**Documentazione del Progetto AcademyShop.**

Il progetto AcademyShop è stato sviluppato utilizzando il pattern a tre livelli (Three-tier architecture) combinato con il Repository Pattern. L'architettura a tre livelli suddivide l'applicazione nei seguenti livelli: Presentation Layer (Controller), Business Logic Layer (Business), e Data Access Layer (Data e Repository).

1. Struttura del Progetto

* UtenteController
* ManageUtenteBusiness
* ManageUtenteData
* RepositoryUtente
* IRepositoryUtente
* AcademyShopDB

1. Specifiche WEB API:

⦁ **Creazione di nuovo utente:** Il metodo per la creazione di un nuovo utente accetta in input un modello con le informazioni necessarie. Queste informazioni devono essere conformi alla struttura dati presente sul database, che include i seguenti campi: Cognome, Nome, DataNascita , CittaNascita, ProvinciaNascita, Sesso, Codice fiscale, Email, Password.

⦁ **Eliminazione utente:** Il metodo per l'eliminazione di un utente rimuove un utente esistente dal database utilizzando il suo ID univoco. Questo metodo richiede l'ID dell'utente da eliminare come parametro di input.

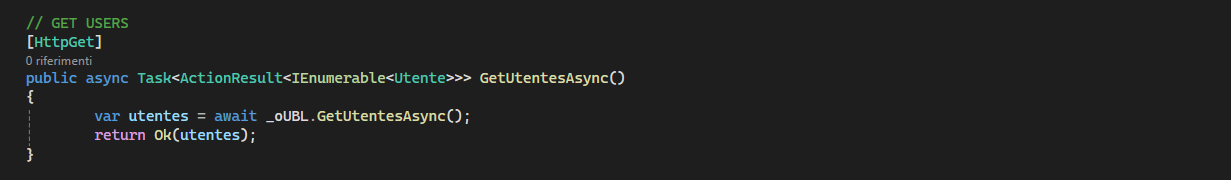
1. UtenteController:

La classe **UtenteController** è un controller API che espone le operazioni della gestione degli utenti.

Il costruttore di UtenteController accetta un'istanza di **ManageUtenteBusiness**, che viene utilizzata per chiamare i metodi del Business Layer.

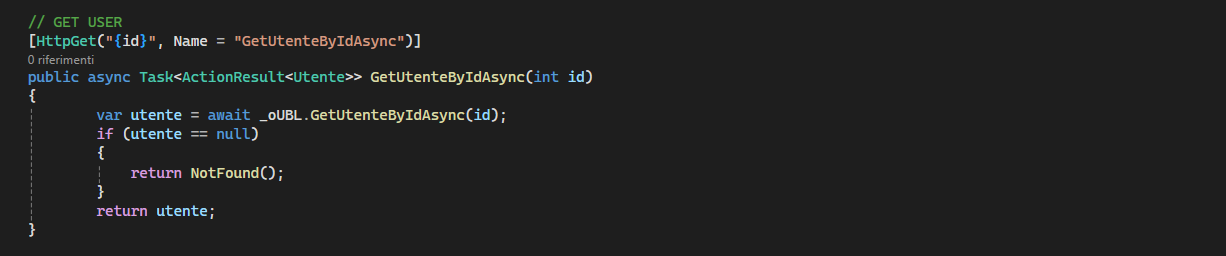


* **GetUtentesAsync**



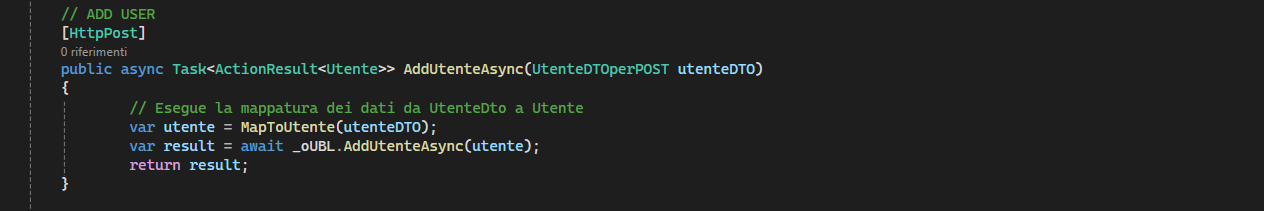
Questo metodo gestisce le richieste HTTP GET per recuperare la lista degli utenti. Utilizza il metodo GetUtentesAsync del business layer per ottenere i dati e restituisce una lista degli utenti.

* **GetUtenteByIdAsync**



Anche questo metodo gestisce le richieste HTTP GET per recuperare stavolta un utente specifico tramite il suo ID. Utilizza il metodo GetUtenteByIdAsync del Business Layer e restituisce l'utente richiesto. Se l'utente non viene trovato, restituisce un NotFound.

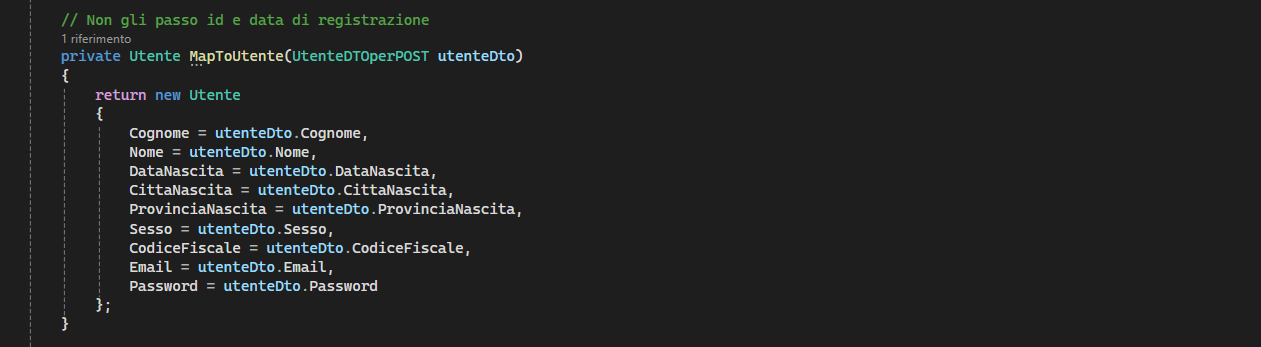
* **AddUtenteAsync**



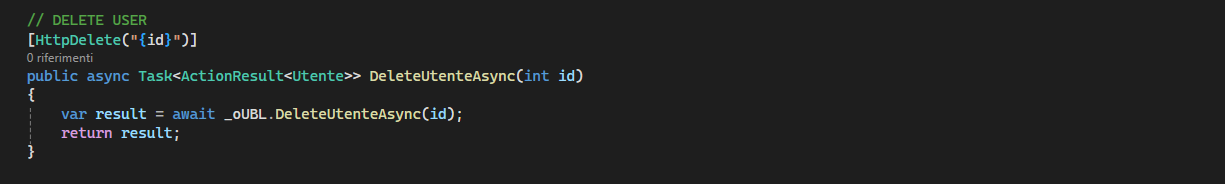
Questo metodo gestisce le richieste HTTP POST per creare un nuovo utente. Riceve un modello UtenteDTOperPOST, esegue la mappatura e utilizza il metodo AddUtenteAsync del Business Layer per aggiungere il nuovo utente. Restituisce il risultato dell'operazione attraverso l’uso del DTO per mappare solo i campi necessari da swagger senza “Id” e “DataRegistrazione”.

* **MapToUtente**

Ecco nel dettaglio il metodo che esegue la mappatura dei dati.



* **DeleteUtenteAsync**



Questo metodo gestisce le richieste HTTP DELETE per eliminare un utente tramite il suo ID. Utilizza il metodo DeleteUtenteAsync del Business Layer e restituisce il risultato dell'operazione.

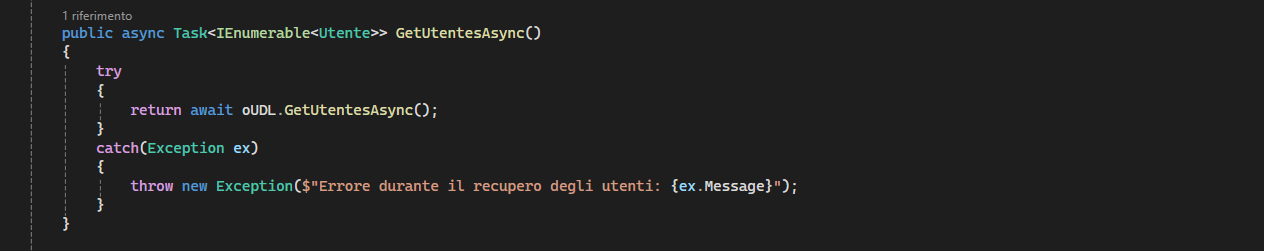
1. ManageUtenteBusiness

La classe ManageUtenteBusiness contiene la logica di business per la gestione degli utenti. Questa classe interagisce con il livello dati tramite la classe **ManageUtenteData**.

Il costruttore accetta un'istanza di **ManageUtenteData**, che viene utilizzata per accedere ai metodi del Data Layer.

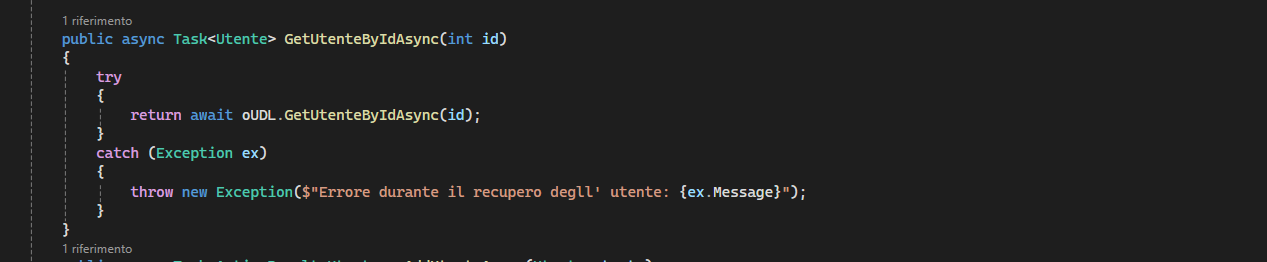


* **GetUtentesAsync**



Questo metodo recupera la lista degli utenti tramite il metodo GetUtentesAsync del Data Layer generando un’eccezione in caso di errore.

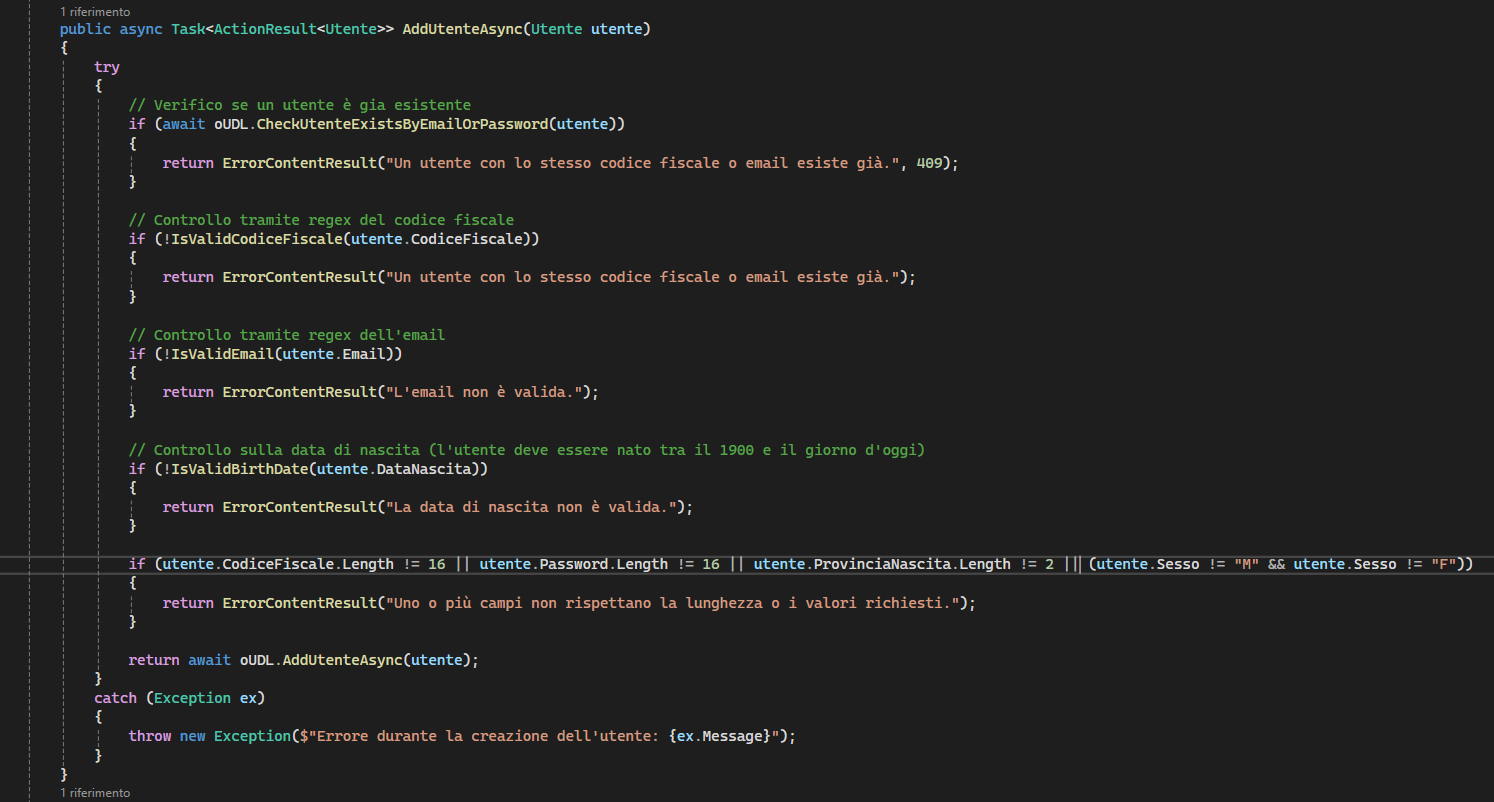
* **GetUtenteByIdAsync**



Questo metodo recupera un utente specifico tramite il suo ID utilizzando il metodo GetUtenteByIdAsync del Data Layer, viene generata un’eccezione in caso di errore.

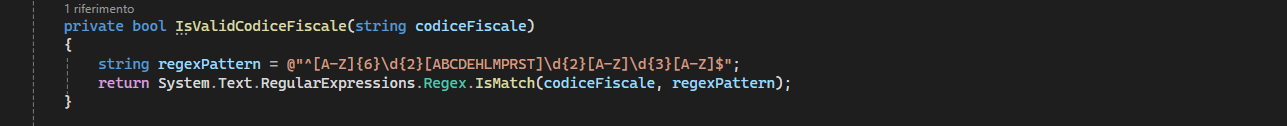
* **AddUtenteAsync**

Questo metodo aggiunge un nuovo utente dopo aver eseguito una serie di controlli di validità. Se l'utente esiste già, restituisce un errore. Inoltre, verifica la validità del codice fiscale, dell'email e della data di nascita, oltre i vari controlli di business.



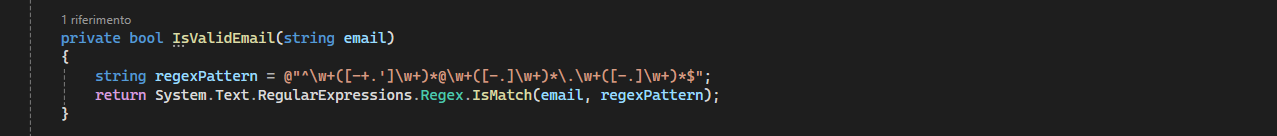
Vediamo nel dettaglio i metodi privati che vengono implementati in AddUtenteAsync per verificare la validità dei dati utente:

* **IsValidCodiceFiscale**



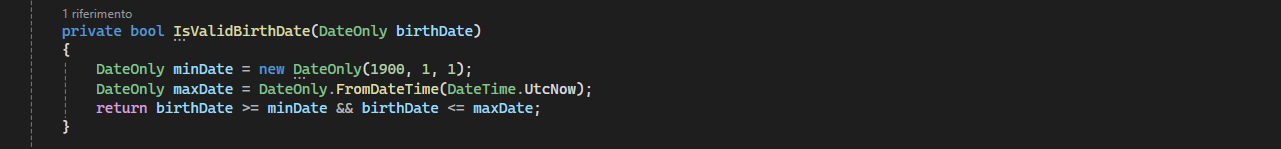
Questo metodo verifica se il codice fiscale fornito è valido. La validità del codice fiscale viene controllata utilizzando un'espressione regolare che corrisponde al formato standard del codice fiscale italiano.

* **IsValidEmail**



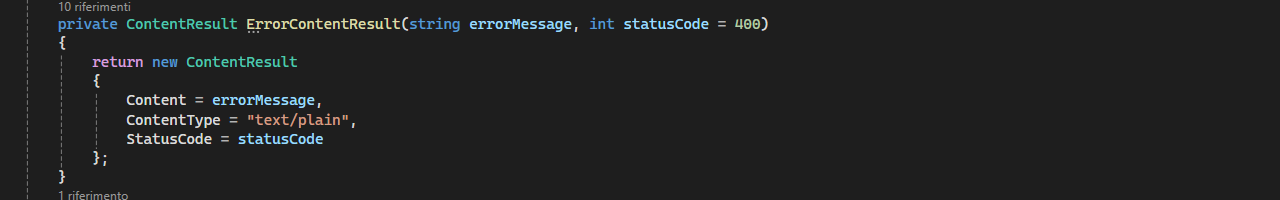
Questo metodo verifica se l'indirizzo email fornito è valido. L'indirizzo email viene controllato utilizzando un'espressione regolare che corrisponde ai formati standard degli indirizzi email.

* **IsValidBirthDate**



Questo metodo verifica se la data di nascita fornita è valida. La data di nascita viene controllata per assicurarsi che sia compresa tra il 1º gennaio 1900 e la data attuale.

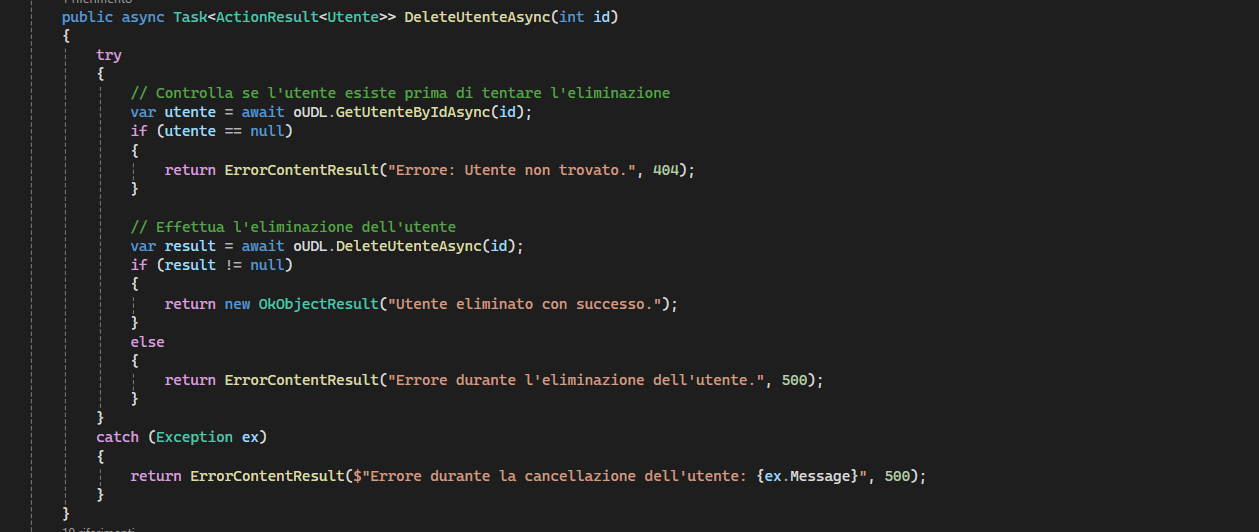
* **ErrorContentResult**



Questo metodo accetta due parametri: **errorMessage**, che rappresenta il messaggio di errore da restituire, e **statusCode**, che rappresenta il codice di stato HTTP associato al messaggio di errore (400 è il valore predefinito).

Il metodo restituisce un oggetto ContentResult contenente il messaggio di errore specificato. Questo oggetto viene poi restituito come risultato di una richiesta HTTP quando si verifica un errore nel sistema.

##### **DeleteUtenteAsync**

Il metodo DeleteUtenteAsync ha il compito dell'eliminazione di un utente specifico dal database. Riceve come parametro l'ID dell'utente che si desidera eliminare. Prima di procedere, viene effettuato un controllo per verificare se l'utente esiste effettivamente. 

4. ManageUtenteData

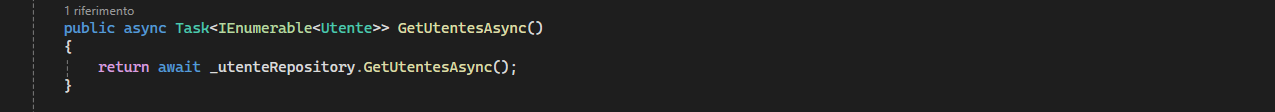
La classe ManageUtenteData funge da interfaccia tra il Business Layer e il Data Layer. Questa classe utilizza IRepositoryUtente per accedere ai metodi di accesso ai dati.

Il costruttore accetta un'istanza di IRepositoryUtente, che viene utilizzata per interagire con il repository.



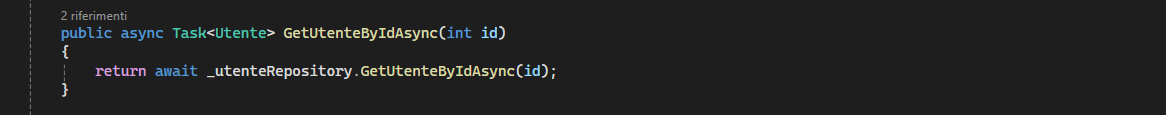
* **GetUtentesAsync**

Questo metodo recupera la lista degli utenti tramite il metodo GetUtentesAsync del repository.



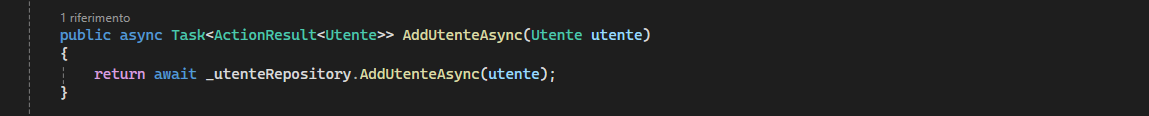
* **GetUtenteByIdAsync**

Questo metodo recupera un utente specifico tramite il suo ID utilizzando il metodo GetUtenteByIdAsync del repository.



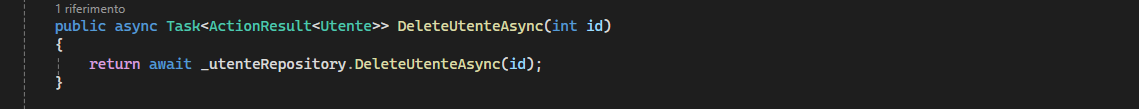
* **AddUtenteAsync**

Questo metodo aggiunge un nuovo utente utilizzando il metodo AddUtenteAsync del repository.



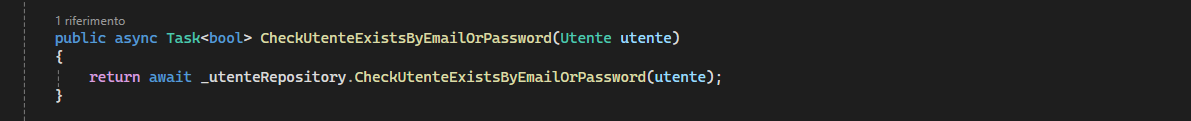
* **DeleteUtenteAsync**

Questo metodo elimina un utente specifico tramite il suo ID utilizzando il metodo DeleteUtenteAsync del repository.



* **CheckUtenteExistsByEmailOrPassword**

Questo metodo verifica se un utente con lo stesso codice fiscale o email esiste già utilizzando il metodo CheckUtenteExistsByEmailOrPassword del repository.



1. RepositoryUtente

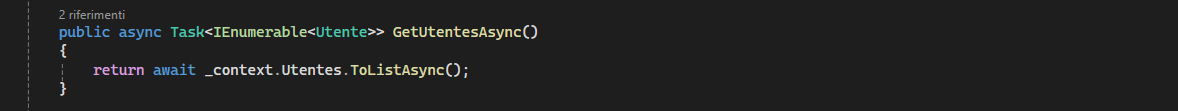
La classe RepositoryUtente implementa l'interfaccia IRepositoryUtente e fornisce i metodi per l'accesso ai dati relativi agli utenti. Questa classe interagisce direttamente con il database utilizzando Entity Framework Core.

Il costruttore accetta un'istanza di **AcademyShopDBContext**, che rappresenta il contesto del database.



* **GetUtentesAsync**

Questo metodo recupera la lista degli utenti dal database.



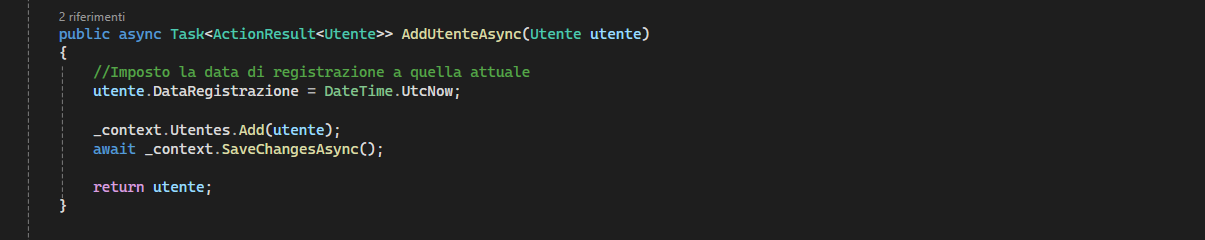
* **GetUtenteByIdAsync**

Questo metodo recupera un utente specifico tramite il suo ID dal database.



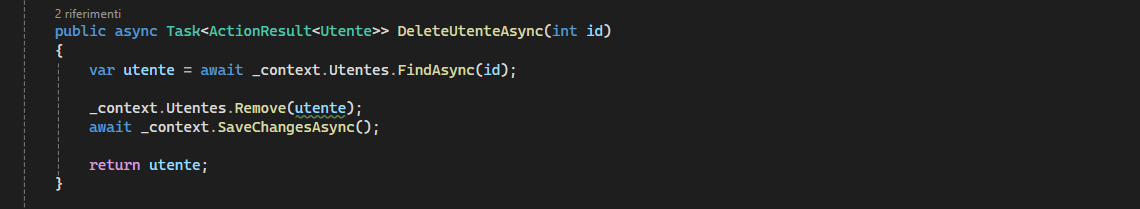
* **AddUtenteAsync**

Questo metodo aggiunge un nuovo utente al database e imposta la data di registrazione a quella attuale.



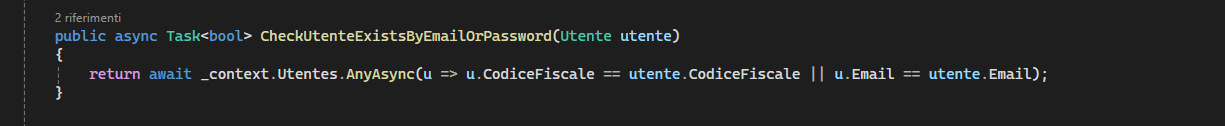
* **DeleteUtenteAsync**

Questo metodo elimina un utente specifico tramite il suo ID dal database.



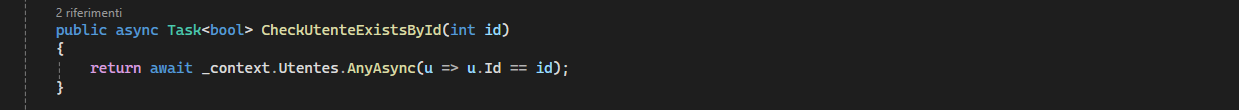
* **CheckUtenteExistsByEmailOrPassword**

Questo metodo verifica se un utente con lo stesso codice fiscale o email esiste già nel database.



* **CheckUtenteExistsById**

Questo metodo è utilizzato per verificare che l’utente già esista nel database attraverso il codice fiscale e l’email.



1. IRepositoryUtente

L'interfaccia IRepositoryUtente definisce i metodi necessari per interagire con il repository degli utenti. Oltre ai metodi standard per ottenere, aggiungere ed eliminare utenti, include anche un metodo per verificare l'esistenza di un utente in base all'email e alla password.

