System Test Plan

MeerKat

System Test Plan

Versione 0.1.8

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Stefano Nicolò Zito	0512117457
Gabriel Tabasco	0512116905
Giuseppe Ballacchino	0512116596

Francesco Giuseppe Trotta	0512116788

Revision History

Data	Versione	Descrizione	Autore
14/12/2024	0.1	Scrittura indice e stesura Introduzione e Relationship to other documents	Stefano Nicolò Zito
14/12/2024	0.1.2	System overview e Features to be tested	Gabriel Tabasco
15/12/2024	0.1.3	Pass/Fails criteria e Approach	Francesco Giuseppe Trotta
15/12/2024	0.1.4	Approach e Suspension and resumption	Giuseppe Ballacchino
16/12/2024	0.1.5	Test cases, Test schedule e review	Francesco Giuseppe Trotta, Gabriel Tabasco, Stefano Nicolò Zito, Giuseppe Ballacchino
11/02/2025	0.1.6	Revisione Finale Pre-Delivery	Stefano Nicolò Zito, Francesco Giuseppe Trotta
04/03/2025	0.1.7	Revisione successiva agli errori esposti nella presentazione	Stefano Nicolò Zito, Francesco Giuseppe Trotta
21/03/2025	0.1.8	Aggiunta dei risultati dei test	Gabriel Tabasco

Indice

7. **Suspension and resumption**
8. **Testing materals**
9. **Test cases**
10. **Testing schedule**
11. Test Results

1. Introduction

I test da effettuare sul sistema hanno lo scopo di assicurare principalmente due cose:

- Che tutte le funzionalità richieste del sistema siano presenti e funzionanti.
- Che i design goal riguardanti performance e gestione dei dati siano stati rispettati.

Per questo, nella fase di testing è necessario dare priorità a testare il corretto e completo funzionamento dei sottosistemi, il tempo di risposta delle funzionalità di quest'ultimi e la corretta protezione dell'accesso ai dati e la trasparenza di locazione ad essi relativa.

Una volta che la presenza di queste caratteristiche è stata verificata, la rilevazione di difetti in altri aspetti del sistema può essere delegata a patching e a iterazioni successive del processo di testing, destinate a release future.

2. Relationship to other documents

I test descritti in questo documento sono, come già detto nell'introduzione, incentrati sui sottosistemi descritti nel SDD. I test che verranno descritti saranno basati sul trovare difetti all'interno dei servizi e funzionalità interne di ciascun sottosistema e quindi, affianco a ogni test case, sarà specificato il sottosistema di cui gli elementi testati fanno parte.

Inoltre, i test descritti serviranno anche a valutare la qualità del software in relazione ai Design Goal e quindi ai relativi Non-Functional Requirements descritti nel RAD; anche in questo caso, se i casi di test fanno riferimento a particolari requisiti o goal, questi verranno esplicitati nella definizione del test case.

3. System overview

L'elenco delle unità di sistema usate per il testing individuale di unità sono:

- **UserUnit,** comprende le classi usate nella gestione del profilo degli utenti: User, UserModelView e UserView. Questa unità va testata per prima.
- **TeamUnit,** comprende le classi usate nella gestione dei Team: Team, TeamModelView, TeamView. Questa unità va testata per seconda in quanto tutte le altre dipendono da questa in qualche misura, tranne la UserUnit da cui dipende questa;
- **TaskUnit,** comprende le classi usate nella gestione della singola task: Task, TaskViewModel;
- **TaskListUnit,** comprende le classi usate nella gestione delle liste di Task, è dipendente dalla TaskUnit: TaskList, TaskListViewModel, TaskView;
- **ChatUnit,** comprende le classi usate nell'implementazione delle chatroom: ChatRoom, ChatViewModel, ChatView;
- **StatUnit,** comprende le classi usate nella gestione delle statistiche di lavoro: Stat, StatViewModel, StatView;

4. Features to be tested

A livello di funzionalità, gli aspetti del sistema che verranno testati sono:

- Registrazione Utente
- Creazione Team
- Creazione Task list
- Creazione Task
- Consegna Task

5. Pass/Fails criteria

Per ogni test case, definiti più avanti si seguono i seguenti criteri generali in modo da stabilire se il test è andato a buon fine o no:

- Un test va a buon fine se si presenta un Failure nel sistema e si individua almeno genericamente il Fault che lo ha causato.

- Un test va a buon fine se si individua una parte di sistema che non rispetta i requisiti di performance, quindi con tempi di risposta elevati
- Un test va a buon fine se si individua una parte del sistema che non rispetta i requisiti riguardanti il corretto accesso ai dati

6. Approach

Il processo di testing seguirà un approccio incrementale e modulare, concentrandosi inizialmente sui test delle singole unità del sistema per poi progredire verso test di integrazione e infine sul testing di sistema, seguendo uno schema di tipo Bottom-Up.

Unit Testing

Questa fase si incentrerà sul testing di unità singole del sistema in particolare test basati sulla validazione dei dati e su classi Helper, non verrà effettuato su operazioni di persistenza siccome venendo il DB gestito tramite una libreria ORM non si sarebbe riscontrato nulla se non un malfunzionamento della libreria stessa.

In questa fase si utilizzeranno test automatici basati su framework di testing compatibili con il linguaggio e il framework di sviluppo (.NET e C#).

Integration Testing

Una volta verificata l'affidabilità delle singole unità, si procederà al testing delle interazioni tra i sottosistemi. L'approccio all'integrazione seguirà uno schema incrementale:

- Saranno testate inizialmente le unità con meno dipendenze (ad esempio, **UserUnit**) e progressivamente quelle più complesse, come **TaskListUnit** e **ChatUnit**. Partendo dal testare le singole unità con l’integrazione della persistenza.
- Verranno utilizzati scenari di test definiti per simulare flussi reali di utilizzo del sistema, garantendo che i sottosistemi lavorino correttamente in combinazione tra loro.

System Testing

Il sistema sarà testato nella sua interezza per verificare che tutte le funzionalità richieste siano presenti e funzionanti in modo integrato.

Automazione e Monitoraggio

- Saranno utilizzati strumenti di automazione per garantire la ripetibilità e l'efficienza dei test.

Testing di Regressione

Ogni volta che il codice sarà modificato, verranno effettuati test di regressione per assicurare che le modifiche non abbiano introdotto nuovi difetti e che le funzionalità esistenti continuino a funzionare correttamente.

7. Suspension and Resumption

La sospensione del testing può essere svolta solo in casi di modifiche al codice da apportare che si scoprono durante la fase di testing, conseguentemente a un testing o indipendentemente da esso per altre ragioni tecnologiche o di design.
Alla ripresa della fase di testing va eseguito testing di regressione sulle componenti modificate.

8. Testing Materials

Per il testing del sistema a livello hardware è necessaria una macchina multicore dotata di connessione a internet, sulla quale devono essere presenti i seguenti elementi software: un IDE e compilatore in grado di eseguire codice C# e le componenti necessarie all'uso del framework .NET, preferibilmente l'IDE Rider.

Inoltre la macchina deve essere dotata delle librerie e plugin per il corretto utilizzo del framework Avalonia UI.

9. Testing Cases

UC1: Registration

Parametri: email - password - data di nascita

Categorie: relativamente ai parametri distinguiamo queste categorie:

1. L'Email non esiste nel database e finisce con il pattern @provider.com
2. la password contiene più di 8 caratteri

3. la data di nascita indica che l'utente ha almeno 18 anni

Scelte e Vincoli: le possibili scelte e vincoli per le categorie sono i seguenti:

1. email
    a. Email non esiste nel database e finisce con il pattern @provider.com, l'Email è corretta
    b. Email esiste nel database, l'Email non è corretta
    c. Email non esiste nel database e non finisce con il pattern @provider.com, L'Email non è corretta
2. password
    a. La password ha più di 8 caratteri, la password è corretta
    b. La password ha meno di 8 caratteri, la password non è corretta
3. data di nascita
    a. la data di nascita e avvenuta almeno 18 anni fa, la data di nascita è corretta
    b. la data di nascita e avvenuta di 18 anni fa, la data di nascita non è corretta

Test frame:

- TF1: Email corretta, Password Corretta, Data di nascita corretta → ORACOLO: l'utente viene aggiunto al database e ciò è notificato all'utente con un messaggio
- TF2: Email corretta, Password corretta, Data di nascita non corretta → ORACOLO: Mostrato un messaggio di errore al utente nella pagina di registrazione
- TF3: Email corretta, Password non corretta, Data di nascita corretta → ORACOLO: Mostrato un messaggio di errore al utente nella pagina di registrazione
- TF4: Email non corretta, Password corretta, Data di nascita corretta → ORACOLO: Mostrato un messaggio di errore al utente nella pagina di registrazione

Test case:

- TC1 - TF1 :

    - *Parametri:* “JohnDoe@gmail.com”, “Password1”, “01/01/2000” (email non presente nel DB)
    - *Risultati: utente inserito nel database e notificato ciò al utente*
- TC2 - TF2:
    - *Parametri:* “JohnDoe@gmail.com”, “Password1”, “01/01/2024” (email non presente nel DB)
    - *Risultati:* Mostrato un messaggio di errore al utente nella pagina di registrazione
- TC3 - TF3:
    - *Parametri:* “JohnDoe@gmail.com”, “Pass”, “01/01/200” (email non presente nel DB)
    - *Risultati:* Mostrato un messaggio di errore al utente nella pagina di registrazione
- TC4 - TF4:
    - *Parametri:* “JohnDoe@gmail.com”, “Password1”, “01/01/200” (email presente nel database)
    - *Risultati:* Mostrato un messaggio di errore al utente nella pagina di registrazione

---

UC 2: Create Team

Parametri: descrizione - deadline

Categorie: relativamente ai parametri distinguiamo queste categorie:

1. la descrizione ha una lunghezza inferiore a 500 caratteri.
2. la deadline del progetto non è una data precedente a quella attuale

Scelte e Vincoli: le possibili scelte e vincoli per le categorie sono i seguenti:

1. descrizione
    a. la descrizione è lunga meno di 500 caratteri, descrizione corretta
    b. la descrizione è lunga piò di 500 caratteri, descrizione non corretta
2. deadline
    a. la deadline è una data futura rispetta a quella attuale, deadline corretta

b. la deadline è una data passata rispetta a quella attuale, deadline non corretta

Test frame:

- TF1: descrizione corretta, deadline corretta → ORACOLO: Team aggiunto al database e user portato alla pagina di selezione dei Team
- TF2: descrizione corretta, deadline non corretta → ORACOLO: messaggio di errore mostrato al utente
- TF3: descrizione non corretta, deadline corretta → ORACOLO: messaggio di errore mostrato al utente
- TF4: descrizione non corretta, deadline non corretta → ORACOLO: messaggio di errore mostrato al utente

Test case:

- TC1 - TF1 :
  - *Parametri:* "Questo team è stato creato come un esempio per il testing.", "13/04/2027"
  - *Risultati:* Team aggiunto al database e user portato alla pagina di selezione dei Team
- TC2 - TF2:
  - *Parametri:* "Questo team è stato creato come un esempio per il testing.", "13/04/2021"
  - *Risultati:* messaggio di errore mostrato al utente
- TC3 - TF3:
  - *Parametri:* "Questo team è stato creato come un esempio per il testing... (lunghezza maggiore della specifica)", "13/04/2027"
  - *Risultati:* messaggio di errore mostrato al utente
- TC4 - TF4:
  - *Parametri:* "Questo team è stato creato come un esempio per il testing... (lunghezza maggiore della specifica)", "13/04/2021"
  - *Risultati:* messaggio di errore mostrato al utente

UC5: Create TaskList

Parametri: titolo - descrizione - deadline

Categorie: relativamente ai parametri distinguiamo queste categorie:

1. la lunghezza del titolo non deve superare 100 caratteri
2. la descrizione ha una lunghezza inferiore a 500 caratteri.

Scelte e Vincoli: le possibili scelte e vincoli per le categorie sono i seguenti:

1. titolo
    a. il titolo è lungo meno di 100 caratteri, titolo corretto
    b. il titolo è lungo più di 100 caratteri, titolo non corretto
2. descrizione
    a. la descrizione è lunga meno di 500 caratteri, descrizione corretta
    b. la descrizione è lunga più di 500 caratteri, descrizione non corretta

Test frame:

- TF1: titolo corretto, descrizione corretta → ORACOLO: TaskList aggiunta al database e user portato alla pagina di gestione task
- TF2: titolo non corretto, descrizione corretta → ORACOLO: messaggio di errore mostrato al utente
- TF3: titolo corretto, descrizione non corretta → ORACOLO: messaggio di errore mostrato al utente
- TF4: titolo non corretto, descrizione non corretta → ORACOLO: messaggio di errore mostrato al utente

Test case:

- TC1 - TF1 :
    - *Parametri:* "Tasklist d'esempio", "Questo task list è stato creato come un esempio per il testing."
    - *Risultati:* Tasklist aggiunto al database e user portato alla pagina di selezione dei Team
- TC2 - TF2:
    - *Parametri:* "Tasklist d'esempio...(lunghezza superiore a specifica)", "Questo task list è stato creato come un esempio per il testing."
    - *Risultati:* messaggio di errore mostrato al utente

- TC3 - TF3:
  - *Parametri:* “Tasklist d’esempio”, “Questo task list è stato creato come un esempio per il testing... (lunghezza maggiore della specifica)”
  - *Risultati:* messaggio di errore mostrato al utente
- TC4 - TF4:
  - *Parametri:* “Tasklist d’esempio...(lunghezza superiore a specifica)”, “Questo task list è stato creato come un esempio per il testing... (lunghezza superiore a specifica)”
  - *Risultati:* messaggio di errore mostrato al utente

UC4: add Task

Parametri: titolo - descrizione - deadline

Categorie: relativamente ai parametri distinguiamo queste categorie:

1. la lunghezza del titolo non deve superare 100 caratteri
2. la descrizione ha una lunghezza inferiore a 500 caratteri.
3. la deadline del progetto non è una data precedente a quella attuale

Scelte e Vincoli: le possibili scelte e vincoli per le categorie sono i seguenti:

1. titolo
   a. il titolo è lungo meno di 100 caratteri, titolo corretto
   b. il titolo è lunga più di 100 caratteri, titolo non corretto
2. descrizione
   a. la descrizione è lunga meno di 500 caratteri, descrizione corretta
   b. la descrizione è lunga più di 500 caratteri, descrizione non corretta
3. deadline
   a. la deadline è una data futura rispetta a quella attuale, deadline corretta
   b. la deadline è una data passata rispetta a quella attuale, deadline non corretta

Test frame:

- TF1: titolo corretto, descrizione corretta, deadline corretta → ORACOLO: Team aggiunto al database e user portato alla pagina di selezione dei Team
- TF2: titolo corretto, descrizione corretta, deadline non corretta → ORACOLO: messaggio di errore mostrato al utente
- TF3: titolo corretto, descrizione non corretta, deadline corretta → ORACOLO: messaggio di errore mostrato al utente
- TF4: titolo non corretto, descrizione corretta, deadline corretta → ORACOLO: messaggio di errore mostrato al utente

Test case:

- TC1 - TF1 :
  - *Parametri:* "Task d'esempio", "Questo task è stato creato come un esempio per il testing.", "13/04/2027"
  - *Risultati:* Team aggiunto al database e user portato alla pagina di selezione dei Team
- TC2 - TF2:
  - *Parametri:* "Task d'esempio", "Questo task è stato creato come un esempio per il testing.", "13/04/2021"
  - *Risultati:* messaggio di errore mostrato al utente
- TC3 - TF3:
  - *Parametri:* "Task d'esempio", "Questo task è stato creato come un esempio per il testing... (lunghezza maggiore della specifica)", "13/04/2027"
  - *Risultati:* messaggio di errore mostrato al utente
- TC4 - TF4:
  - *Parametri:* "Task d'esempio...(lunghezza superiore a specifica)", "Questo task è stato creato come un esempio per il testing.", "13/04/2027"
  - *Risultati:* messaggio di errore mostrato al utente

UC7: Task Delivery

Parametri: data

Categorie: relativamente ai parametri distinguiamo queste categorie:

1. la data odierna è precedente alla deadline della task

Scelte e Vincoli: le possibili scelte e vincoli per le categorie sono i seguenti:

1. data
    a. la data è precedente alla data segnata come deadline, consegna in tempo
    b. la data non è precedente alla data segnata come deadline, consegna in ritardo

Test frame:

- TF1: consegna in tempo → ORACOLO: task aggiornata allo stato consegnata in orario
- TF2: consegna in ritardo → ORACOLO: task aggiornata allo stato consegnata in ritardo l'utente e notificato con un messaggio a schermo.

Test case:

- TC1 - TF1 :
    - *Parametri: data odierna,* precedente alla deadline
    - *Risultati:* task aggiornata allo stato consegnata in orario
- TC2 - TF2:
    - *Parametri: data odierna,* successiva alla deadline
    - *Risultati:* task aggiornata allo stato consegnata in ritardo l'utente e notificato con un messaggio a schermo.

---

10. Testing schedule

Fase 1: Preparazione dell'ambiente di test

- **Obiettivo:** Configurare l'ambiente hardware, software e database necessario per eseguire i test.
- **Attività:**
    - Installazione di Meerkat su piattaforme target (Windows, MacOS, Linux, App mobile).

    - Configurazione del server backend e database.
    - Creazione di account utente di base per test.
- **Durata stimata:** 1-2 giorni.
- **Responsabili:** Team DevOps e QA Lead.

Fase 2: Test funzionali di base (User Flow iniziale)

2.1. Registrazione utente

- **Test Case:** UC1 Registration
- **Dipendenze:** Nessuna.
- **Durata stimata:** 1 giorno.
- **Risorse necessarie:** 1 QA tester.
- **Attività:**
    - Verifica dell'inserimento corretto dei dati.
    - Test di input invalidi (e.g., email non valida, password non corrispondenti).

2.2. Creazione di un team

- **Test Case:** UC 2 Create Team
- **Dipendenze:** UC1 Registration (Utente registrato come Project Manager).
- **Durata stimata:** 1-2 giorni.
- **Risorse necessarie:** 1 QA tester.
- **Attività:**
    - Verifica della creazione del team con dettagli validi.
    - Test di input mancanti o errati.

Fase 3: Test funzionali avanzati (Task Management e Messaging)

3.1. Creazione di task

- **Test Case:** UC5 create Tasklist, UC4 add Task

- **Dipendenze:** UC2 Create Team (Il team deve esistere per poter creare una task).
- **Durata stimata:** 2 giorni.
- **Risorse necessarie:** 1 QA tester.
- **Attività:**
  - Verifica della creazione di task in una lista esistente.
  - Creazione di nuove liste di task.
  - Test di modifica e rimozione di task.

3.2. Completamento di task

- **Test Case:** UC7 TaskDelivery
- **Dipendenze:** UC4 add Task (Per poter concludere un task, questo deve esister a priori).
- **Durata stimata:** 1 giorno.
- **Risorse necessarie:** 1 QA tester.
- **Attività:**
  - Controllo dello stato del task come "coinsegnato in ritardo" o “consegnato in tempo”.

Fase 4: Test di sistema (integrazione e prestazioni)

4.1. Test di integrazione

- **Obiettivo:** Verificare il flusso end-to-end tra registrazione, creazione del team, task e messaggistica.
- **Dipendenze:** I Test Case riportati nella Fase 3 non devono aver riportato grandi problematiche
- **Durata stimata:** 2-3 giorni.
- **Risorse necessarie:** 2 QA tester.
- **Attività:**
  - Creazione di un team con task e verifiche di messaggi tra membri.

- Test delle interazioni tra task completati e visualizzazione del progresso.

4.2. Test di prestazioni

- **Obiettivo:** Valutare la scalabilità e la risposta del sistema.
- **Durata stimata:** 3 giorni.
- **Risorse necessarie:** 1 QA tester, 1 DevOps.
- **Attività:**
  - Simulazione di carico con 100, 500 e 1000 utenti simultanei.
  - Verifica del comportamento con numerosi task e messaggi.

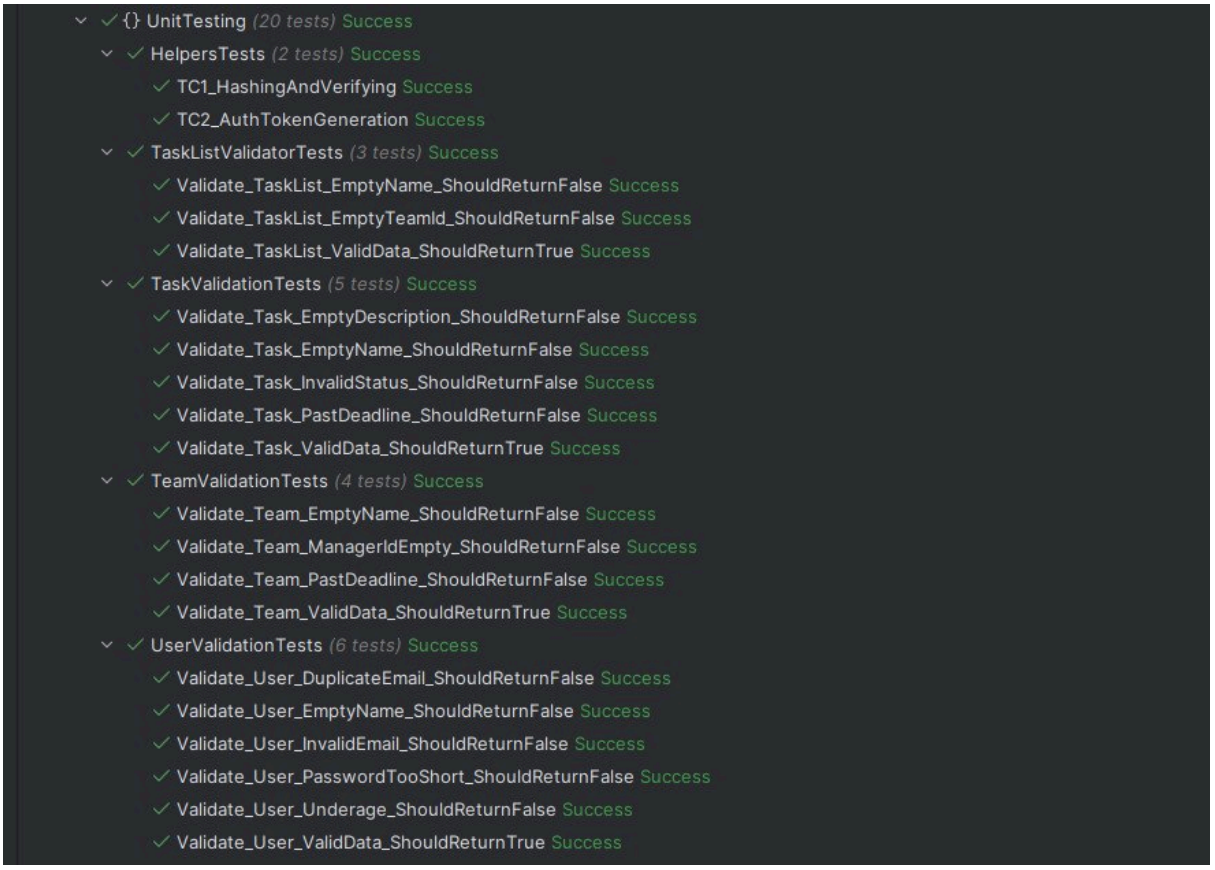
Sintesi del Piano

Fase	Attività	Durata	Risorse
Preparazione ambiente	Configurazione piattaforme e database	1-2 giorni	DevOps, QA
Test funzionali di base	Registrazione, creazione team	3 giorni	1 QA
Test avanzati	Task, completamento, messaggistica	6 giorni	1 QA
Test di sistema	Integrazione e prestazioni	5-6 giorni	2 QA, 1 DevOps

Totale stimato: 15 giorni lavorativi.

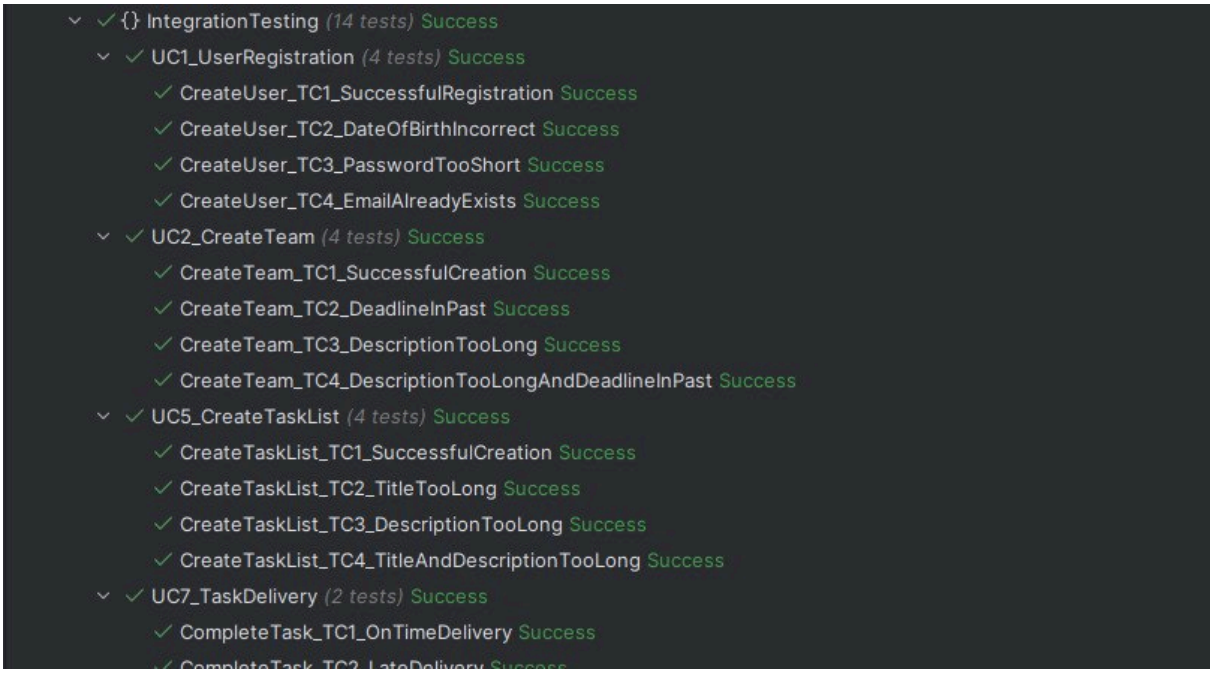
---

11. Test Results



Gli Unit Test hanno evidenziato alcuni problemi di validazione dei dati, che una volta risolti hanno permesso la corretta interazione dei Controller con la Persistenza.

È stato evidenziato anche il corretto funzionamento dei token di autenticazione e hashing delle password, i quali problemi sarebbero stati molto più complessi da individuare.



Gli Integration Test, successivi alle correzioni degli Unit Test, non hanno evidenziato particolari problemi, garantendo sicurezza sulla manipolazione dei dati e sulla loro interazione con il layer di persistenza.