Università degli Studi di Salerno

Corso di Ingegneria del Software

AstroMark
Test Plan
Versione 1.1

Data: 30/01/2025

Partecipanti:

Nome	Matricola
Giuseppe Cavallaro	0512116926
Mario Cosenza	0512116320
Mario Fasolino	0512116965
Giulio Sacrestano	0512116812

Scritto da:	Mario Cosenza
	Mario Fasolino
	Giulio Sacrestano

Revision History

Data	Versione	Descrizione	Autore
15/12/2024	1.0	Stesura inziale TP con definizione dei sistemi da testare e dei casi di test.	Mario Cosenza, Mario Fasolino, Giulio Sacrestano
30/01	1.1	Revisione finale del documento	Mario Cosenza, Mario Fasolino, Giulio Sacrestano

Indice

1. Introduzione

Il documento di **Test Plan** descrive le strategie, le metodologie e gli strumenti utilizzati per la verifica e la validazione del sistema **AstroMark**. L'obiettivo è garantire che il sistema soddisfi i requisiti funzionali e non funzionali, offrendo affidabilità, robustezza e un'esperienza utente ottimale. Il documento fornisce una panoramica delle attività di testing, dei casi di test definiti, e delle risorse necessarie per assicurare la qualità del prodotto finale.

2. Relazione con altri documenti

Il presente progetto si basa sull'analisi e il confronto con piattaforme di gestione della didattica già consolidate e affermate nel settore, le quali hanno dimostrato notevole efficacia. Tra queste, un punto di riferimento significativo è rappresentato dalle soluzioni sviluppate da Argo per la gestione della didattica.

Di seguito si presenta un elenco dei documenti chiave del progetto a cui si fa esplicito riferimento:

- **Problem Statement**: Documento che definisce i problemi principali che il progetto intende affrontare e risolvere.
- **Object Design Document (ODD)**: Documento di Object Design che descrive l'architettura logica e fisica del sistema AstroMark, delineando la struttura delle principali classi, interfacce e componenti che costituiscono l'applicazione.
- **System Design Document (SDD)**: Documento che descrive l'architettura del sistema e le componenti principali del progetto.
- **Requirement Analysis Document (RAD)**: Documento di analisi dei requisiti che dettaglia le esigenze funzionali e non funzionali del sistema.

Oltre ai documenti del progetto, si fa riferimento ad opere di letteratura tecnica che hanno contribuito allo sviluppo metodologico e concettuale di questo lavoro:

- **Object-Oriented Software Engineering Using UML, Patterns, and Java™ Third Edition** di Bernd Bruegge & Allen H. Dutoit.

3. Panoramica sistema

Il sistema AstroMark, progettato come piattaforma open-source per la gestione della didattica nelle scuole secondarie di secondo grado, si basa su un'architettura **three-tier** articolata in interfaccia utente, logica applicativa e gestione dei dati. Questa suddivisione garantisce una chiara distinzione delle responsabilità tra i componenti software, facilitando l'individuazione del livello di granularità su cui condurre i test unitari.

Ogni sottosistema dall'interfaccia utente all'autenticazione, dalla gestione dei profili e dei permessi alla comunicazione in tempo reale, fino alla pianificazione delle lezioni, l'orientamento e la valutazione è concepito secondo principi di coesione e separazione delle responsabilità. Questo approccio rende possibile testare singolarmente ciascuna unità, verificandone sia il corretto funzionamento interno sia l'integrazione con i moduli adiacenti.

La componente di gestione dei dati, pensata per garantire integrità, sicurezza e prestazioni ottimali, rappresenta un ulteriore punto di verifica: i test modulari su questo livello consentiranno di valutare la correttezza delle operazioni di lettura/scrittura, la coerenza dei dati in contesti concorrenti e la resistenza del sistema a carichi elevati.

L'organizzazione architetturale e la chiara definizione dei confini tra i sottosistemi consentono di eseguire test unitari mirati, di identificare con precisione le dipendenze tra i componenti e di assicurare una copertura completa delle funzionalità critiche. In particolare, durante le attività di testing verrà dedicata particolare attenzione ai sottosistemi di Authentication, User Management, Chat e Gestione delle Giustificazioni.

4. Funzionalità da essere testate/non testate

Nella piattaforma AstroMark sarà data priorità al testing dei requisiti funzionali e non funzionali con la massima rilevanza, soprattutto per le funzionalità che prevedono input manuale dell'utente. Ogni interazione con l'utente verrà sottoposta a verifiche approfondite al fine di garantire affidabilità, usabilità e aderenza alle specifiche, assicurando così un'elevata qualità dell'esperienza d'uso.
Di seguito, le funzionalità che saranno testate:

Authentication:

- Autenticazione

User Management:

- Modifica indirizzo
- Modifica password

Agenda:

- Prenotazione ricevimento

Mark

- Inserimento voto

5. Criteri di successo/insuccesso

Il criteri di successo e insuccesso per il piano di test sono definiti in base al soddisfacimento dei requisiti funzionali e non funzionali del sistema. Un test è considerato superato se tutti i risultati effettivi coincidono con i risultati attesi descritti nei casi di test e se non vengono riscontrati errori critici o difetti che possano compromettere le funzionalità principali del sistema. In caso di successo, tutte le funzionalità testate devono rispettare i requisiti specificati, i test delle prestazioni devono raggiungere i livelli accettabili definiti, e non devono verificarsi errori bloccanti o ad alta priorità durante l'esecuzione.

In caso di insuccesso, si considerano come criticità i risultati divergenti da quelli attesi, la presenza di bug bloccanti o critici nel sistema, e il mancato rispetto dei vincoli non funzionali, come tempi di risposta superiori ai limiti consentiti o errori sotto stress elevato.

6. Approccio

L'approccio al testing adottato per questo progetto segue una strategia sistematica e basata su tecniche consolidate per garantire la copertura delle funzionalità richieste. In particolare, utilizziamo il **Category Partition Testing** per identificare e definire le categorie di input significative, i relativi casi di test e le possibili combinazioni, garantendo un'analisi completa e strutturata dei requisiti.

Unit Testing:
Per i test unitari, utilizziamo **JUnit** e **Mockito**, strumenti che consentono di isolare le singole unità di codice e verificarne il comportamento in condizioni controllate. Mockito è utilizzato per simulare dipendenze esterne e testare i metodi in isolamento, minimizzando le interferenze.

Integration Testing:
Il testing di integrazione per **AstroMark** adotta un approccio Sandwich, che combina i vantaggi delle strategie bottom-up e top-down, risultando particolarmente adatto a un sistema orientato agli oggetti. I test sono definiti tramite il framework **JUnit**, mentre **Mockito** viene utilizzato per il mocking delle dipendenze. L'automazione dei test è gestita con **Maven**.
Il processo è diviso in tre step principali:

1. Test delle classi **DAO** (Repository) e la loro interazione con il database.
2. Test delle classi **Service** con i **DAO**, verificando la logica di business. Infine, si testano i **Controller**, mockando i servizi sottostanti con **Mockito** per simulare le risposte.

System Testing:
Per il test di sistema, utilizziamo **Selenium**, uno strumento che consente di automatizzare i test delle interfacce utente e verificare il comportamento complessivo del sistema in scenari realistici. Selenium viene impiegato per simulare l'interazione dell'utente con il sistema, valutando la corretta integrazione delle funzionalità e la conformità ai requisiti funzionali.

L'intero processo di testing è integrato in una pipeline di **Continuous Integration/Continuous Deployment (CI/CD)** automatizzata, implementata tramite **GitHub Actions**. Ad ogni commit o pull request nel repository, GitHub Actions esegue automaticamente l'intera suite di test, inclusi i test unitari, di integrazione e di sistema. Questo approccio consente di rilevare rapidamente eventuali regressioni o errori introdotti da nuove modifiche, garantendo un feedback immediato agli sviluppatori.

7. Sospensione e ripresa

Il processo di testing può essere sospeso o ripreso in base a specifiche condizioni, definite per garantire che le risorse siano utilizzate in modo efficiente e che i test siano eseguiti in un ambiente appropriato.

Criteri di Sospensione

Il testing verrà sospeso nei casi in cui si verifichino difetti critici o bloccanti che impediscano l'esecuzione di ulteriori test, come errori che compromettano l'avvio del sistema o l'accesso a funzionalità chiave. La sospensione avverrà anche in caso di mancata disponibilità dell'ambiente di test, ad esempio per problemi infrastrutturali come errori nei server, mancanza di accesso alle risorse necessarie o malfunzionamenti degli strumenti di automazione. Inoltre, il testing sarà sospeso se si superano i limiti di tempo o di budget previsti per la fase di testing, nel caso sia necessario rivedere la pianificazione o ottenere ulteriori approvazioni.

Criteri di Ripresa

Il testing riprenderà una volta che i difetti critici o bloccanti saranno stati identificati, corretti e validati attraverso un processo di bug fixing seguito da un ciclo di regression testing. Inoltre, la ripresa avverrà quando l'ambiente di test sarà stato completamente ripristinato e reso operativo, garantendo l'accesso a tutte le risorse necessarie. Infine, il testing continuerà dopo che saranno state effettuate le revisioni necessarie alla pianificazione o saranno state allocate risorse aggiuntive per supportare la prosecuzione delle attività.

8. Materiale di test (requisiti hardware/software)

Per garantire la qualità, l'affidabilità e la robustezza della piattaforma **AstroMark** è fondamentale disporre di strumenti di testing adeguati e performanti. Gli strumenti selezionati supportano l'intero ciclo di vita del testing, facilitando l'esecuzione di test unitari, di integrazione e di sistema in modo efficiente e accurato. L'adozione di framework e strumenti specifici consente di automatizzare i processi di verifica, ridurre i tempi di sviluppo e individuare tempestivamente eventuali anomalie o vulnerabilità. Di seguito vengono elencati i requisiti hardware e software necessari per implementare una strategia di testing efficace e completa per **AstroMark**.

Requisiti Hardware:

Postazioni di Sviluppo e Testing:

- **CPU:** Minimo 2 core
- **RAM:** Almeno 8 GB
- **Storage:** SSD con capacità adeguata a PostgreSQL
- Connessione Internet

Requisiti Software:

- **IntelliJ IDEA:** IDE utilizzato per lo sviluppo e il testing sia del front-end che del back-end.
- **Spring Boot:** framework per l'esecuzione dell'applicazione back-end.
- **NPM:** gestore di pacchetti per il linguaggio di programmazione JavaScript e TypeScript.
- **Amazon Corretto 21:** distribuzione di OpenJDK utilizzata per eseguire l'applicazione.
- **PostgreSQL:** DBMS utilizzato per la gestione dei dati.
- **Docker**: piattaforma per la containerizzazione necessaria a Testcontainers per funzionare. Docker fornisce l'ambiente di esecuzione per i container di PostgreSQL.
- **Testcontainers:** libreria Java che semplifica la creazione e la gestione di container Docker per i test di integrazione. Permette di avviare un'istanza di PostgreSQL in un container Docker on-demand durante l'esecuzione dei test.
- **Cloudflare R2:** utilizzato per gli allegati ai messaggi, richiede una **chiave di test** per l'accesso e la gestione degli oggetti durante i test.
- **JUnit:** framework per i test unitari in Java.
- **Mockito:** libreria per la creazione di mock e stubs nei test.
- **Selenium:** strumento per i test di sistema e l'automazione del browser.
- **Postman:** strumento utilizzato per formulare e inviare richieste agli endpoint REST, facilitando il testing delle API.

9. Agenda per il testing

Dopo la conclusione della fase di design, si avvierà la pianificazione dettagliata dei test per garantire una copertura completa e accurata delle funzionalità del sistema.
In ottica di Continuous Integration (CI), i casi di test saranno sviluppati in parallelo con lo sviluppo del codice, assicurando che ogni nuova commit sia accompagnata da test specifici che ne verifichino il corretto funzionamento fin dalle prime fasi di implementazione. Questo approccio permette di identificare e risolvere tempestivamente eventuali problemi di integrazione.
Grazie all'implementazione della CI, una volta completata l'implementazione di una funzionalità, questa viene integrata nel repository principale e automaticamente sottoposta a una serie di test automatizzati, che includono test unitari e di integrazione, mirati a verificare l'integrità e la conformità del prodotto rispetto ai requisiti definiti. La CI prevede l'esecuzione automatica di tutti i test a ogni integrazione, effettuando un regression testing continuo. Questo assicura che le nuove modifiche non abbiano introdotto nuovi errori o regressioni.

10. Glossario

Termine	Definizione
Category Partition Testing	Tecnica utilizzata per identificare e definire le categorie di input significative, i relativi casi di test e le possibili combinazioni, garantendo un'analisi completa e strutturata dei requisiti.
Unit Testing	Test che isolano le singole unità di codice per verificarne il comportamento in condizioni controllate, spesso utilizzando strumenti come JUnit e Mockito.
JUnit	Framework utilizzato per i test unitari in Java, consente di scrivere e eseguire test automatici per verificare il corretto funzionamento del codice.
Mockito	Libreria per la creazione di mock e stubs nei test, utilizzata per simulare dipendenze esterne e testare i metodi in isolamento.
Integration Testing	Test che verificano l'interazione tra diverse componenti del sistema, adottando approcci come il Sandwich per combinare strategie bottom-up e top-down.
Sandwich approach	Approccio per il testing di integrazione che combina i vantaggi delle strategie bottom-up e top-down, risultando particolarmente adatto a sistemi orientati agli oggetti.
Maven	Strumento di automazione del build e gestione delle dipendenze, utilizzato per gestire l'automazione dei test nel progetto AstroMark.
System Testing	Test che verificano il comportamento complessivo del sistema in scenari realistici, utilizzando strumenti come Selenium per automatizzare i test delle interfacce utente.
Selenium	Strumento per l'automazione dei test delle interfacce utente e la verifica del comportamento complessivo del sistema in scenari realistici.
Continuous Integration/Continuous Deployment (CI/CD)	Pipeline automatizzata che integra e distribuisce continuamente il codice, eseguendo automaticamente l'intera suite di test ad ogni commit o pull request.
GitHub Actions	Servizio di automazione CI/CD utilizzato per eseguire automaticamente la suite di test ad ogni commit o pull request nel repository.

Criteri di Sospensione	Condizioni specifiche che determinano quando il processo di testing deve essere sospeso, come difetti critici o mancata disponibilità dell'ambiente di test.
Criteri di Ripresa	Condizioni che determinano quando il processo di testing può essere ripreso, come la correzione dei difetti critici o il ripristino dell'ambiente di test.
Postman	Strumento utilizzato per formulare e inviare richieste agli endpoint REST, facilitando il testing delle API.
IntelliJ IDEA	IDE utilizzato per lo sviluppo e il testing sia del front-end che del back-end nel progetto AstroMark.
Spring Boot	Framework Java per lo sviluppo di applicazioni Spring con configurazioni automatiche e componenti pronti all'uso.
NPM	Gestore di pacchetti per il linguaggio di programmazione JavaScript e TypeScript, utilizzato per gestire le dipendenze del progetto.
Amazon Corretto 21	Distribuzione di OpenJDK utilizzata per eseguire l'applicazione AstroMark.
PostgreSQL	DBMS utilizzato per la gestione dei dati nel progetto AstroMark.
Cloudflare R2	Servizio utilizzato per la gestione degli allegati ai messaggi, richiede una chiave di test per l'accesso e la gestione degli oggetti durante i test.
Java Mail	Libreria utilizzata per inviare email, necessita di un server SMTP di test per verificare l'invio delle e-mail senza inviarle reali.