# Architettura applicativa funzionalità

Richiesta Consumer

OrdineController

GetAllOrdiniByUserId

ManageOrdineBusiness

GetAllOrdiniByUserId

ManageOrdineData

GetAllOrdiniByUserId

IRepositoryOrdine

GetAllOrdiniByUserId

RepositoryOrdine

GetAllOrdiniByUserId

AcademyShopDB

-Ordine; -Prodotto;

-Stato\_Ordine; -Dettagli\_Ordine;

# Requisiti funionali metodo GetAllOriniByUserId

Get di tutti gli ordini **:** il metodo deve accettare in input l’**ID** dell’utente che intende ricercare tutti i suoi ordini.

Successivamente sarà necessario effettuare i seguenti passaggi:

* **query sul database:** recuperare tutti gli ordini relativi all’utente con l’**ID** di pertinenza.

Nello specifico, i dati da restituire sono i seguenti:

* DATA\_REGISTRAZIONE, tabella ORDINE
* DATA\_AGGIORNAMENTO, tabella ORDINE
* DESCRIZIONE (stato ordine), tabella STATO\_ORDINE

e la lista di tutti i prodotti collegati all’ordine in esame, ogni elemento della lista dovrà contenere :

* ID PRODOTTO, tabella DETTAGLIO\_ORDINE
* DESCRIZIONE, tabella PRODOTTO
* NOME, tabella PRODOTTO
* QUANTITA, tabella DETTAGLIO\_ORDINE
  + **Ritorno dati e gestione errori**: è necessario ritornare al consumer un JSON contenente i dati sopracitati, e status code 200. Nel caso in cui si verifichi un errore nell’esecuzione della query, occorre restituire status code 500.

## Api Layer

Qesto è il mio metodo nella classe OrdineController che restituisce il metodo presente nel business layer che viene richiamato grazie al campo oOBL.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Immagine che contiene testo, software, Software multimediale, Software per la grafica

Descrizione generata automaticamente

L'attributo **[HttpGet("GetAllOrdiniByUserId/{userId}")]** definisce la rotta per questa azione. Il segnaposto **{userId}** indica che **userId** sarà un parametro nella rotta. Il metodo restituisce un **Task<IActionResult>**, il che significa che è un metodo asincrono che restituisce un risultato che può essere un qualsiasi tipo di risultato di azione. Il metodo include un blocco **try-catch** per catturare eventuali eccezioni e restituire un errore 500 con un messaggio dettagliato.

[HttpGet("GetAllOrdiniByUserId/{userId}")]

public async Task<IActionResult> GetOrdiniByUserId(int userId)

{

try

{

var ordini = await oOBL.GetOrdiniByUserId(userId);

return Ok(ordini);

}

catch (Exception ex)

{

// Gestisci eventuali errori qui

return StatusCode(500, "Si è verificato un errore durante il recupero degli ordini: " + ex.Message);

}

}

## Business Layer

Nel business layer (ManageOrdineBusiness)si effettua la chiamata al metodo presente nel ManageOrdineData,tramite il campo oUDL , che poi effettua tutti i controlli necessari nel caso in cui si riscontri un errore .

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Sistema operativo

Descrizione generata automaticamente

Il metodo **GetOrdiniByUserId** esegue alcune operazioni importanti come la verifica dell'esistenza dell'utente e il recupero degli ordini.

public async Task<List<OrdineDettaglioDTOperGET>> GetOrdiniByUserId(int userId)

{

try

{

// Verifica se l'utente esiste

bool utenteExists = await oUDL.CheckUtenteExistsById(userId);

if (!utenteExists)

{

// Se l'utente non esiste, restituisci una risposta di errore 400 (Bad Request)

throw new ArgumentException();

}

// Chiama il metodo corrispondente del data layer per recuperare gli ordini dell'utente

var ordini = await oODL.GetOrdiniByUserId(userId);

if (ordini.Count == 0)

{

// Se non ci sono ordini, restituisci un messaggio appropriato

throw new InvalidOperationException();

}

// Se ci sono ordini, restituisci gli ordini dell'utente

return ordini;

}

catch (InvalidOperationException o)

{

throw new Exception("Non ci sono ordini per questo utente", o);

}

catch (ArgumentException u)

{

throw new Exception("Utente non trovato", u);

}

catch (Exception ex)

{

// Gestisci eventuali errori qui

throw new Exception("Errore durante il recupero degli ordini dell'utente.", ex);

}

}

## Data Layer o DAO

In questa classe ManageOrdineData faccio ritornare la firma del metodo presente nella classe IRepositoryOrdine, tramite il campo \_repositoryOrdine utile per la dipendenza esterna necessaria per effettuare operazioni di accesso e manipolazione dei dati degli ordini.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

public async Task<List<OrdineDettaglioDTOperGET>> GetOrdiniByUserId(int userId)

{

return await \_repositoryOrdine.GetOrdiniByUserId(userId);

}

## IRepository

L’interfaccia repository è progettata per definire operazioni relative alla gestione degli ordini.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente



## RepositoryOrdine

Questa classe **RepositoryOrdine** è un'implementazione concreta dell'interfaccia **IRepositoryOrdine**.

Immagine che contiene testo, software, schermata

Descrizione generata automaticamente

Qui è presente ache il metodo per verificare se l‘utente è presente nel DB

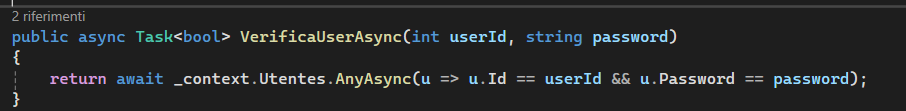


Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

public class RepositoryOrdine : IRepositoryOrdine

{

private readonly AcademyShopDBContext \_context;

public RepositoryOrdine(AcademyShopDBContext context)

{

\_context = context;

}

public async Task<List<OrdineDettaglioDTOperGET>> GetOrdiniByUserId(int userId)

{

try

{

var ordini = await \_context.Ordines

.Where(o => o.FkIdUtente == userId)

.Select(o => new OrdineDettaglioDTOperGET

{

ProdottoNome = o.DettaglioOrdines.First().FkIdProdottoNavigation.Nome,

DataRegistrazione = o.DataRegistrazione,

DataAggiornamento = o.DataAggiornamento,

StatoOrdineDescrizione = o.FkIdStatoNavigation.Descrizione,

ProdottoId = o.DettaglioOrdines.First().FkIdProdottoNavigation.Id,

ProdottoDescrizione = o.DettaglioOrdines.First().FkIdProdottoNavigation.Descrizione,

Quantita = o.DettaglioOrdines.First().Quantita

})

.ToListAsync();

return ordini; // Restituisce la lista degli ordini trovati.

}

catch (Exception ex)

{

// In caso di eccezione, solleva una nuova eccezione con un messaggio specifico.

throw new Exception("Non ci sono ordini per questo utente", ex);

}

}

Questo metodo implementa il metodo definito nell'interfaccia **IRepositoryOrdine**. Riceve come argomento l'ID dell'utente e restituisce una lista di oggetti **OrdineDettaglioDTOperGET** che rappresentano gli ordini relativi all'utente specificato.

esegue una query sul contesto del database **\_context** per recuperare gli ordini relativi a un determinato utente, identificato dall'ID **userId**. Dopo aver selezionato gli ordini, viene creata una lista di oggetti **OrdineDettaglioDTOperGET** utilizzando il metodo **Select**.

Utilizza l'operatore **Select** per mappare i risultati della query su oggetti **OrdineDettaglioDTOperGET**, che contiene i dettagli degli ordini per la visualizzazione.

Questa è la classe DTO (Creando una nuova Class Library(DtoLayer.Dto) )

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

## **Database**

Immagine che contiene testo, elettronica, schermata, software

Descrizione generata automaticamente

## **PROVE SWAGGER**

Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente

Immagine che contiene testo, software, schermata, Carattere

Descrizione generata automaticamente

Inserisco l’Id dell’utente

Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente

Mi restituisce la lista degli ordini dell’’utente con l’Id 1 e uno status code 200.

Se provo a inserire un Id utente presente nel database ma che non ha nessun ordine mi dovra restituire un Status code errore 500 con la scritta Ordini non presenti per questo utente.

Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente

Se inserisco un ID utente non presente nel database mi restituira Utente non presente

Immagine che contiene schermata, testo, software, linea

Descrizione generata automaticamente

.