Università di Salerno
Corso di Ingegneria del Software

ClickEat
Test Plan
Versione 1.0

Progetto: ClickEat	Versione: 1.0
Documento: Test Plan	Data: 06/02/2019

Partecipanti:

Nome	Matricola
Cupito Andrea [CA]	0512104538
Amoriello Luca [AL]	0512104658
Pasquariello Giovanni [PG]	0512105020
Russo Vincenzo [RV]	0512104130

Scritto da:	Cupito Andrea

Revision History

Data	Versione	Descrizione	Autore
06/01/2019	1.0	Stesura del Test Plan	Membri del gruppo

Sommario

1. Introduzione

Lo scopo di tale documento è quello di pianificare l'attività di test del sistema ClickEat al fine di verificare se esistono divergenze tra il comportamento atteso e quello osservato. In questa attività andremo a rilevare eventuali errori prodotti nel codice, al fine di rilasciare un sistema capace di soddisfare l'utente. Le attività di Testing sono state pianificare per le seguenti componenti:

1. Gestione Sala;
2. Gestione Ristorante;
3. Gestione Utente;
4. Gestione Ordinazione;

Si noti che verranno testate soltanto le funzionalità implementate e descritte nell'ODD. Oltre alla gestione del test delle funzionalità, vengono pianificate le responsabilità del team e lo scheduling del test.

2. Documenti correlati

Il Test Plan ha una stretta relazione con i documenti ad esso precedente, poiché prima di passare alla fase di testing il sistema è stato implementato e strutturato sulla base di questi ultimi. Questo permette di rilevare eventuali differenze tra il comportamento atteso e quello osservato del sistema. Di seguito riportiamo la relazione che intercorrono tra il Test Plan e la documentazione precedente.

2.1. Relazioni con il documento di analisi dei requisiti (RAD)

La relazione tra test Plan e RAD riguarda principalmente i requisiti funzionali e non funzionali del sistema, poiché i test che saranno effettuati sulle funzionalità faranno riferimento a ciò che è stato espresso nel RAD.

2.2. Relazioni con il System Design Document (SDD)

Nel System Design Document abbiamo suddiviso il nostro sistema in sottosistemi e l'architettura in tre livelli: Presentation layer, Application Logic layer e Storage layer. Il test dei vari componenti dovrà rimanere fedele a tale suddivisione.

2.3. Relazioni con l'Object Design Document (ODD)

Il test d'Integrazione farà riferimento alle interfacce delle classi definite nell'ODD.

3. Panoramica del sistema

Come già definito nel System Design Document la struttura del nostro sistema è divisa secondo una architettura “Three Thiers” cioè a tre livelli: Presentation layer, Application Logic layer, Storage layer. In questo caso il livello più alto interagisce con il livello applicativo che a sua volta si occuperà di eseguire le operazioni nel Database di ClickEat, cercando di garantire il più basso accoppiamento e il maggior grado di coesione possibile tra le varie classi. Il sistema è stato, inoltre, suddiviso in sottosistemi più piccoli, in particolare la suddivisione è avvenuta mediante le gestioni, definite nel paragrafo 1 di questo documento. Ognuna di questa gestione prevede operazioni per la gestione dei dati (modifica, inserimento, ecc.) e saranno proprio quest’ultime ad essere testa nel corso del testing.

4. Funzionalità da testare e da non testare

Sono state testate tutte le componenti delle classi Manager che vanno ad interagire con i dati persistenti. Gli unici metodi che non verranno direttamente testati con Junit sono i metodi interni alle classi Bean (getter e setter, per meglio intenderci); i quali sono certamente funzionanti in quanto creati interamente tramite Eclipse IDE.

5. Criteri pass/fail

I dati in input (test case) saranno suddivisi in classi di equivalenza, ovvero verranno raggruppati in insiemi dalle comuni caratteristiche, per i quali sarà sufficiente testare un solo elemento rappresentativo della classe. Un input avrà superato il test se l’output sarà quello atteso, cioè quello che è stato specificato del membro del team che si occuperà del testing su quello specifico test case.

6. Approccio

Le tecniche di testing adottate riguarderanno inizialmente il testing delle singole unità del sistema, in modo da testare la correttezza di ciascuna di esse. A seguire verrà effettuato il testing di Integrazione, che si focalizzerà sul test delle interfacce di tali unità. Infine, verrà effettuato il testing di sistema, che vedrà come protagonista del testing l'intero sistema assemblato delle sue componenti.

6.1. Testing di Unità

Durante questa fase, verranno ricercate le condizioni di fallimento, isolando le singole componenti ed utilizzando test driver e stub, cioè implementazioni parziali di componenti che dipendono o da cui dipendono le componenti testate. La strategia utilizzata per questa fase è il Black-Box, che si focalizza sul comportamento Input/Output, ignorando l'effettiva codifica della componente. Al fine di minimizzare il numero di test case, i possibili input verranno partizionati in specifiche classi di equivalenza. Gli stati erronei individuati in questa fase, come in qualsiasi altra fase di testing, dovranno essere prontamente comunicati e corretti.

6.2. Testing di Integrazione

In questa fase si procederà all'integrazione delle componenti di una funzionalità che verranno testate nel complesso attraverso una strategia Sandwich. Quest'ultima ci permetterà di testare simultaneamente i layer ad alto e basso livello. Tale scelta è giustificata dal fatto che la codifica del sistema è stata effettuata in parallel, lavorando in maniera sincrona sia al front-end che al back-end.

6.3. Testing di Sistema

Lo scopo di questa fase è quello di dimostrare che il sistema soddisfi effettivamente i requisiti richiesti e sia, quindi, pronto al rilascio. Come per il testing di unità, si cercherà di testare le funzionalità più importanti per l'utente e quelle che hanno una maggiore probabilità di fallimento. Principalmente il testing si è concentrato sulle componenti con le quali l'utente può effettuare modifiche al sistema (form, e tutto ciò che prevede l'inserimento dei dati), ciò permetterà di testare soprattutto le politiche di controllo adottate dal team.

7. Sospensione e ripresa

7.1. Criteri di sospensione

La fase di testing del sistema verrà sospesa quando si raggiungerà un compromesso tra qualità del prodotto e costi dell'attività di testing. Il testing verrà quindi portato avanti quanto più possibile nel tempo senza però rischiare di ritardare la consegna finale del progetto.

7.2. Criteri di ripresa

L'hardware necessario per l'attività di test è un pc con connessione ad internet, dal momento che il database del sistema è stato caricato online per renderlo disponibili a tutti.

8. Materiale per il testing

8.1. Criteri di ripresa

L'hardware necessario per l'attività di test è un pc con connessione ad internet, dal momento che il database del sistema è stato caricato online per renderlo disponibili a tutti.

9. **Test cases**

9.1. **creaUtente**

Category partition per i test case del metodo creaUtente:

Parametro: nomeUtente **Formato [FNU]** : ^[A-Za-z\s]{3,35}$	
Lunghezza [LNU]	1. <3 or >35 [error] 2. >=3 and <=35 [LNU OK]
Formato [FNU]	1. Non rispetta il formato [if LNU_OK] [error] 2. Rispetta il formato [if LNU_OK] [property FNU_OK]

Parametro: cognomeUtente **Formato [FCU]** : ^[A-Za-z\s]{3,35}$	
Lunghezza [LCU]	1. <3 or >35 [error] 2. >=3 and <=35 [LCU OK]
Formato [FCU]	1. Non rispetta il formato [if LCU_OK] [error] 2. Rispetta il formato [if LCU_OK] [property FCU_OK]

Parametro: ruoloUtente **Formato [FRU]** : ^((\b(Cameriere)\b)*(\b(Amministratore)\b)*(\b(Cassiere)\b)*)+$	
Lunghezza [LRU]	1. !=Cameriere || !=Cassiere || !=Amministratore [error] 2. ==Cameriere || ==Cassiere || ==Amministratore [LRU_OK]
Formato [FRU]	1. Non rispetta il formato [if LRU_OK] [error] 2. Rispetta il formato [if LRU_OK] [property FRU_OK]

Parametro: idLogin **Formato [FIL]** : ^[a-z]{1}\.{1}[a-z]{4,34}$	
Lunghezza [LIL]	1. <6 or >36 [error] 2. >=6 and <=36 [LIL_OK]
Formato [FIL]	1. Non rispetta il formato [if LIL_OK] [error] 2. Rispetta il formato [if LIL_OK] [property FIL_OK]
Uguaglianza [UIL]	1. Uguale ad un dato già presente [if LIL_OK] [if FIL_OK] [error] 2. Differente dai dati già presenti [if LIL_OK] [if FIL_OK] [property UIL_OK]

Parametro: passwordUtente **Formato [FPU]** : ^[A-Za-z0-9]{8,36}$	
Lunghezza [LPU]	1. <8 or >36 [error] 2. >=8 and <=36 [LPU_OK]
Formato [FPU]	1. Non rispetta il formato [if LPU_OK] [error] 2. Rispetta il formato [if LPU_OK] [property FPU_OK]

9.2. **valoriLogin**

Category partition per i test case del metodo valoriLogin:

Parametro: idLogin **Formato [FIL]** : ^[a-z]{1}\.{1}[a-z]{4,34}$	
Lunghezza [LIL]	1. <6 or >36 [error] 2. >=6 and <=36 [LIL_OK]
Formato [FIL]	1. Non rispetta il formato [if LIL_OK] [error] 2. Rispetta il formato [if LIL_OK] [property FIL_OK]
Uguaglianza [UIL]	1. Uguale ad un dato già presente [if LIL_OK] [if FIL_OK] [error] 2. Differente dai dati già presenti [if LIL_OK] [if FIL_OK] [property UIL_OK]

Parametro: passwordUtente **Formato [FPU]** : ^[A-Za-z0-9]{8,36}$	
Lunghezza [LPU]	1. <8 or >36 [error] 2. >=8 and <=36 [LPU_OK]
Formato [FPU]	1. Non rispetta il formato [if LPU_OK] [error] 2. Rispetta il formato [if LPU_OK] [property FPU_OK]
Uguaglianza [UPU]	1. Differente dai dati presenti [if LPU_OK] [if FPU_OK] [error] 2. Uguale ad un dato già presente[if LPU_OK] [if FPU_OK] [property UPU_OK]

9.3. **creaIngrediente**

Category partition per i test case del metodo creaIngrediente:

Parametro: nomeIngrediente **Formato [FNI]** : ^[A-Za-z\s]{4,36}$	
Lunghezza [LNI]	1. <4 or >36 [error] 2. >=4 and <=36 [LNI_OK]
Formato [FNI]	1. Non rispetta il formato [if LNI_OK] [error] 2. Rispetta il formato [if LNI_OK] [property FNI_OK]
Uguaglianza [UNI]	1. Uguale ad un dato già presente [if LNI_OK] [if FNI_OK] [error] 2. Differente dai dati già presenti [if LNI_OK] [if FNI_OK] [property UNI_OK]

9.4. **creaTavolo**

Category partition per i test case del metodo creaTavolo:

Parametro: numeroTavolo **Formato [FNT]** : ^[0-9]{1,3}$	
Lunghezza [LNT]	1. 1< or >3 [error] 2. >=1 and <=3 [LNT_OK]
Formato [FNT]	1. Non rispetta il formato [if LNT_OK] [error] 2. Rispetta il formato [if LNT_OK] [property FNT_OK]
Uguaglianza [UNT]	1. Uguale ad un dato già presente [if LNT_OK] [if FNT_OK] [error] 2. Differente dai dati già presenti [if LNT_OK] [if FNT_OK] [property UNT_OK]

9.5. **creaPiatto**

Category partition per i test case del metodo creaPiatto:

Parametro: nomePiatto **Formato [FNP]** : ^[A-Za-z\s]{4,36}$	
Lunghezza [LNP]	1. <3 or >35 [error] 2. >=3 and <=35 [LNU OK]
Formato [FNP]	1. Non rispetta il formato [if LNP_OK] [error] 2. Rispetta il formato [if LNP_OK] [property FNP_OK]
Uguaglianza [UNP]	1. Uguale ad un dato già presente [if LNP_OK] [if FNP_OK] [error] 2. Differente dai dati già presenti [if LNP_OK] [if FNP_OK] [property UNP_OK]

Parametro: categoriaPiatto **Formato [FCP]** : ^((\b(Primi)\b)*(\b(Secondi)\b)*(\b(Contorni)\b)*)+$	
Lunghezza [LCP]	1. !=Primi || != Secondi || !=Contorni [error] 2. 2. ==Primi || == Secondi || ==Contorni [LCP_OK]
Formato [FCP]	1. Non rispetta il formato [if LCP_OK] [error] 2. Rispetta il formato [if LCP_OK] [property FCP_OK]

Parametro: prezzoPiatto **Formato [FPP]** : ^[0-9]{1,5},[0-9]{2}$	
Lunghezza [LPP]	1. <8 or >14 [error] 2. >=8 and <=14 [LPP_OK]
Formato [FPP]	1. Non rispetta il formato [if LPP_OK] [error] 2. Rispetta il formato [if LPP_OK] [property FPP_OK]

Parametro: listaIngredienti **Formato [FLI]** : ^((,{0,1})([A-Za-z\s]{4,36})(,{0,1}))+$	
Lunghezza [LLI]	1. <4 or >150 [error] 2. >=4 and <=150 [LLI_OK]
Formato [FLI]	1. Non rispetta il formato [if LLI_OK] [error] 2. Rispetta il formato [if LLI_OK] [property FLI_OK]

10. Pianificazione del Testing

Il Team di Testing dovrà essere composto soltanto da persone che hanno un conoscenza completa e approfondita del sistema e delle tecniche di testing che andremo a adoperare. I documenti Test Plan e Test case Specification saranno la base sulla quale si fonderanno le attività di Testing; è quindi necessaria una conoscenza completa di quest'ultimi. Il Team che si dedicherà all'attività di testing avrà come principale obiettivo la ricerca di fault (algoritmici, principalmente); i quali dovranno essere prontamente segnalati agli sviluppatori per pervenire ad una rapida risoluzione della problematica individuata. In presenza di errori si effettuerà un Refactoring sul codice al fine di risolvere l'errore. L'attività di testing è fondamentale nello sviluppo di un sistema software in quanto la mancanza di tale attività può portare al completo fallimento del sistema.

10.1. Determinazione dei ruoli

Il Team dedicato all'attività di Testing è composto da:

- Amoriello Luca: il quale si occuperà della fase di Unit Testing
- Cupito Andrea: il quale si occuperà della fase di System Testing