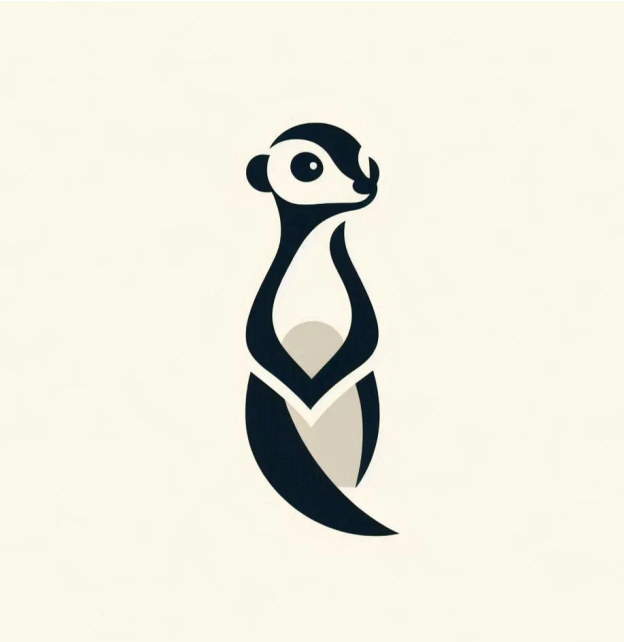
System Test Plan

MeerKat

System Test Plan

Versione 0.1.5



Coordinatore del progetto:

Nome	Matricola

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

Indice

1. Introduction

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

- Che tutte le funzionalità di base 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 funzionamento completo 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:

- **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;

- **UserUnit,** comprende le classi usate nella gestione del profilo degli utenti: User, UserModelView e UserView;
- **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 di sistema, gli aspetti del sistema che vanno testati sono:

- gestione dei task e delle liste di task (creazione, eliminazione, modifiche di entrambe) (da FR5 a FR8),
- invio e ricezione di messaggi in chat individuale,
- invio e ricezione di messaggi nelle chat di gruppo,
- raccolta, elaborazione e presentazione di statistiche riguardanti i tempi di lavoro (da FR10 a FR13),
- creazione, eliminazione e modifica di team e dei membri di quest'ultimi (da FR1 a FR5)

5. Pass/Fails criteria

Per ogni test case, definito più avanti nel sistema 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, test di sistema completi.

Unit Testing

Ogni sottosistema sarà testato singolarmente, verificando il corretto funzionamento delle sue funzionalità principali. In questa fase si utilizzeranno test automatici basati su framework di testing compatibili con il linguaggio e il framework di sviluppo (.NET e C#).

- Le unità saranno testate nell'ordine di dipendenza: inizieremo con **UserUnit**, seguita da **TeamUnit**, e così via, rispettando la gerarchia delle dipendenze indicate nella panoramica del sistema.
- Verranno verificati i comportamenti attesi delle classi e dei metodi, nonché il rispetto dei requisiti funzionali.

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**.
- 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.

- Questa fase includerà test sulle prestazioni, per verificare i tempi di risposta dei sottosistemi, e test di conformità ai requisiti non funzionali, come la protezione dei dati e la trasparenza di localizzazione.
- Si eseguiranno anche test di accettazione per confrontare i risultati ottenuti con i Design Goal definiti nel **RAD**.

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.

9. Testing Cases

Test Case: User Registration

Test Case Specification Identifier: TC-Registration-01

SubSystem: UserManagement.

Test Items:

- UserUnit

Input Specifications:

- Nome: "Mario"
- Cognome: "Rossi"
- Email: "mario.rossi@example.com"
- Data di nascita: "01/01/1985"
- Password: "Password123!"
- Conferma password: "Password123!"

Output Specifications:

- Utente registrato con successo.
- Password crittografata correttamente
- Invio di un'email di conferma all'indirizzo fornito.

Environmental Needs:

- Applicazione Desktop
- Server di backend funzionante con connessione a internet.

Special Procedural Requirements:

- Testa con dati validi e con combinazioni di dati mancanti o invalidi (e.g., email non valida, password non corrispondenti).

Intercase Dependencies:

- Nessuna.

Test Case: User Login

Test Case Specification Identifier: TC-Login-01

SubSystem: UserManagement.

Test Items:

- UserUnit

Input Specifications:

- Email: "mario.rossi@example.com"
- Password: "Password123!"

Output Specifications:

- Accesso effetuato con successo.
- ridirezione alla pagine di gestione deii progetti.

Environmental Needs:

- Applicazione Desktop
- Server di backend funzionante con connessione a internet.

Special Procedural Requirements:

- Testa con dati validi e con combinazioni di dati mancanti o invalidi (e.g., email non valida, password non corrispondenti).

Intercase Dependencies:

- Dipende dal caso di test TC-Registration-01

Test Case 4.1.2: Team Creation

Test Case Specification Identifier: TC-TeamCreation-01

SubSystem: TeamManagement.

Test Items:

- TeamUnit

Input Specifications:

1. Nome: "Team A"

2. Descrizione: "Progetto software remoto."
3. Immagine del team: Upload di un'immagine PNG.
4. Data di consegna: "30/12/2024".
5. Email dei collaboratori:
    - "filippo.bianchi@example.com"
    - "lucia.neri@example.com"
    - "andrea.verdi@example.com"
    - "giulia.rossi@example.com"

Output Specifications:

- Creazione del team con i dettagli specificati.
- Inviti inviati ai collaboratori via notifiche in-app.

Environmental Needs:

- Applicazione Desktop
- Server di backend funzionante con connessione a internet.

Special Procedural Requirements:

- Testa anche con team senza collaboratori aggiunti.
- Verifica i casi di errore, come email non valide o duplicati.

Intercase Dependencies:

- Dipende dal caso di test TC-Login-01.

Test Case 4.1.3: Task Creation

Test Case Specification Identifier: TC-TaskCreation-01

SubSystem: TaskManagement

Test Items:

- TaskUnit
- TaskListUnit

Input Specifications:

1. Nome del task: "Creare logo".
2. Descrizione: "Disegnare un'icona per l'applicazione".
3. Data di scadenza: "31/12/2024".
4. Nome della nuova lista: "Long-Term Tasks".

Output Specifications:

- Il task viene aggiunto alla lista corretta.
- La nuova lista è visibile nella pagina di gestione dei task.

Environmental Needs:

- Account autenticato con accesso al team creato in TC-TeamCreation-01.

Special Procedural Requirements:

- Testa l'aggiunta di task a una lista esistente e la creazione di una nuova lista.
- Verifica la possibilità di modificare o eliminare task e liste.
- Testare con dati invalidi, come dati precedenti a quella attuale.

Intercase Dependencies:

- Dipende dal caso di test TC-TeamCreation-01.

Test Case 4.1.4: Task Completion

Test Case Specification Identifier: TC-TaskCompletion-01

SubSystem: TaskManagement

Test Items:

- TaskUnit

Input Specifications:

1. Nome del file consegnato: "logo.png".

Output Specifications:

- Task segnato come completato.

- File caricato e associato al task completato.

Environmental Needs:

- Accesso al task creato in TC-TaskCreation-01.
- Accesso alla funzionalità di caricamento file.

Special Procedural Requirements:

- Verifica i casi di errore durante il caricamento del file (e.g., file corrotto, formato non supportato).

Intercase Dependencies:

- Dipende da TC-TaskCreation-01 per l'esistenza del task.

Test Case 4.1.5: Group Message

Test Case Specification Identifier: TC-GroupMessage-01

SubSystem: Chat Management

Test Items:

- ChatUnit

Input Specifications:

1. Nome del topic: "Meeting 20-10-2024".
2. Messaggio: "Il meeting è previsto per lunedì alle 14:00".

Output Specifications:

- Il topic è creato correttamente.
- Il messaggio appare nella chat di gruppo.

Environmental Needs:

- Ambiente configurato con un team esistente e collaboratori.

Special Procedural Requirements:

- Testa anche l'invio di messaggi vuoti o troppo lunghi.

Intercase Dependencies:

- Dipende da TC-TeamCreation-01 per l’esistenza del team.

Test Case 4.1.6: Individual Message

Test Case Specification Identifier: TC-IndividualMessage-01

SubSystem: Chat Management

Test Items:

- ChatUnit

Input Specifications:

1. Destinatario: "Mario Rossi".
2. Messaggio: "Ho bisogno di chiarimenti sul task".

Output Specifications:

- Messaggio inviato con successo.
- Stato del messaggio passa da "inviato" a "visualizzato".

Environmental Needs:

- Ambiente configurato con utenti registrati e autenticati.

Special Procedural Requirements:

- Verifica anche i casi di messaggi non inviati per connessione instabile.

Intercase Dependencies:

- Dipende da TC-TeamCreation-01 per l’esistenza del team.

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:** 2 giorni.
- **Responsabili:** Team DevOps e QA Lead.

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

2.1. Registrazione utente

- **Test Case:** TC-Registration-01
- **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).
  - Controllo del flusso di conferma via email.

2.2. Creazione di un team

- **Test Case:** TC-TeamCreation-01
- **Dipendenze:** TC-Registration-01 (utente registrato).
- **Durata stimata:** 2 giorni.
- **Risorse necessarie:** 1 QA tester.
- **Attività:**
  - Verifica della creazione del team con dettagli validi.
  - Test di input mancanti o errati.
  - Controllo della funzionalità di invito.

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

3.1. Creazione di task

- **Test Case:** TC-TaskCreation-01
- **Dipendenze:** TC-TeamCreation-01 (team esistente).
- **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:** TC-TaskCompletion-01
- **Dipendenze:** TC-TaskCreation-01 (task esistente).
- **Durata stimata:** 1 giorno.
- **Risorse necessarie:** 1 QA tester.
- **Attività:**
  - Verifica del caricamento di file associati al task.
  - Controllo dello stato del task come "completato".

3.3. Messaggi di gruppo

- **Test Case:** TC-GroupMessage-01
- **Dipendenze:** TC-TeamCreation-01 (team esistente).
- **Durata stimata:** 1 giorno.
- **Risorse necessarie:** 1 QA tester.
- **Attività:**

- Verifica dell'invio di un messaggio di gruppo.
- Test della creazione di nuovi topic.
- Controllo dei messaggi vuoti o con contenuti troppo lunghi.

3.4. Messaggi individuali

- **Test Case:** TC-IndividualMessage-01
- **Dipendenze:** TC-Registration-01 (utenti registrati).
- **Durata stimata:** 1 giorno.
- **Risorse necessarie:** 1 QA tester.
- **Attività:**
  - Verifica dell'invio di messaggi privati.
  - Controllo degli stati "inviato" e "visualizzato".

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.
- **Durata stimata:** 2 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	2 giorni	DevOps, QA
Test funzionali di base	Registrazione, creazione team	3 giorni	1 QA
Test avanzati	Task, completamento, messaggistica	5 giorni	1 QA
Test di sistema	Integrazione e prestazioni	5 giorni	2 QA, 1 DevOps

Totale stimato: 15 giorni lavorativi.