Specifica Tecnica
Versione 0.2.0

Stato	In redazione
Responsabile	Marco Piccoli
Verificatore	Emanuele Artusi
Redattori	Sara Ferraro
Distribuzione	*ALimitedGroup* M31 Prof. Tullio Vardanega Prof. Riccardo Cardin

Descrizione

Questo documento contiene la *Specifica Tecnica* descritto dal gruppo *ALimitedGroup* per il Capitolato numero 6 proposto da M31

Registro delle Modifiche

Vers.	Data	Autore	Verificatore	Descrizione
0.2.0	2025-02-27	S. Ferraro	M. Piccoli	Continuazione sezione architettura.
0.1.0	2025-02-25	S. Ferraro	E. Artusi	Prima redazione documento. Sezione introduzione. Sezione tecnologie. Sezione architettura.

Indice

Lista delle tabelle

Lista delle immagini

1 - Introduzione

1.1 - Scopo del documento

Il presente documento ha l'obiettivo di descrivere in dettaglio l'*architettura* del prodotto software, fornendo una visione chiara e strutturata delle sue componenti, della loro interazione e della loro distribuzione nel sistema.

Il documento di **Specifica Tecnica**[G] funge da riferimento per la *progettazione* e *realizzazione del prodotto*, garantendo coerenza con il *Proof of Concept*[G] (PoC[G]) iniziale e introducendo miglioramenti volti a consolidarne la maturità architetturale.

Nello specifico, questo documento si propone di:

- Definire l'**architettura logica** del prodotto, illustrando le componenti principali, i loro ruoli e le interconnessioni tra di esse;
- Esporre l'**architettura di deployment**[G], delineando la distribuzione delle componenti nel sistema in esecuzione;
- Documentare i **design pattern architetturali** adottati, evidenziando le scelte progettuali derivate dalle tecnologie selezionate;
- Identificare eventuali **idiomi** (pattern di livello inferiore) utilizzati per ottimizzare la qualità del codice;
- Fornire ulteriori **dettagli progettuali** che valorizzino le scelte architetturali e facilitino la comprensione e manutenzione del prodotto.

1.2 - Glossario

Per tutte le *definizioni*, *acronimi* e *abbreviazioni* utilizzati in questo documento, si faccia riferimento al **Glossario**, fornito come documento separato, che contiene tutte le spiegazioni necessarie per garantire una comprensione uniforme dei termini tecnici e dei concetti rilevanti per il progetto.
Le parole che possiedono un riferimento nel Glossario saranno indicate nel modo che segue:

`parola`[G]

1.3 - Riferimenti

1.3.1 - Riferimenti normativi

- **Capitolato d'appalto C6: Sistema di Gestione di un Magazzino Distribuito - M31**
  https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C6.pdf
  Ultimo Accesso 25 Febbraio 2025
- **Norme di Progetto**[G] **ver. 1.0.0**
  https://alimitedgroup.github.io/NP%20v1.0.0.pdf
  Ultimo Accesso 25 Febbraio 2025

1.3.2 - Riferimenti informativi

- **Lezione rovesciata - Documentazione**
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/FC1.pdf
  Ultimo Accesso 25 Febbraio 2025

- **Regolamento del Progetto didattico**
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf
  Ultimo Accesso 25 Febbraio 2025
- **Glossario ver. 1.0.0**
  https://alimitedgroup.github.io/Glossario.pdf
  Ultimo Accesso 25 Febbraio 2025

2 - Tecnologie

Il progetto si basa su un insieme di tecnologie moderne e robuste, selezionate per le loro capacità di supportare efficacemente un'architettura a microservizi e garantire scalabilità, affidabilità e manutenibilità del sistema.

La scelta tecnologica è stata guidata dalla necessità di creare un sistema di gestione del magazzino distribuito che possa operare in modo efficiente anche in condizioni di carico variabile, mantenendo elevati standard di prestazioni e resilienza.

Le tecnologie adottate sono state organizzate in categorie, in base al loro ruolo all'interno dell'architettura: linguaggi di programmazione per lo sviluppo del codice, strumenti per la comunicazione tra microservizi, soluzioni per la containerizzazione e il *deployment*, e piattaforme per il monitoraggio del sistema.

Di seguito sono elencate e descritte le tecnologie utilizzate, evidenziando le loro caratteristiche principali.

2.1 - Linguaggi di programmazione e *framework*

Tecnologia	Versione	Descrizione
GO		Go è un linguaggio di programmazione open-source sviluppato da Google, progettato per essere efficiente, semplice e scalabile. È particolarmente adatto per lo sviluppo di sistemi distribuiti, microservizi e applicazioni cloud-native, grazie alla sua compilazione rapida, alla gestione automatica della memoria e alla facilità di deployment con binari standalone.

Tabella 1: Tecnologie per la programmazione e lo sviluppo software

2.2 - Tecnologie per la comunicazione e messaggistica

Tecnologia	Versione	Descrizione
NATS		NATS è un sistema di messaggistica open-source progettato per la comunicazione scalabile, affidabile e a bassa latenza tra servizi distribuiti. Supporta il pub/sub, request/reply e message queueing, rendendolo adatto a microservizi. Grazie alla sua leggerezza e semplicità, NATS permette un'elevata efficienza nella gestione della comunicazione tra componenti, garantendo resilienza e facilità di scalabilità senza necessità di configurazioni complesse.

Tabella 2: Tecnologie per la comunicazione e messaggistica

2.3 - Tecnologie per la containerizzazione e *deployment*

Tecnologia	Versione	Descrizione
Docker		Docker è una piattaforma di containerizzazione che consente di impacchettare applicazioni e le loro dipendenze in container leggeri e portabili. Grazie alla sua architettura basata su immagini e container, Docker permette di garantire consistenza tra ambienti di sviluppo, test e produzione, semplificando il deployment e la scalabilità delle applicazioni. È particolarmente utile per microservizi e sistemi distribuiti, migliorando l'efficienza nell'uso delle risorse e la velocità di distribuzione del software.

Tabella 3: Tecnologie per la containerizzazione e *deployment*

2.4 - Tecnologie per il monitoraggio dei microservizi

Tecnologia	Versione	Descrizione
Grafana		Grafana è una piattaforma open-source per la visualizzazione e l'analisi di dati di monitoraggio. Supporta diverse fonti di dati (come Prometheus, Loki e Mimir) e consente la creazione di dashboard interattive per il monitoraggio in tempo reale.
Prometheus		Prometheus è un sistema di monitoraggio e allerta open-source focalizzato sulla raccolta di metriche attraverso un modello pull.
Loki		Loki è un sistema di log aggregation sviluppato da Grafana Labs, ottimizzato per la gestione dei log in modo scalabile ed efficiente. Si integra con Grafana per la visualizzazione e utilizza un'architettura simile a Prometheus, semplificando la correlazione tra metriche e log.
Mimir		Mimir è un'estensione di Prometheus sviluppata da Grafana Labs per la gestione di metriche su larga scala. Consente lo storage e la gestione distribuita di serie temporali, migliorando la scalabilità e la resilienza rispetto a un'istanza standalone di Prometheus.

Tabella 4: Tecnologie per il monitoraggio dei microservizi

3 - Architettura

3.1 - Architettura logica

Il sistema è progettato seguendo l'**architettura esagonale**, un modello che promuove una netta separazione tra la logica di *business* e le interazioni con servizi esterni, fonti di dati e interfacce utente.
Questo approccio organizza il sistema attorno a un nucleo centrale, circondato da porte che fungono da interfacce con il mondo esterno, garantendo modularità e testabilità.

Il **nucleo** dell'applicazione contiene la logica di dominio e le regole di *business*, progettato per essere indipendente dai dettagli tecnologici esterni, in modo da favorire la manutenibilità e l'estendibilità del sistema.

Le **porte** costituiscono il punto di connessione tra il nucleo e il mondo esterno, consentendo una comunicazione strutturata:

- *Inbound Ports* (o *Use Cases*): consentono l'invocazione della logica del nucleo da parte di componenti esterni, definendo i punti di accesso all'applicazione e isolando la logica di dominio da implementazioni tecnologiche specifiche.
- *Outbound Ports*: permettono al nucleo di interagire con servizi esterni, mantenendo un'astrazione che preserva l'indipendenza della logica di business dai dettagli di implementazione.

I ***services*** implementano le inbound ports e fanno parte della business logic, concentrandosi esclusivamente sulla logica di dominio senza dipendenze tecnologiche specifiche.

Gli ***adapters*** rappresentano lo strato esterno del sistema e si suddividono in:

- *Input Adapters* (o *Controllers*): ricevono input dall'esterno e invocano le operazioni sulle porte in ingresso, traducendo le richieste esterne in operazioni comprensibili per il nucleo.
- *Output Adapters*: gestiscono la comunicazione con l'esterno attraverso le porte in uscita, traducendo le risposte del nucleo in formati comprensibili per i servizi esterni.

3.2 - Architettura di *deployment*

3.2.1 - Sistema a microservizi

L'architettura di deployment[G] adottata per il sistema è basata su **microservizi**, come richiesto dal capitolato[G] .
Questa scelta consente una maggiore scalabilità, resilienza e indipendenza nello sviluppo e nel *deployment* dei componenti software.

Ogni microservizio è indipendente e responsabile di un insieme specifico di funzionalità[G].

I microservizi comunicano tra loro tramite NATS[G] , un sistema di messaggistica publish-subscribe ad alte prestazioni. Questa soluzione permette:

- Comunicazione asincrona, sincrona ed *event-driven*, riducendo l'accopiamento tra i servizi;
- Maggiore scalabilità, in quanto i messaggi possono essere gestiti in parallelo;
- Affidabilità nella trasmissione dei dati grazie alla capacità di gestire il *buffering* e il re-invio dei messaggi in caso di errore.

Oltre a NATS [G], i microservizi possono esporre API REST per le comunicazioni con il *client*.

Il *deployment* dei microservizi avviene in ambienti containerizzati tramite Docker[G] . Questo garantisce:

- Scalabilità dinamica, adattando le risorse ai carichi di lavoro;
- Isolamento dei servizi, evitando impatti negativi tra componenti;
- Gestione semplificata del ciclo di vita dei servizi.

Questa architettura consente di ottenere un sistema altamente scalabile, resiliente e facilmente manutenibile, ottimizzato per ambienti distribuiti e carichi di lavoro variabili.

3.2.2 - *Client* monolitico

Il *client* è progettato come un'applicazione monolitica che funge da interfaccia unificata verso i diversi microservizi del *backend*[G] .Questa scelta architetturale offre diversi vantaggi:

- Esperienza utente coerente: un'interfaccia unificata garantisce consistenza nell'interazione con le diverse funzionalità del sistema;
- Semplificazione della gestione dello stato: la gestione delle sessioni utente e della sincronizzazione dei dati sono facilitate;
- Ottimizzazione delle comunicazioni: il *client* gestisce in modo efficiente le chiamate verso i diversi microservizi, mascherando la complessità dell'architettura distribuita all'utente finale.