Università degli Studi di Salerno
Corso di Ingegneria del Software: Tecniche Avanzate

Code Smile
Master Test Plan Document
Versione 1.0

Team:
Antonio Caiazzo
Emanuele Iovane
Salvatore Di Martino

Partecipanti:

Nome	**Matricola**
Antonio Caiazzo	NF22500205
Salvatore Di Martino	NF22500114
Emanuele Iovane	NF22500162

Scritto da:	Antonio Caiazzo, Salvatore Di Martino, Emanuele Iovane

Indice

1. Introduzione

Il presente documento definisce la strategia complessiva, l'approccio e le attività pianificate per il processo di testing del progetto *CodeSmile*. Il piano copre i diversi livelli di test previsti, includendo test di unità, di integrazione e di sistema. Per il testing di unità e di integrazione vengono riutilizzati ed estesi i test già esistenti, mantenendo la continuità con le tecniche adottate in precedenza.

2. Strategie di testing

2.1. Testing di unità

L'attività di unit testing ha l'obiettivo di validare in maniera isolata i moduli e le classi principali di *CodeSmile*, garantendo il corretto comportamento delle singole funzionalità. La suite di test è organizzata seguendo la suddivisione in package del sistema: all'interno della directory *test/unit_testing* sono presenti sottocartelle che rispecchiano i moduli del codice, assicurando una corrispondenza diretta tra ciascun test e il relativo componente.
L'approccio adottato è di tipo white-box, con particolare attenzione alla branch coverage. L'obiettivo è verificare che tutti i rami logici del codice vengano esercitati durante l'esecuzione dei test. Per la misurazione della copertura viene utilizzato *coverage.py*, che fornisce report dettagliati sulle linee e sulle diramazioni effettivamente testate.

2.2. Testing di integrazione

L'attività di integration testing mira a validare le interazioni tra i principali componenti architetturali del sistema *CodeSmile*. L'attenzione è rivolta in particolare ai moduli che implementano l'intero flusso di analisi: dall'interfaccia utente (CLI) alla generazione del report finale, passando per l'*Inspector* e il *RuleChecker, ed* all'integrazione del gateway della WebApp con i vari services.
L'obiettivo è assicurare che tali componenti collaborino correttamente e che il flusso end-to-end interno si comporti come previsto.

2.3. Testing di sistema

Il system testing è condotto con un approccio black-box, focalizzandosi sul comportamento osservabile del sistema, senza considerare gli aspetti interni già verificati attraverso i test di unità e integrazione. La progettazione dei test segue il *Category Partition Method*, che ha consentito di individuare in modo sistematico le combinazioni più rilevanti relative ai file di input, ai parametri di configurazione del tool e alle strutture dei progetti analizzati.
La validazione del sistema è stata effettuata tramite una suite di test strutturata secondo tale metodo (vedi documento Pre_Modification_System_Testing.pdf).

2.4. Testing E2E per la WebApp

Per il modulo WebApp è previsto un ciclo di test end-to-end (E2E) volto a verificare il corretto funzionamento dei principali flussi utente, assicurando che tutte le componenti del sistema interagiscano in modo coerente in scenari di utilizzo realistici. L'esecuzione dei test verrà automatizzata tramite strumenti come Cypress, con l'obiettivo di rendere il processo più efficiente e ripetibile.

Gli obiettivi riguardano in particolare la corretta gestione della navigazione tra le pagine, il funzionamento del caricamento dei file Python e dei progetti completi, la capacità dell'applicazione di gestire errori dell'API (come risposte HTTP 500) informando l'utente senza interrompere l'esecuzione, e la corretta generazione del report finale con relativa visualizzazione dei risultati e possibilità di download.
I test E2E sono considerati superati quando l'intero processo di upload e analisi dei file si conclude senza errori inattesi, con tempi di risposta adeguati, quando l'interfaccia utente mantiene sempre disponibili gli elementi necessari alla fruizione del servizio, e quando tutti i flussi di navigazione – dalla homepage alle varie sezioni e fino alla visualizzazione del report – risultano stabili, coerenti e privi di interruzioni.

3. Coverage

L'obiettivo complessivo è raggiungere almeno l'80% di branch coverage, escludendo i moduli relativi alla parte sperimentale di intelligenza artificiale o ai dataset. In dettaglio:

- **Unit Testing:** puntare a una copertura ≥80% per i package core (*components*, *detection_rules*, *extractor*).
- **Integration Testing:** valutare la percentuale di codice esercitata dai test integrati, con un obiettivo superiore al 50%.
- **System ed E2E Testing:** pur non garantendo necessariamente un'elevata copertura in termini di linee di codice, assicurano la validazione completa dei flussi principali del sistema in condizioni operative reali.

4. Criteri di Accettazione

Il piano di test copre le tre richieste di modifica approvate, e presenta i seguenti criteri di accettazione:

- Raggiungimento dell'80% di branch coverage nei package core.
- Assenza di bug critici nel testing di sistema e nel testing E2E.
- Tutti i casi di test pianificati, sono stati eseguiti e approvati, comprensivi sia dei casi di test identificati dopo la modifica che i test selezionati per il regression testing.
- Corretto funzionamento del sistema di generare un'analisi coerente, comprensiva delle nuove funzionalità programmate.