TP – Test Plan

Versione	1.0
Data	29/11/2021
Destinatario	Prof.ssa F. Ferrucci, Prof. Fabio Palomba
Presentato da	Tutti i team member
Approvato da	Alice Vidoni, Salvatore Amideo

Revision History

Data	Versione	Descrizione	Autori
29/11/2021	0.1	Prima stesura	[Tutti]
29/11/2021	1.0	Revisione PM	Alice Vidoni, Salvatore Amideo

Sommario

1. Introduzione

Nel documento in oggetto verranno definiti gli approcci e le attività di testing in merito alla piattaforma web UnisaEat. Verranno identificate le funzionalità della piattaforma da testare o meno, approcci e strumenti da utilizzare per l'attività di testing. L'obiettivo principale del testing è quello di identificare e prevedere quanti più difetti possibili di un'applicazione tramite i suoi malfunzionamenti per evitare che essi possano verificarsi durante il normale funzionamento. Per cui, diremo che le attività di testing avranno avuto successo se riusciremo a identificare quanti più possibili fault e failure presenti nel sistema prima che il software venga messo in esercizio.

Consideremo solo le gestioni aventi requisiti con priorità alta o media e quindi:

- TesserinoDigitaleManagement
- ProfiloManagment
- OrdinePastoManaegment
- MenuManagement
- ChatManagement
- PersonaleADISUManagement
- OperatoreMensaManagement
- FAQManagement
- StatisticheSettimanaliManagement
- TicketPiattaformaManagement
- NotificheManagement
- QRManagement
- AutenticazioneManagement

1.1 Definizioni, Acronimi e Abbreviazioni

1.1.1 Definizioni

- **Branch Coverage:** metodo che richiede che tutti i rami del programma o gli stati condizionali vengo

testati almeno una volta durante un processo di test;

- **Failure:** mancata prestazione di un servizio atteso;

- **Fault:** causa di un failure, insieme di informazioni che nel momento in cui sono processate generano un fallimento;
- **Model View Control:** modello architetturale utilizzato in programmazione per dividere il codicein blocchi dalle funzionalità ben distinte;

1.1.2 Acronimi e Abbreviazioni

- UR_TP_Vers.1.1: Utilizzata per indicare il Test Plan;
- RAD: Abbreviazione utilizzata per indicare il Requirement Analysis Document;
- SDD: Abbreviazione utilizzata per indicare il System Design Document;
- ODD: Abbreviazione utilizzata per indicare il Object Design Document;
- SOW: Abbreviazione utilizzata per indicare lo Statement Of Work;
- BC: Abbreviazione utilizzata per indicare il Business Case;
- TP: Abbreviazione utilizzata per indicare il Test Plan;
- STC: Abbreviazione utilizzata per indicare il System Test Case;
- MVC: Abbreviazione utilizzata per indicare l'architettura Model View Controller;
- DB: Abbreviazione utilizzata per indicare il Database;
- UE_CP_Vers.1.1: Abbreviazione utilizzata per indicare il documento riguardante il Category Partition;

1.2 Riferimenti

- Kathy Schwalbe, "Information Technology Project Management", International Edition 7E, Cengage Learning, 2014;
- Bernd Bruegge, Allen H. Dutoit, "Object-Oriented Software Engineering Using UML, Patterns and Java", Third Ed., Pearson, 2010;
- Sommerville, "Software Engineering", Addison Wesley;
- PMBOK ® Guide and Software Extention to the PMBOK® Guide, Fifth Ed., Project Management Institute, 2013
- Documentazione di Progetto:
  - UE_RAD_Vers.2.1;
  - UE_SDD_Vers.1.0;

- UE_STC_Vers.1.0;
- UE_SOW_Vers.1.1;

2. Documenti Correlati

Il presente documento è in stretta relazione con i documenti prodotti fino al rilascio della versione 1.0 del documento in oggetto, inoltre, lo sarà ugualmente anche con documenti che verranno sviluppati e rilasciati successivamente, per cui, sarà soggetto a modifiche ed aggiornamenti. I test case saranno basati sulle funzionalità individuate nel documento di raccolta ed analisi dei requisiti.

2.1 Relazione con il documento di raccolta ed analisi dei requisiti (RAD)

La relazione fra TP e RAD riguarda i requisiti funzionali e non funzionali del sistema, infatti il RAD contiene la descrizione dettagliata delle funzionalità nella quale è indicata anche la priorità.

2.2 Relazione con il System Design Document (SDD)

Nell' SDD è presente la architettura del sistema (MVC), la struttura dei dati e i servizi dei sottosistemi.

2.3 Relazione con l'Object Design Document (ODD)

Nell'ODD (ancora non sviluppato al momento del rilascio dell'UE_TP_Vers.1.0) sono contenuti i package e le classi del sistema.

2.4 Relazione con lo Statement Of Work (SOW)

Nello Statement Of Work uno dei criteri di premialità, è quello di ottenere una branch coverage del 75%, i test case verranno strutturati in modo tale da poter soddisfare tale criterio.

3. Panoramica del sistema

Il sistema proposto consiste in una piattaforma Web che si rivolge ai clienti della mensa di Unisa, operatori mensa e personale dell'Adisu. Gli utenti che avranno accesso alla piattaforma saranno: cliente, operatore mensa, personale Adisu, admin. Tutti gli utenti potranno effettuare il login ed il logout. Il personale Adisu viene inserito manualmente dall'admin, mentre gli operatori mensa vengono inseriti dal personale Adisu. Una volta effettuato il login la piattaforma metterà a disposizione diverse funzionalità a seconda del tipo di utente ad aver effettuato l'accesso. Tutti gli utenti possono visualizzare la propria area personale. Tutti gli utenti tranne l'admin possono compilare un ticket, risolvibile dall'admin. Il cliente potrà richiedere,

rinnovare, ricaricare un tesserino digitale e visualizzarne il saldo. Potrà prenotare e pagare pasti utilizzando il tesserino. Per qualsiasi problema, potrà comunicare via chat con un membro del personale Adisu. Gli operatori mensa si occuperanno dell'inserimento del menù giornaliero visualizzabile dai clienti e potranno visualizzare le statistiche settimanali riguardanti i pasti. Il personale Adisu gestirà le FAQ, visualizzabili dal cliente. L'architettura utilizzata è di tipo repository, in quanto adatta quando il sistema è basato sull'archivio dei dati e la loro modifica è frequente e coinvolge più parti. In questo tipo di architettura i sottosistemi che compongono il software accedono e modificano una singola struttura dati, chiamata repository; questo fa sì che i vari sottosistemi siano fra di loro relativamente indipendenti, in quanto interagiscono solo mediante il repository. Come pattern è utilizzato l'MVC (Model-View-Controller), un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito object-oriented, in grado di separare la logica di presentazione dei dati, dalla logica di business.

4. Possibili Rischi

ID	Rischio	Risvolto	Probabilità	Impatto
R1	Pianificazione della fase ditesting non adeguata	Negativo	Bassa	Alto
R2	Il team non segue l'approccio definito	Negativo	Media	Alto
R3	Il team trova difficoltà nello specificare i casi di test	Negativo	Media	Alto
R4	Il team trova difficoltà nell'uso dei tool scelti per svolgere l'attività di testing	Variabile	Media	Medio-Alto
R5	Le attività di testing proseguono oltre i tempiprestabiliti	Variabile	Media	Medio-Alto
R6	I casi di test definiti non riescono ad ottenere unabranch coverage del 75%	Negativo	Media	Medio

5. Funzionalità da testare

Come indicato in precedenza, andranno testati solo i requisiti che presentano una priorità media o alta. Di seguito l'elenco dei requisiti da testare raggruppati per ogni gestione:

- TesserinoDigitaleManagement
  - RichiestaTesserino
  - RinnovoTesserino
  - RicaricaTesserino
  - VisualizzazioneSaldo
- ProfiloManagement
  - VisualizzazioneAreaPersonale
  - ModificaPassword
- OrdinePastoManagement
  - SceltaPasti
  - PagamentoPasto
  - VisualizzaRiepilogoOrdine
- MenuManagement
  - VisualizzazioneMenu
  - InserimentoMenu
  - ModificaMenu
- ChatManagement
  - CreazioneChat
  - InvioMessaggio
  - VisualizzaChat
  - RimozioneMessaggio
  - ModificaMessaggio
- PersonaleAdisuManagement
  - InserimentoPersonaleAdisu
  - RimozionePersonaleAdisu
  - VisualizzazioneListaPersonaleAdisu

- OperatoreMensaManagement
    - InserimentoOperatoreMensa
    - RimozioneOperatoreMensa
    - VisualizzazioneListaOperatoriMensa
- FAQManagement
    - VisualizzazioneFAQ
    - InserimentoDomandaInFAQ
    - ModificaDomandaInFAQ
    - RimozioneDomandaInFAQ
- StatisticheSettimanaliManagement
    - VisualizzazioneStatisticheSettimanali
- TicketManagement
    - CompilazioneTicket
    - VisualizzazioneListaDiTicket
    - RisoluzioneTicket
- NotificheManagement
    - VisualizzazioneElencoNotifiche
    - RimozioneNotifica
- QRManagement
    - VisualizzazioneQR
- AutenticazioneManagement
    - Login
    - Logout

6. Funzionalità da non testare

Non sono presenti nel sistema requisiti che presentano una priorità bassa. Per questo motivo non saranno presenti funzionalità da non testare.

7. Approccio

7.1 Testing di unità

In questa fase andremo a testare ogni singola funzione degli oggetti creati. Questa rappresenterà la nostra unità. Verrà utilizzato un approccio black-box, ovvero non sarà basato sulla conoscenza dell'architettura e del funzionamento interno di un componente ma sulle sue funzionalità esternamente esposte. Per tale fase utilizzeremo i tool Mocha, framework di test per NodeJS, al quale abbineremo Chai, una libreria per redigere asserzioni.

7.2 Testing di integrazione

In questa fase le singole unità vengono combinate e testate come gruppo. Per poter effettuare l'integration test è stata scelta la strategia bottom-up, in quanto consente di poter iniziare l'attività di testing non appena il primo modulo è stato specificato. Questo approccio richiede la costruzione di driver per simulare l'ambiente chiamante. In generale però, può portare alla problematica che i moduli possano essere codificati senza avere una chiara idea di come dovranno essere connessi ad altre parti del sistema. La riusabilità del codice è uno dei principali benefici dell'approccio bottom-up. Nonostante questa strategia di testing di integrazione abbia alcune limitazioni, risulta essere la più semplice e naturale forma con cui eseguire questo tipo di testing.

7.3 Testing di sistema

Prima della messa in esercizio del sistema verrà effettuata una fase di testing di sistema per dimostrare che i requisiti commissionati dal cliente sono soddisfatti. In questo tipo di testing andremo a testare le funzionalità usate più frequentemente dal cliente e quelle che presentano maggiore possibilità di fallimento. Trattandosi di una applicazione web verrà utilizzato il tool Selenium, che si occuperà di simulare l'interazione con il sistema dal punto di vista dell'utente.

8. Criteri di Pass/Fail

Una volta individuati i vari dati di input del test, questi verranno raggruppati in base a caratteristichecomuni in insiemi. In questo modo sarà possibile diminuire ed ottimizzare il numero di test. La fase ditest avrà successo se individuerà una failure, cioè se l'output osservato sarà diverso da quello atteso. Ogniqual volta verrà individuata

una failure, questa verrà analizzata e, se legata ad un fault, si procederà alla sua correzione. Una volta completata la correzione, si procederà, in modo iterativo, ad una nuova fase di test per verificare che la modifica non ha impattato su altri componenti del sistema. Di contro, il testing fallirà se l'output osservato sarà uguale all'oracolo.

9. Criteri di Sospensione e Ripresa

9.1 Criteri di sospensione

La fase di testing verrà interrotta quando verranno raggiunti i risultati attesi in accordo con in tempie i costi allocati alle attività di testing.

9.2 Criteri di ripresa

Le attività di testing riprenderanno in seguito a delle modifiche che potrebbero introdurre nuovi errori all'interno del sistema.

10. Test Deliverables

I documenti rilasciati durante e al termine della fase di test sono:

- Test Plan;
- System Test Case;
- Category Partition;
- Test Case Specification;
- Unit Test Report;
- Test Execution Report;
- Test Incident Report;
- Test Summary Report;

11. Materiale per il Testing

Per svolgere le attività di testing è necessario un computer e una copia del DB in locale.

12. Necessità di training per il Team

Il Team ha già affrontato nella fase preliminare del corso di IS il testing, per cui non saranno previste sessioni di training.

13. Responsabilità

Ogni team member sarà responsabile del testing della gestione alla quale è stato assegnato. Il Team non verrà diviso in "tester" e "developer" per evitare overhead di comunicazione.

14. Glossario

- **Developer:** figura professionale che si occupa dello sviluppo di applicazioni software;
- **Errore:** una misura della differenza stimata tra il valore osservato o calcolato di una quantità e ilsuo valore reale;
- **Rischio:** è la potenzialità che un'azione o un'attività scelta (includendo la scelta di non agire) portia una perdita o ad un evento indesiderabile;
- **Tester:** una persona, una macchina o dispositivo utilizzato per verificare se un sistema o componente funziona correttamente;
- **Testing:** processo o metodo per trovare errori in un'applicazione o un programma software in modo che l'applicazione funzioni in base ai requisiti dell'utente finale;
- **Tool:** strumento software utilizzato per ottenere un dato risultato o semplificare delle operazioni;