**INFO PROGETTO**

**Qui stiamo definendo un micrservizio di catalogo dei prodotti di un e-commerce.**

**Per il progetto, adottiamo una decomposizione funzionale combinata con decomposizione per dominio.**

**Flusso delle operazioni**

1. **Richiesta: L'utente invia una richiesta HTTP.**
2. **Controller: Il controller riceve la richiesta e interagisce con il servizio.**
3. **Servizio: Il servizio applica la logica e utilizza il repository per accedere ai dati.**
4. **Repository: Il repository interagisce con MongoDB e restituisce i dati al servizio.**
5. **Risposta: Il servizio invia il risultato al controller, che lo restituisce come risposta HTTP.**

**Questo design segue il principio Clean Architecture, separando le responsabilità in core, adattatori e infrastruttura.**

**Numero le pagine andando in ordine partendo dalla request fino al db,**

**1. src/core/domain/Product.ts**

* **Definizione del modello: Rappresenta l'entità Product con i campi necessari:**
  + **id, name, description, price, stock.**
* **Ruolo: È la rappresentazione del dominio dell'applicazione.**

**Partiamo quindi dalla definizione del modello del prodotto che utilizzeremo in questo servizio**

**Immagine che contiene schermata, software, Software multimediale, testo

Descrizione generata automaticamente**

**2. src/index.ts**

* **Funzione principale: Configura il server Express, la connessione a MongoDB e definisce i percorsi delle API (/products, /products/:id).**
* **Ruolo: È il punto di ingresso dell'applicazione.**
* **Processo:**
  + **Connette MongoDB.**
  + **Assegna le funzioni dei controller alle rotte:**
    - **POST /products → createProduct.**
    - **GET /products → getAllProducts.**
    - **GET /products/:id → getProductById.**

**Immagine che contiene schermata, testo, software, Software multimediale

Descrizione generata automaticamente**

**3. src/adapters/http/ProductController.ts**

* **Controller: Implementa i metodi delle API.**
* **Processo:**
  + **Inizializza il repository (ProductRepository) e il servizio (CatalogService).**
  + **Assegna la logica alle API:**
    - **createProduct: Riceve i dati del prodotto dal body e li salva.**
    - **getAllProducts: Recupera tutti i prodotti e li restituisce.**
    - **getProductById: Recupera un prodotto per ID e lo restituisce.**
* **Ruolo: Punto di ingresso delle richieste HTTP.**

**Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente**

**E come vedi stiamo creando un istanza CatalogService che nello specifico utilizza mongoDB che come vedremo dopo nella classe nel costruttore ha la port (interfaccia) di repository slengando l’implementazioni dal core.**

Adattatore (ProductRepository):

* È l'implementazione specifica del repository che utilizza MongoDB come storage.
* Implementa l'interfaccia ProductRepositoryPort, permettendo al core di interagire con il database senza conoscerne i dettagli.

Core (CatalogService):

* È indipendente dall'implementazione specifica del repository.
* Accetta qualsiasi implementazione che rispetti l'interfaccia ProductRepositoryPort, rendendolo riutilizzabile e testabile.

Concetto di inversione delle dipendenze:

* Iniettando l'adattatore nel core, hai separato le responsabilità tra dominio (core) e infrastruttura (repository).
* Questo design consente di cambiare l'implementazione (ad esempio, usare un database diverso da MongoDB) senza modificare il core.

Scopo del microservizio:

* Il microservizio diventa focalizzato sul dominio (gestione dei prodotti), mentre la persistenza è solo un dettaglio di implementazione.
* Raggiungi un'architettura più flessibile e manutenibile.

**4. src/core/ports/ProductServicePort.ts**

* **Interfaccia: Definisce i metodi che il servizio deve implementare:**
  + **addProduct: Aggiunge un prodotto.**
  + **getAllProducts: Recupera tutti i prodotti.**
  + **getProductById: Recupera un prodotto per ID.**
* **Ruolo: È un contratto (porta di input) per il core dell'applicazione.**

**Quindi dal controller che è un adapter per le richieste http comunichiamo all’interfaccia (port di input) specifica:**

**Immagine che contiene schermata, software, testo, Software multimediale

Descrizione generata automaticamente**

**5. src/core/application/CatalogService.ts**

* **Servizio del dominio: Implementa la logica principale utilizzando la porta di output (ProductRepositoryPort).**
* **Ruolo: Coordina le operazioni con il repository:**
  + **Aggiunge un prodotto (addProduct).**
  + **Recupera tutti i prodotti (getAllProducts).**
  + **Recupera un prodotto per ID (getProductById).**

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

**E come dall’interfeccia ci spostiamo alla definizione del servizio che avevamo creato nel controller che, come vedi, utilizza l’interfaccia productRepositoryPort nel costruttore rendendo flessibile e non specificatamente legato ad una implemntazione il core.**

**6. src/core/ports/ProductRepositoryPort.ts**

* **Interfaccia: Definisce i metodi che un repository deve implementare:**
  + **save: Salva un prodotto.**
  + **getAll: Recupera tutti i prodotti.**
  + **getById: Recupera un prodotto per ID.**
* **Ruolo: È un contratto (porta di output) che il repository deve rispettare.**

Immagine che contiene schermata, testo, software, Software multimediale

Descrizione generata automaticamente

**Questa è l’interfaccia utilizzata nel costruttore del servizio che sarà una port di output dato che una volta ricevuta la richiesta esegue una operazione esterna dal core in direzione del repository (mongo nel nostro caso).**

**7. src/adapters/persistence/ProductRepository.ts**

* **Repository: Implementa l'interfaccia ProductRepositoryPort interagendo con MongoDB.**
* **Processo:**
  + **Usa un modello Mongoose (ProductModel) per mappare i dati su MongoDB.**
  + **Converte i documenti MongoDB in oggetti Product del dominio.**
* **Ruolo: Adattatore che collega il database al core.**

Immagine che contiene schermata, testo, software, Software multimediale

Descrizione generata automaticamente

**E qui siamo nell’adapter che si va ad interfacciare con il repository specifico ossia mongodb.**

**In questo modo la richiesta arriva al db e poi torna indietro non legando il core alle diverse implementazioni nel nostro caso express e mongodb.**

**Vediamo perché utilizzare sia la classe Product (che rappresenta il dominio del prodotto) che il productSchema con Mongoose è utile:**

**Differenza tra la tua classe Product e IProductDocument + productSchema:**

1. **Classe Product (dominio):**
   * **Questa classe è una rappresentazione del prodotto nel tuo codice. È utilizzata per la logica di business e le operazioni generali sul prodotto.**
   * **È il modello che definisce la struttura del prodotto in memoria (nel codice) e non ha alcuna connessione diretta con il database.**
2. **IProductDocument e productSchema (MongoDB e Mongoose):**
   * **IProductDocument: è usato per definire come MongoDB gestisce i dati in base alla struttura del documento. È specifico per la struttura dei dati nel database e per garantire che TypeScript sappia che tipo di dato aspettarsi quando interagisci con Mongoose.**
   * **productSchema: è il modo in cui MongoDB è informato su come dovrebbe essere strutturato un documento Product nel database. In altre parole, definisce la struttura del prodotto nel contesto di MongoDB.**

**Perché non basta la tua classe Product:**

* **Compatibilità con MongoDB: La classe Product che hai in Product.ts è una semplice classe TypeScript, mentre MongoDB, per funzionare correttamente con Mongoose, ha bisogno di una definizione che gli dica come dovrebbe essere strutturato ogni documento. Questa definizione è quella che ottieni tramite productSchema.**
* **Interazione con il database: MongoDB richiede una struttura definita per poter salvare e leggere i dati in modo efficiente. Mongoose fornisce questa struttura tramite il productSchema che collega il modello di dati (Product) al database.**
* **Accesso ai dati con Mongoose: Quando usi Mongoose, l'oggetto restituito da operazioni come find() o save() non è direttamente un'istanza della tua classe Product. Questi oggetti sono documenti Mongoose che hanno metodi speciali (ad esempio save(), update(), ecc.). Per fare in modo che TypeScript lavori con questi oggetti come se fossero una classe Product (con i metodi e la struttura che hai definito), usi IProductDocument e productSchema per "connettere" il mondo MongoDB con il mondo TypeScript.**
* **ProductModel è un oggetto creato da Mongoose che ci permette di interagire con la collezione di prodotti nel database.**

**Questo modello è quello che utilizziamo per eseguire operazioni come find(), save(), findById() sui prodotti nel database. Grazie a ProductModel, possiamo leggere e scrivere i dati nel nostro database MongoDB. In altre parole: È il "ponticello" che collega il nostro codice con il database MongoDB**

**In breve:**

**Se usassi solo la classe Product senza il productSchema, non avresti la compatibilità necessaria per interagire con MongoDB in modo strutturato e sicuro, perché Mongoose richiede uno schema per gestire correttamente la creazione, il salvataggio e la lettura dei documenti nel database.**