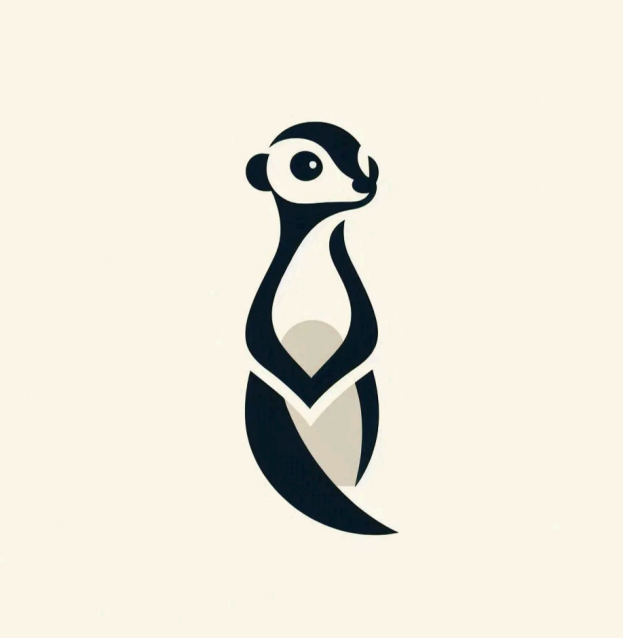
Test Unit Document

MeerKat

Test Unit Document

Versione 0.1



Coordinatore del progetto:

Nome	Matricola

Revision History

Data	Versione	Descrizione	Autore
13/02/2025	0.0.1	Stesura del documento	Gabriel Tabasco

Indice

1. Introduzione

1.1 Obiettivo del Documento

Questo documento descrive l'approccio adottato per la creazione e il miglioramento dei test unitari relativi al controller `UserController` di un'applicazione ASP.NET Core. L'obiettivo principale è garantire la correttezza del comportamento del sistema, identificando e risolvendo eventuali bug o problemi logici presenti nel codice sorgente.

1.2 Ambito di Applicazione

Il controller `UserController` gestisce operazioni CRUD per gli utenti e l'autenticazione tramite JWT. I test unitari analizzati riguardano la validità delle operazioni e la sicurezza dell'applicazione.

1.3 Riferimenti

- Documentazione ufficiale ASP.NET Core
- Principi di test unitari con NUnit

2. Descrizione Generale del Controller

2.1 Funzionalità Implementate

- **Recupero utenti:** Endpoint GET per ottenere tutti gli utenti o uno specifico.
- **Creazione utente:** Endpoint POST per registrare un nuovo utente.
- **Aggiornamento utente:** Endpoint PUT per modificare i dati di un utente esistente.
- **Eliminazione utente:** Endpoint DELETE per rimuovere un utente.
- **Autenticazione:** Endpoint POST per effettuare il login con email e password.

2.2 Requisiti di Sicurezza e Autenticazione

- Le operazioni CRUD richiedono autenticazione.
- La registrazione e il login sono accessibili senza autenticazione.
- Gli endpoint sensibili richiedono privilegi amministrativi o autorizzazione dell'utente interessato.

3. Problemi Identificati nei Test Unitari

3.1 Validazione Utente (UserValidator)

Problema: Il metodo `IsValid` non escludeva l'ID dell'utente corrente nel controllo di email duplicata, impedendo l'aggiornamento dei dati.

Correzione: Modifica della query per escludere l'ID dell'utente corrente:

```
if (await meerkatContext.Users.AnyAsync(u ⇒ u.Email == user.Email && u.I
```

```
d != user.Id)) return false;
```

3.2 Validazione della Password nella Creazione di un Nuovo Utente

Problema: La password veniva validata dopo l'hashing, rendendo impossibile rilevare password vuote o troppo corte.

Correzione: Spostata la validazione prima dell'hashing:

```
if (!ModelState.IsValid || !(await user.IsValid(meerkatContext))) { return Bad Request(ModelState); }
user.Password = HashingHelper.Hash(user.Password);
```

4. Approccio ai Test Unitari

4.1 Struttura dei Test

I test unitari sono stati implementati con NUnit utilizzando un contesto in memoria (`InMemoryContext`) per simulare il database.

4.2 Casi di Test Implementati

- **GetUsers_ReturnsOkResult_WhenUserIsAdmin**: Verifica che un amministratore possa recuperare tutti gli utenti.
- **GetUsers_ReturnsUnauthorizedResult_WhenUserIsNotAdmin**: Un utente senza privilegi non può accedere alla lista utenti.
- **CreateUser_ReturnsCreatedResult_WithValidData**: Un nuovo utente viene creato con dati validi.
- **CreateUser_ReturnsBadRequestResult_WithInvalidData**: Viene restituito `Bad Request` per dati invalidi.
- **UpdateUser_ReturnsOkResult_WithValidData_WhenUserIsAdmin**: Un amministratore può aggiornare un utente.
- **UpdateUser_ReturnsUnauthorizedResult_WhenUserIsNotAdminOrSelf**: Un utente non autorizzato non può aggiornare dati di altri utenti.
- **DeleteUser_ReturnsNoContentResult_WhenUserIsAdminOrSelf**: Un utente può eliminare il proprio account o un amministratore può eliminare

altri utenti.

- **DeleteUser_ReturnsUnauthorizedResult_WhenUserIsNotAdminOrSelf**: Un utente non autorizzato non può eliminare altri account.
- **UserLogin_ReturnsOkResult_WithCorrectCredentials**: Il login ha successo con credenziali corrette.
- **UserLogin_ReturnsUnauthorizedResult_WithIncorrectCredentials**: Il login fallisce con credenziali errate.

5. Risultati e Conclusioni

Le modifiche apportate al codice e l'introduzione dei test unitari hanno migliorato la robustezza del `UserController`. I test confermano che:

- Le operazioni CRUD funzionano come previsto.
- La validazione dei dati previene errori logici.
- L'autenticazione e l'autorizzazione sono gestite in modo sicuro.

Queste migliorie garantiscono una maggiore affidabilità e manutenibilità del codice nel tempo.