Università degli Studi di Salerno
Corso di Ingegneria del Software

Test Plan

Car – Zone

Versione 0.2

13/02/2025

Coordinatore del progetto:

Nome	**Matricola**
Francesco Pio Cataudo	0512116773

Partecipanti:

Nome	**Matricola**
Francesco Pio Cataudo	0512116773
Francesco Santoro	0512117079
Francesco Pio Bottaro	0512118180
Errico Aquino	0512117730

Revision History

Data	Versione	Descrizione	Autori
13/12/2024	0.1	Strutturazione documento ed informazioni iniziali.	Team
13/02/2025	0.2	Revisione	Team

Sommario

1.Introduzione

Questo documento definisce il piano di test per il sistema, con l'obiettivo di garantire la qualità e la conformità ai requisiti definiti. Il piano di test stabilisce il framework per la pianificazione, l'esecuzione e la valutazione dei test in modo efficiente e sistematico.

L'obiettivo principale è validare le funzionalità implementate e verificare il corretto funzionamento delle interazioni tra i diversi componenti del sistema, riducendo il rischio di malfunzionamenti e garantendo il rispetto degli standard di qualità richiesti.

Gli aspetti chiave che verranno testati includono:

- **Funzionalità del sistema**: Test per verificare il comportamento delle principali funzionalità come la gestione degli utenti, la gestione degli ordini, l'elaborazione dei pagamenti e la gestione del carrello.
- **Integrazione tra i componenti**: Assicurarsi che i moduli del sistema comunichino correttamente tra loro, in particolare tra i pacchetti dao, entity, servlet e utility.
- **Performance e scalabilità**: Valutare i tempi di risposta del sistema sotto carico e la capacità di gestire un numero elevato di richieste concorrenti.
- **Sicurezza**: Testare la protezione dei dati degli utenti, inclusa la gestione delle sessioni e la crittografia delle password.
- **Usabilità**: Verificare che il sistema sia intuitivo e facile da usare per gli utenti finali.

Questo documento sarà utilizzato dai responsabili del progetto e dai tester per garantire un approccio strutturato e metodico al processo di testing, con un'attenzione particolare alla tempestività e all'efficacia dei test.

2. Relazione con altri documenti

Il piano di test è strettamente correlato ad altri documenti prodotti durante lo sviluppo del sistema, in particolare:

- **Requisiti di sistema** (RAD - Requirement Analysis Document): Definisce i requisiti funzionali e non funzionali che il software deve soddisfare. I test verranno tracciati rispetto a questi requisiti per garantire che il sistema risponda alle specifiche stabilite.

- **Progettazione del sistema** (SDD - Software Design Document): Specifica l'architettura del software, descrivendo i componenti, le interfacce e le interazioni tra le diverse parti del sistema. I test di integrazione e unitari saranno pianificati in base a questa struttura.

- **Documento di Progettazione degli Oggetti** (ODD - Object Design Document): Definisce la struttura orientata agli oggetti del sistema, dettagliando le classi, i metodi pubblici e le dipendenze tra i moduli. Questo documento sarà utilizzato per la definizione dei test unitari e di integrazione.

I test saranno definiti in modo da verificare la conformità del sistema ai requisiti e alla progettazione stabilita in questi documenti. Per garantire la tracciabilità tra i requisiti e i test, verrà adottata una nomenclatura che colleghi ogni test ai relativi requisiti e componenti progettuali. Questo approccio assicurerà una copertura completa e sistematica del processo di testing.

3. Overview del Sistema

Questa sezione fornisce una panoramica strutturale del sistema, definendo i componenti principali che saranno sottoposti a test durante il processo di verifica e validazione.

Il sistema è suddiviso nei seguenti moduli principali:

- **Modulo di autenticazione e gestione utenti**: Responsabile della gestione degli accessi, dell'autenticazione e della sicurezza delle sessioni.

- **Modulo di gestione dei prodotti**: Gestisce la creazione, l'aggiornamento e la rimozione dei prodotti disponibili nel sistema.

- **Modulo di gestione degli ordini**: Si occupa della creazione, modifica e monitoraggio degli ordini effettuati dagli utenti.

- **Modulo di gestione del carrello**: Consente agli utenti di aggiungere e rimuovere prodotti dal carrello e finalizzare l'acquisto.

- **Modulo di amministrazione**: Permette agli amministratori di gestire le impostazioni del sistema, gli utenti e le statistiche di utilizzo.

Relazioni e dipendenze

Ogni modulo dipende da specifici componenti del sistema e interagisce con gli altri per garantire il corretto funzionamento dell'intera applicazione:

- Il **modulo di autenticazione** è utilizzato trasversalmente da tutti gli altri moduli per gestire gli accessi e le autorizzazioni.
- Il **modulo di gestione dei prodotti** è collegato al modulo di gestione del carrello e degli ordini, assicurandosi che solo i prodotti disponibili possano essere acquistati.
- Il **modulo di gestione degli ordini** dipende sia dal modulo del carrello che dal modulo di pagamento per convalidare e completare le transazioni.
- Il **modulo di amministrazione** offre funzionalità di supervisione e controllo sull'intero sistema.

Componenti da testare nei test unitari

Nei test unitari verranno verificati i seguenti componenti:

- **Classi DAO**: Verifica dell'integrità dei metodi di accesso ai dati e delle query eseguite.
- **Classi Entity**: Validazione delle strutture dati e dei relativi metodi getter e setter.
- **Servlet**: Test sulle operazioni HTTP, inclusa la gestione delle richieste e delle risposte.
- **Classi Utility**: Controllo della correttezza delle funzioni di supporto, come la formattazione delle date e la validazione degli input.

Questa suddivisione fornirà un quadro chiaro e dettagliato del sistema, permettendo di individuare eventuali malfunzionamenti e garantendo una copertura completa dei test.

4. Feature da testare / non testare

Funzionalità da testare

Le seguenti funzionalità verranno testate per garantire il corretto funzionamento del sistema:

- **Autenticazione e gestione utenti** (checkadmin, checkcustomer): Test sulla registrazione, login, logout e gestione delle sessioni.
- **Gestione dei prodotti** (addproduct, deleteProduct, updateproductname, updateproductprice, updateproductquantity, updateproductcategoryid, updateproductbrandid, updateproductdescription, updateproductimage): Creazione, aggiornamento e rimozione di prodotti.
- **Gestione del carrello** (addtocart, removecart, removetable_cart): Aggiunta, rimozione e aggiornamento di articoli nel carrello.

- **Gestione degli ordini** (removeorders, removetable_order_details, payprocess, ShippingAddress2): Creazione, monitoraggio e completamento degli ordini.
- **Gestione dei pagamenti** (payprocess): Validazione dei metodi di pagamento e gestione delle transazioni.
- **Gestione dei contatti** (addContactus, remove_contactus): Test della gestione delle richieste di contatto.
- **Ruoli e autorizzazioni**: Test per verificare la corretta gestione degli accessi in base ai ruoli utente.
- **Integrazione tra moduli**: Test per garantire il corretto scambio di dati tra i vari moduli del sistema.
- **Performance e sicurezza**: Test di carico, stress test e verifica della protezione dei dati utente.

Funzionalità non testate

- **Gateway di pagamento esterni**: Moduli come PayPal o Stripe.
- **Compatibilità con browser obsoleti**: Sarà testato solo sui browser recenti.
- **Interfaccia grafica**: I test si concentreranno sulla logica di business e sulle API.

Questa selezione garantisce un testing efficace, concentrando le risorse sulle aree critiche del sistema.

5. Pass / Fail criteria

Questa sezione definisce i criteri per determinare il successo o il fallimento dei test condotti sul sistema. I criteri di valutazione sono definiti sulla base dei requisiti funzionali e non funzionali e sono strutturati come segue:

5.1 Criteri di successo

Un test sarà considerato **superato** se soddisfa i seguenti criteri:

- **Funzionalità**: Ogni funzione testata restituisce l'output atteso e si comporta come specificato nei documenti di progettazione.
- **Affidabilità**: Il sistema esegue i test senza crash, errori fatali o malfunzionamenti che impediscano il normale utilizzo.

- **Performance**: I tempi di risposta per le operazioni chiave rientrano nei limiti specificati (es. autenticazione in meno di 2 secondi, caricamento pagine sotto 3 secondi).
- **Sicurezza**: Il sistema non presenta vulnerabilità critiche nei test di autenticazione, autorizzazione e gestione delle sessioni.
- **Integrazione**: Le comunicazioni tra i vari moduli avvengono correttamente senza perdita di dati o inconsistenze.

5.2 Criteri di fallimento

Un test sarà considerato **fallito** se si verificano uno o più dei seguenti eventi:

- **Errore funzionale**: L'output del test non corrisponde alle specifiche.
- **Crash del sistema**: Il sistema si arresta inaspettatamente o genera errori bloccanti.
- **Problemi di performance**: I tempi di risposta sono superiori ai limiti definiti.
- **Bug critici**: Sono rilevati problemi di sicurezza che permettono accessi non autorizzati o perdita di dati.
- **Fallimento nei test di integrazione**: I moduli non riescono a comunicare tra loro o si verificano incongruenze nei dati.

5.3 Strategie per la gestione degli errori

Nel caso di un test fallito, saranno seguite le seguenti azioni:

1. **Segnalazione immediata**: Il test fallito verrà registrato in un report di errore con dettagli sul contesto e la riproducibilità del problema.
2. **Assegnazione delle priorità**: Gli errori verranno classificati come **critici, maggiori o minori** in base all'impatto sul sistema.
3. **Risoluzione e ritest**: Dopo la correzione del bug, il test verrà rieseguito per verificare la risoluzione del problema.
4. **Analisi di regressione**: Se necessario, verranno effettuati test aggiuntivi per garantire che le correzioni non abbiano introdotto nuovi errori in altre parti del sistema.

Questi criteri assicurano che i test siano efficaci nell'identificare eventuali problemi e che ogni test fallito venga gestito in modo tempestivo e strutturato.

6. Approccio

Questa sezione descrive l'approccio generale adottato per il testing del sistema, basandosi sulle informazioni del codice sorgente e sulle dipendenze tra i vari componenti.

6.1 Strategia di testing

L'approccio seguito è una combinazione di **test unitari**, **test di integrazione** e **test di sistema**, con particolare attenzione alle interazioni tra i moduli del software. Le strategie utilizzate includono:

- **Test unitari**: Saranno condotti su tutte le classi dei pacchetti dao, entity, servlet e utility per verificare il corretto funzionamento dei metodi pubblici e delle dipendenze interne.
- **Test di integrazione**: Si concentreranno sulla verifica delle interazioni tra DAO e il database, tra le servlet e i controller, e tra i servizi del sistema.
- **Test funzionali**: Verifica dell'esecuzione corretta delle operazioni fondamentali del sistema, come la gestione degli utenti, degli ordini e del carrello.
- **Test di performance**: Simulazioni di carico e stress per misurare la reattività del sistema.
- **Test di sicurezza**: Controllo della protezione dei dati, gestione delle sessioni e protezione contro accessi non autorizzati.

6.2 Strategia di integrazione

L'integrazione dei componenti sarà testata progressivamente seguendo un approccio **bottom-up**, testando prima i moduli più basilari (DAO e utility), poi le entità e infine le servlet.

Il flusso generale dell'integrazione sarà:

1. **Test sui DAO**: Verifica dell'accesso ai dati, correttezza delle query SQL e gestione delle connessioni al database.
2. **Test sulle entità**: Validazione della struttura dei dati e delle operazioni su oggetti.
3. **Test sulle servlet**: Verifica delle richieste HTTP e della loro gestione.
4. **Test di integrazione finale**: Controllo del flusso generale tra moduli e componenti.

6.3 Utilizzo di UML per l'integrazione

Per illustrare le dipendenze tra i test e i componenti, verrà utilizzato un **diagramma UML delle classi e delle dipendenze**, che evidenzierà il modo in cui i vari moduli interagiscono tra loro.

6.4 Automatizzazione dei test

Dove possibile, i test unitari e di integrazione saranno automatizzati utilizzando **JUnit** per i test Java e **Mockito** per la simulazione delle dipendenze.

I test di performance saranno eseguiti utilizzando **JMeter**, mentre i test di sicurezza includeranno analisi con strumenti come **OWASP ZAP** per individuare vulnerabilità.

6.5 Strategie per il debugging e la risoluzione dei problemi

In caso di errori o fallimenti nei test:

1. **Logging dettagliato**: Tutti i test saranno accompagnati da log dettagliati per facilitare la risoluzione dei problemi.
2. **Analisi delle cause**: Ogni bug identificato sarà analizzato per determinare la causa principale e classificato in base alla sua criticità.
3. **Retest**: Dopo la correzione, il test sarà rieseguito per confermare che il problema sia stato risolto.
4. **Test di regressione**: Per garantire che le modifiche non abbiano introdotto nuovi problemi.

Questo approccio garantirà un testing efficace e strutturato del sistema.

7. Sospensione e ripresa

Questa sezione definisce i criteri per la sospensione e la ripresa dei test all'interno del ciclo di verifica del sistema. La sospensione dei test può avvenire a causa di problemi tecnici, anomalie critiche o indisponibilità delle risorse necessarie. La ripresa del testing avviene dopo la risoluzione delle problematiche rilevate, assicurando la continuità del processo di validazione.

7.1 Criteri per la sospensione dei test

I test verranno sospesi in presenza di una o più delle seguenti condizioni:

- **Blocco critico del sistema**: Se viene riscontrato un bug che impedisce il funzionamento di funzionalità chiave, i test saranno interrotti fino alla sua risoluzione.
- **Indisponibilità dell'ambiente di test**: Se i server, database o altre risorse necessarie per l'esecuzione dei test non sono disponibili o presentano malfunzionamenti.
- **Errori nei dati di test**: Se i dati utilizzati per i test risultano incoerenti o corrompono l'integrità del database.

- **Problemi di integrazione tra moduli**: Se viene riscontrata un'incompatibilità grave tra i diversi componenti del sistema che impedisce il proseguimento del test.
- **Fallimento di test critici ripetuti**: Se un test critico fallisce ripetutamente senza una causa immediatamente individuabile, sarà necessaria una revisione approfondita del codice prima della ripresa dei test.

7.2 Criteri per la ripresa dei test

I test riprenderanno solo dopo aver verificato la risoluzione delle problematiche sopra elencate. In particolare:

- **Correzione dei bug bloccanti**: Ogni errore critico deve essere identificato, risolto e confermato tramite test di regressione.
- **Ripristino dell'ambiente di test**: Le risorse necessarie per l'esecuzione dei test (server, database, strumenti di test) devono essere funzionanti e accessibili.
- **Aggiornamento dei dati di test**: Se necessario, i dati di test devono essere rivisti e ripristinati a uno stato coerente.
- **Esecuzione dei test di verifica**: Prima di riprendere il testing completo, devono essere eseguiti test specifici per verificare che il problema originario sia stato risolto.

7.3 Attività da ripetere alla ripresa dei test

Dopo la ripresa dei test, le seguenti attività dovranno essere rieseguite:

1. **Test di verifica sulle correzioni effettuate**: Controllare che il problema risolto non si ripresenti.
2. **Test di regressione**: Assicurarsi che le modifiche apportate non abbiano compromesso altre parti del sistema.
3. **Ri-esecuzione dei test precedentemente sospesi**: Riprendere l'esecuzione dei test interrotti per completare la validazione del sistema.
4. **Aggiornamento della documentazione dei test**: Registrare le modifiche apportate e le azioni intraprese per la ripresa del testing.

Questo approccio garantisce che i test siano condotti in modo efficace e che le interruzioni non compromettano l'integrità dell'intero processo di validazione del sistema.

8. Testing materias (hardware/software requirements

Le seguenti applicazioni e strumenti sono richiesti per l'esecuzione dei test:

- **Ambiente di sviluppo e debugging**:
    - IDE: IntelliJ IDEA / Eclipse / Visual Studio Code
    - JDK 11+
    - Servlet container: Apache Tomcat 9+
- **Strumenti di testing**:
    - **JUnit**: per test unitari
    - **Mockito**: per simulazione di dipendenze
    - **Postman**: per test delle API REST
    - **JMeter**: per test di carico e performance
    - **OWASP ZAP**: per test di sicurezza
- **Database per test**:
    - MySQL / PostgreSQL con dati di test preconfigurati
    - Strumenti di gestione database: MySQL Workbench / pgAdmin

8.3 Strumenti di automazione e CI/CD

Per garantire un'integrazione continua e un esecuzione automatizzata dei test, saranno utilizzati:

- **Jenkins**: per l'automatizzazione del testing e deployment
- **GitHub Actions**: per l'esecuzione automatizzata dei test su commit e merge
- **Docker**: per la creazione di ambienti di test isolati

8.4 Ambiente di test

I test saranno eseguiti nei seguenti ambienti:

- **Ambiente locale**: per esecuzione di test unitari e di integrazione in sviluppo
- **Ambiente di staging**: simile alla produzione, per test di sistema e accettazione
- **Ambiente di produzione simulata**: utilizzato per test di regressione prima del rilascio

Questa configurazione garantisce un'infrastruttura di testing adeguata, riducendo i rischi di errori nel software e facilitando la rilevazione e la correzione di bug in fase di sviluppo.

9. Testing cases

Questa sezione descrive i test case che verranno utilizzati per verificare il corretto funzionamento del sistema. I test case sono definiti sulla base delle funzionalità implementate nel progetto e coprono test unitari e funzionali.

9.1 Categorie di Test

I test case sono suddivisi nelle seguenti categorie:

- **Test unitari**: verificano il corretto funzionamento delle singole classi e metodi nei pacchetti dao, entity, servlet e utility.
- **Test funzionali**: validano il comportamento delle principali funzionalità del sistema dal punto di vista dell'utente.

9.2 Elenco dei Test Case

ID Test	Nome Test	Descrizione	Input	Output Atteso	Categoria
TCS-DB-001	Test_DBConnection_001	Verificare la connessione al database	Nessun input	Connessione stabilita con successo	Database
TCS-PR-002	Test_AddProduct_002	Validare l'aggiunta di un prodotto	Dati prodotto	Prodotto aggiunto correttamente	Prodotti
TCS-AU-003	Test_UserAuthentication_003	Verificare login utenti	Credenziali utente	Accesso consentito o negato	Autenticazione

TCS-CA-004	Test_CartFunctionality_004	Verificare operazioni carrello	ID prodotto	Aggiunta o rimozione riuscita	Carrello
TCS-OR-005	Test_OrderProcessing_005	Validare gestione ordini	ID ordine	Ordine processato correttamente	Ordini
TCS-SC-006	Test_Security_006	Verificare protezione SQL Injection	Input malevolo	Query bloccata	Sicurezza

9.3 Documentazione e Report dei Test

- Ogni test eseguito sarà documentato con i risultati dettagliati e le eventuali anomalie riscontrate.
- In caso di errori, verranno prodotti **Test Incident Reports** per analizzare le cause e definire le azioni correttive.
- I test saranno rieseguiti dopo ogni correzione per garantire che il problema sia stato risolto e che non siano stati introdotti nuovi errori.

Questa struttura garantisce una copertura essenziale dei test e assicura che il sistema funzioni correttamente nelle operazioni di base.

10. Testing schedule

Questa sezione descrive il piano temporale per l'esecuzione dei test, le responsabilità assegnate, i rischi e le eventuali azioni correttive.

10.1 Fasi del Testing

Fase	Attività	Durata Stimata	Responsabili
Preparazione	Configurazione ambienti di test, creazione dati di test	2 giorni	QA Team

Test unitari	Esecuzione test su classi DAO, Entity, Utility	3 giorni	Sviluppatori
Test funzionali	Verifica delle principali funzionalità del sito	3 giorni	QA Team
Test di integrazione	Verifica delle interazioni tra moduli DAO, Servlet e database	2 giorni	QA Team
Test di sicurezza	Controllo SQL Injection, XSS, protezione dati	2 giorni	QA Team
Test di performance	Test di carico su ordini, carrello e gestione utenti	2 giorni	QA Team
Test di regressione	Rieffettuazione test critici dopo correzione bug	2 giorni	QA Team
Validazione finale	Verifica generale e approvazione prima del rilascio	1 giorno	QA Team, PM

10.2 Rischi e Contromisure

Rischio	Impatto	Contromisura
Ambienti di test non disponibili	Alto	Creare ambienti ridondanti
Bug critici bloccanti	Alto	Debugging immediato, iterazione test
Ritardi nelle correzioni	Medio	Definizione di SLA per fix bug
Risorse insufficienti per testing	Medio	Pianificazione preventiva del personale

10.3 Responsabilità

- **Sviluppatori**: Eseguono test unitari e correggono bug.
- **QA Team**: Esegue test funzionali, di integrazione e sicurezza.
- **Project Manager (PM)**: Supervisiona il processo e approva il rilascio finale.

10.4 Strumenti Utilizzati

- **JUnit** per test unitari
- **Selenium** per test funzionali automatizzati
- **Postman** per test API
- **JMeter** per test di carico e performance
- **OWASP ZAP** per test di sicurezza

Questa pianificazione garantisce un testing completo ed efficace prima della messa in produzione del sistema.