elasticsearch)

Accanto al database relazionale, l'applicazione utilizza **Elasticsearch** per garantire un motore di ricerca full-text performante. Questo sistema di persistenza è particolarmente adatto per interrogazioni basate sul contenuto delle note e sui tag, superando le limitazioni dei database relazionali in termini di rapidità e scalabilità per ricerche testuali complesse.

Ogni nota gestita dal sistema viene indicizzata in Elasticsearch attraverso un documento contenente l'identificativo della nota, il titolo, il contenuto, la lista di tag associati e lo username del proprietario. Questa duplicazione dei dati rispetto a PostgreSQL non sostituisce la persistenza relazionale, ma l'affianca per ottimizzare le operazioni di consultazione.

Grazie a questa struttura, l'utente può eseguire ricerche istantanee sia per testo libero sia per tag, ottenendo risposte rapide anche in presenza di un numero elevato di note. L'indice può inoltre essere aggiornato in tempo reale a seguito di operazioni CRUD sul NoteService, così da mantenere sempre allineata la base dati tra PostgreSQL ed Elasticsearch.

3.3 Cache condivisa (Redis)

Per migliorare le prestazioni complessive e ridurre il carico sul database relazionale, l'architettura integra **Redis** come sistema di cache distribuita. Redis viene utilizzato principalmente per memorizzare informazioni di sessione e risultati di query frequentemente richieste. Ciò consente di servire rapidamente le richieste ripetitive degli utenti e di alleggerire il numero di accessi diretti a PostgreSQL, aumentando la scalabilità complessiva del sistema.

Il NoteService gestisce in maniera trasparente l'interazione con Redis, decidendo quali dati devono essere memorizzati temporaneamente e quali recuperati dal database relazionale. Questa strategia di caching fornisce un compromesso ottimale tra persistenza affidabile e velocità di risposta.

4. Razionali progettuali ed architetturali e pipeline di CI/CD

La progettazione dell'applicazione *Quaderno* è stata guidata dalla necessità di coniugare semplicità d'uso, sicurezza, scalabilità e facilità di gestione in ambiente cloud-native. L'adozione di un'architettura a microservizi si è rivelata la scelta più idonea per ottenere modularità e indipendenza dei componenti, consentendo al tempo stesso di distribuire il carico in maniera flessibile e di scalare i singoli servizi in funzione delle esigenze reali. All'interno di questa architettura, l'API Gateway basato su Kong assume un ruolo cruciale, poiché rappresenta l'unico punto di accesso al sistema, semplifica il routing verso i servizi interni e centralizza la gestione della sicurezza, del monitoraggio e delle politiche di traffico.

Il cuore applicativo è costituito dal NoteService, implementato con Spring Boot per sfruttare la solidità e la maturità dell'ecosistema Java. In esso sono state integrate sia le funzionalità di gestione delle note sia quelle relative agli utenti, scelta dettata principalmente da ragioni di tempo e semplicità. In una successiva evoluzione si potrà tuttavia prevedere la separazione della gestione delle identità in un servizio dedicato, con l'introduzione di un Identity Provider come Keycloak. Questa modifica renderebbe l'architettura ancora più modulare e allineata alle best practice in materia di identity management.

La scelta di PostgreSQL come database relazionale principale è stata motivata dall'affidabilità, dalla capacità di garantire consistenza transazionale e dalla sua adattabilità a scenari enterprise. L'integrazione di Elasticsearch risponde invece all'esigenza di fornire un motore di ricerca full-text ad alte prestazioni, particolarmente utile in un contesto applicativo basato sulla gestione e condivisione di note testuali. Redis, infine, è stato introdotto per implementare un meccanismo di cache distribuita che consente di ridurre i tempi di risposta e il carico sul database relazionale, migliorando così l'esperienza utente e la scalabilità complessiva.

Dal punto di vista della sicurezza, la combinazione tra l'uso di token JWT e la validazione centralizzata a livello di Gateway assicura un controllo degli accessi robusto e standardizzato. La gestione del traffico tramite rate limiting e il supporto a TLS sul Gateway

completano il quadro, riducendo la superficie di attacco e garantendo la protezione dei dati in transito.

Per quanto riguarda l'approccio DevOps, si è deciso prevedere un supporto ad una distribuzione con Kubernetes. Ogni componente viene pacchettizzato come immagine Docker e così da poter essere distribuito in un eventuale cluster tramite pipeline automatizzate. Il flusso ad alto livello prevede che ogni modifica al codice sorgente attivi una pipeline di Continuous Integration, che include fasi di linting, test automatici e build delle immagini. Una volta superata questa fase, le immagini vengono pubblicate in un container registry e rese disponibili per il deployment. La fase di Continuous Deployment consiste quindi nell'aggiornamento dell'ambiente Kubernetes attraverso manifest YAML o chart Helm versionati e mantenuti in un repository dedicato.

Questa pipeline garantisce rapidità, coerenza e tracciabilità nelle release, permettendo di ridurre gli errori manuali e di facilitare l'allineamento tra ambienti di sviluppo, test e produzione. Grazie a questa integrazione, il team DevOps può gestire l'applicazione in modo semplice ed efficace, monitorando il comportamento del sistema e applicando eventuali correzioni o ottimizzazioni con cicli iterativi brevi e controllati.

5. Conclusioni

L'intero progetto è stato pensato per evolvere in maniera graduale: la futura introduzione di un Identity Provider OIDC come Keycloak e di un microservizio dedicato alla gestione degli utenti consentirebbe di aumentare ulteriormente la modularità e la sicurezza. L'approccio DevOps integrato, con pipeline di CI/CD che automatizzano build, test e deploy, permette di ridurre gli errori manuali e garantire cicli di rilascio più rapidi e affidabili.

In conclusione, Quaderno offre un compromesso equilibrato tra semplicità e robustezza, ponendo solide basi per una crescita futura sia dal punto di vista funzionale che da quello architettonico. L'applicazione è già in grado di soddisfare i requisiti essenziali, ma al tempo stesso mantiene un design aperto a miglioramenti ed estensioni che potranno essere implementati senza impatti significativi sull'impianto generale.