

Relatore

Ringraziamentiringraziamenti

*Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. , relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante

Ringrazio il mio tutor aziendale Roberto per avermi trasmesso con tenacia e passione le conoscenze del settore.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute. In particolare

,

SommarioSommario

*Sommario

L'obiettivo del presente documento è mostrare il lavoro svolto durante il periodo di stage, dal laureando Fabio Pantano. Lo scopo principale del progetto è analizzare uno dei rami del CRM : il ticketing.

Il primo passo per lo sviluppo del modulo web relativo al ticketing è l'analisi del problema con la conseguente raccolta dei requisiti. In prossimo passo è andare a definire in che modo l'utente si interfaccia con le funzioni del modulo e quindi con quali tecnologie. Durante questo periodo è iniziata un'ulteriore fase di analisi per introdurre nuove funzionalità, come il commento in tempo reale.

L'IDE utilizzato è Eclipse e il linguaggio per lo sviluppo del Model è Java, supportato da vari framework (struts2, maven).

tableofcontents

lof

lot

Introduzione

L'azienda

CWBI è una società italiana di sviluppo software specializzata nella fornitura di soluzioni internet e mobile per banche e assicurazioni. CWBI offre una vasta gamma di servizi quali:

Pianificazione Il lavoro si svolge nelle 300 ore obbligatorie per il tirocinio formativo e si suddivide in:
 Studio ed analisi dell'architettura già presente in azienda;
 Raccolta dei requisiti del prodotto atteso;
 Refactoring di codice di classi esistenti per adattarlo al prodotto;
 Sviluppo del prodotto;
 Test.

Le ore si sono distribuite in 8 settimane lavorative, a loro volta suddivise nel particolare dedicando: [h] IXI

Durata in ore Attività

- 40 Formazione iniziale e introduzione tecnologie utilizzata JAVA/JEE lato server
- 40 Formazione soluzione *baseapp* con apprendimento *framework* di lavoro
- 68 Analisi e raccolta requisiti progetto marketing
- 122 Sviluppo soluzione (Realizzazione soluzione software in java *back-end* e sviluppo *front-end*)
- 15 Test e supporto UAT
- 15 Documentazione progetto

Tabella della pianificazione del lavoro

Analisi dei requisiti

In questo capitolo vengono analizzati tutti i requisiti individuati, come ad esempio apertura ticket, chiusura ecc..

Lo scopo è definire con chiarezza la funzione che svolge ogni requisito all'interno dell'applicazione, andando ad assegnare

APERTO

CHIUSO

Lo stato APERTO viene assegnato al ticket la prima volta che viene aperto e rimarrà così fino alla conclusione delle att

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese

Attori

[H] [width=0.9]usecase/utenti-primari Gerarchia degli attori

Utente Non Autenticato: utente che ancora non ha effettuato l'accesso alla webapp.

Utente Autenticato: utente che ha effettuato l'accesso.

Utente Cliente: utente autenticato con permessi di livello Cliente. Rappresenta i clienti esterni all'azienda.

Utente CWBI: utente autenticato con permessi di livello CWBI. Rappresenta i dipendenti dell'azienda CWBI.

Utente Amministratore: utente autenticato con permessi Amministratore. Rappresenta uno o più dipendenti CWBI

Elenco UC01 - Autenticazione

[H] [width=0.9]usecase/UC01 UC01

UC01

UC02 - Errore Autenticazione

UC02

Si verificano problemi con l'accesso alla webapp;

Viene mostrato un errore che informa l'utente del fallimento dell'operazione.

UC02

UC03

UC04 - Accesso Modulo Ticket [H] [width=0.9]usecase/UC04 UC04

UC04

UC05 - Visualizza Lista Ticket

[H] [width=0.8]usecase/UC05₆₀₇UC05 – UC06 – UC07 – UC08

UC06

Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti. Il requisito è così descritto:

codice identificativo: ogni codice identificativo è univoco e definito seguendo lo standard di codifica **R[Importanza]**
Importanza:

Tipologia:

Codice: identificatore univoco del requisito in forma gerarchica.

classificazione: viene riportata l'importanza del requisito per facilitare la lettura;

descrizione;

fonte: origine del requisito.

Requisiti funzionali

[!h] Tabella del tracciamento dei requisiti funzionali PX1

Requisito Descrizione

R1F1 L’utente ha la possibilità di effettuare il login

R1F2 L’utente con permesso Cliente ha la possibilità di poter accedere al modulo Ticket

R1F3 L’utente con permesso CWBI ha la possibilità di poter accedere al modulo Ticket

R1F4 L’utente amministratore ha la possibilità di poter accedere ai vari moduli

[H] PXI

Requisito Descrizione

R1F26 Il modulo, visualizza i bottoni di modifica su ogni riga, solo se il ticket è stato aperto dall'utente attualmente

R1F26.1 Il modulo, per ogni riga, fornisce il bottone di dettaglio del ticket

R1F26.2 Il modulo, per ogni riga, fornisce il bottone di modifica del ticket

R1F26.3 Il modulo, per ogni riga, fornisce il bottone di elimina del ticket

R1F27 Il modulo permette si effettuare il salvataggio del ticket

R1F28 Il modulo visualizza un errore in caso il salvataggio non vada a buon fine

R2F28.1 Il modulo visualizza un errore in caso il titolo sia vuoto

R2F28.2 Il modulo visualizza un errore in caso la data di scadenza sia vuota

R2F39 Il modulo visualizza un avviso in caso non ci siano ticket da visualizzare

R2F30 Il modulo visualizza un avviso dopo l'eliminazione di un ticket

Requisiti di vincolo

[!h] Tabella del tracciamento dei requisiti di vincolo PXI

Requisito Descrizione

R1V1 Il modulo deve limitare la modifica di un ticket

R1V1.1 Il modulo permette la modifica del ticket all'utente che lo ha aperto

R1V1.2 Il modulo permette la modifica del ticket all'utente amministratore

R1V2 Il modulo deve limitare l'eliminazione di un ticket

R1V2.1 Il modulo permette l'eliminazione del ticket all'utente che lo ha aperto

R1V2.2 Il modulo permette l'eliminazione del ticket all'utente amministratore

R1V3 Il modulo deve limitare la modifica di un commento

[H] Tabella del tracciamento dei requisiti di vincolo PXI

Requisito Descrizione

R1V3.1 Il modulo permette la modifica del commento all'utente che lo ha aperto

R1V3.2 Il modulo permette la modifica del commento all'utente amministratore

R1V4 Il modulo deve limitare l'eliminazione di un commento

R1V4.1 Il modulo permette l'eliminazione del commento all'utente che lo ha aperto

R1V4.2 Il modulo permette l'eliminazione del commento all'utente amministratore

R1V5 Il modulo deve essere sviluppato in Java

R1V6 Il modulo integra classi preesistenti in altri moduli

R1V7 Il modulo deve essere sviluppato secondo l'architettura dell'azienda

R1V8 Devono essere utilizzati i framework previsti dall'azienda per lo sviluppo delle applicazioni aziendali

Progettazione e codifica

Il capitolo inizialmente presenta gli strumenti e le tecnologie analizzate e utilizzate per la realizzazione del prodotto

Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

*HTML5 Tecnologia standard per la creazione di pagine web. È studiata e conosciuta con il corso di Tecnologie Web

*CSS Tecnologia standard per la creazione di pagine web e il loro abbellimento. Fornisce una vasta gamma di funzioni. Bootstrap è utilizzato nello stile *inline* di *HTML* e le sue classi vengono inserite all'interno del *tag* "class" di *HTML* di *HTML5*. La differenza tra le due versioni di *Bootstrap* 3 e 5 è nella gamma di funzionalità che offrono. La versione 5 è la più recente. Ad oggi si cerca di migrare dalle versioni più vecchie a quella più recente. *Servlet Le *Servlet* permettono di soddisfare o

Il framework è composto da tre elementi principali:

Request Handler: viene mappato ad un URI dallo sviluppatore;

Response Handler: la risposta verrà passata ad un'altra risorsa che la completerà;

Tag: aiutano lo sviluppatore per lo sviluppo.

Per configurare tutti i collegamenti tra i vari elementi e le loro interazioni si utilizza il file ***struts.xml***. In questo file ver

*JQUERY Taconite *JQUERY Taconite* permette di aggiornare DOM multipli utilizzando il risultato di una singola azione. Viene generato un *XML* con le istruzioni per l'aggiornamento dei diversi *DOM*

*Hibernate È un framework che permette di mappare gli oggetti del modello ad un *database* relazionale. Lo sviluppatore

*Spring *Spring* è un framework volto ad aiutare lo sviluppo di applicazioni più o meno complesse attraverso la sua architettura. L'elemento principale di *Spring* è il *Core Container* che ha il compito di creazione e gestione di tutti gli oggetti dell'applicazione. Normalmente un browser può scaricare al massimo due risorse contemporaneamente e questo limite porta ad un caricamento lento

*Apache Maven *Apache Maven* è uno strumento per la gestione delle dipendenze tra un progetto Java e le versioni dei suoi

[H] [width=1.1]diagrammaEvoBase Base di Dati - Tabelle del progetto

Tabella Progetto

Tabella ProgettoUser

Tabella User

Tabella Ticket

Tabella Ticket

Tabella TicketItem

Tabella Allegato

Architettura Lo sviluppo dell'applicazione avviene secondo il *pattern* architetturale **MVC** (*Model - View - Controller*). Questo *pattern* permette di dividere e rendere modulare l'applicazione.

Model: si occupa della gestione dei dati, del salvataggio delle risorse e della logica di business;

View: si occupa di visualizzare i dati salvati nel modello, presentandoli secondo uno schema definito.

Controller: ha il compito di gestire la comunicazione tra il modello e la vista ed elaborare gli input dell'utente per poi

[H] [width=0.4]MVC Schema MVC

Utilizzare il *pattern MVC* permette di avere dei **vantaggi**:

Manutenzione: la suddivisione in componenti rende la manutenzione dell'applicazione più semplice, andando a concentrarsi su una sola parte.

Scalabilità: con l'aumentare delle esigenze l'applicazione richiederà degli aggiornamenti che saranno meglio integrabili.

Testabilità: senza il *pattern MVC*, per eseguire il test su una parte dell'applicazione, bisognerebbe eseguire la diagnosi su tutta l'applicazione.

Separazione delle responsabilità: ogni componente ha un compito ben preciso e non andrà a interessarsi delle parti che non gli competono.

*Model Si parte da un modello preesistente e strutturato secondo gli standard aziendali. Infatti questa parte è divisa in

[H] [width=0.6]SchemaModel Struttura Model CWBI

Come detto in precedenza alcune delle classi già presenti sono utilizzate per supportare le nuove entità introdotte con il nuovo modello.

Il primo passo per la creazione di una nuova classe è mappare i suoi attributi all'interno del file xml che conterrà la mappa delle entità. Prendiamo come esempio la classe Ticket. Allora creeremo il file *ticket.xml* e scriveremo la tabella *ticket-a* con tutti gli attributi.

Dopo si comincia a codificare la classe **TicketDB.java** che rappresenta l'oggetto vero e proprio che sarà utilizzato nella base di dati. Le nomenclature *DB* dopo il nome della classe è uno standard dell'azienda inserito per ogni nuovo modello che si introduce.

Successivamente si creano due classi: **TicketDao** e **TicketDaoHibernate**.

Partiamo con **TicketDao** che rappresenta l'interfaccia in cui saranno presenti le firme di tutte le funzioni per la manipolazione dei dati.

La classe **TicketDaoHibernate** invece è l'implementazione dell'interfaccia **TicketDao**, che implementa quindi le funzioni per la manipolazione dei dati.

In realtà, l'esecuzione effettiva della funzione CRUD richiesta, non avviene dentro TicketDaoHibernate ma all'interno di TicketDaoHibernate.

In generale, **TicketDaoHibernate** implementa le funzioni dell'interfaccia a cui si riferisce, ma le operazioni CRUD vengono delegate a TicketDaoHibernate.

Il prossimo passo è creare le classi service: **TicketService** e **TicketServiceImpl**.

La classe **TicketService** è l'interfaccia al cui interno troviamo le firme delle funzioni che la webapp consentirà di svolgere.

TicketServiceImpl invece è l'implementazione delle funzioni di TicketService. All'interno di ogni funzione, il dato viene manipolato e salvato.

È importante notare che la funzione di ricerca non è una funzione *CRUD* e viene implementata nel service. Infatti nella classe TicketService.

Allora tutte le funzioni all'interno delle applicazioni si riducono sempre a delle semplici funzioni **CRUD**.

L'ultima fase è creare le classi manager. Queste classi non sempre sono necessarie e fungono da supporto per le classi service. Nel progetto **non** sono state codificate classi manager.

*Controller Lo scopo dei *controller* all'interno dell'applicazione è quello di gestire le interazioni tra la parte di *front-end* e la parte di *back-end*.

Il *controller* ha il compito di *istanziare* le classi del Model, richiamarne le funzioni per avere un risultato e inviarlo poi alla *view*. Un'altra annotazione Spring presente è **@Autowired** che serve per indicare le dipendenze dei *bean* (classi). Infatti all'interno dei *controller* si trovano le annotazioni **@Autowired**.

CWBI struttura i controller in due cartelle distinte. Prendiamo come esempio la classe Ticket del modello:

Cartella Form

TicketForm: questa classe identifica i campi di input presenti alla creazione o alla modifica di un ticket, detti appunto *form*. Le caratteristiche di TicketForm saranno adeguate alla controparte dell'oggetto Ticket nel modello. Infatti il *controller* TicketForm.

TicketSearchForm: questa classe identifica i campi di input presenti alla ricerca di un ticket. L'utente per effettuare la ricerca deve compilare i campi di input. Le caratteristiche di tale classe sono redatte in base ai tipi di filtri che si vogliono fornire all'utente e devono essere adeguate.

Cartella Action

Le classi presenti in questa cartella, sono dette *Action* e sono i veri e propri *controller* che svolgono le varie funzioni.

TicketAction: come si può leggere, la classe non presenta la nomenclatura *Form*. Infatti questa *Action* si occupa di gestire le richieste per la creazione e la modifica di un ticket.

Si trovano diverse funzioni, come ad esempio la funzione di caricamento della pagina di dettaglio di un Ticket. In questa pagina si trova la *form* per la creazione e la modifica di un ticket.

TicketFormAction: questo *controller* si occupa di gestire le *Action* che riguardano la pagina di creazione e modifica di un ticket.

Quando si entra nella pagina di creazione di un ticket, i campi dell'oggetto **TicketForm** sono inizializzati vuoti dall'*Action* TicketFormAction.

Alla fine, con l'oggetto **TicketService** il nuovo oggetto **Ticket** viene salvato.

TicketSearchFormAction: l'ultimo *controller* è utilizzato per le pagine di ricerca di un ticket e utilizzano l'oggetto **TicketForm**.

Viene quindi utilizzato l'oggetto **TicketService** per richiamare la funzione di ricerca che prende in input un oggetto **TicketForm**.

*View L'ultimo componente dell'architettura è la **View** che ha il compito di visualizzare i dati secondo una logica definita.

Le pagine che compongono la webapp sono file *jsp* che permettono di scrivere codice con standard HTML o XML, ma anche con standard CSS.

La **View** è supportata anche da diversi *framework* come *Bootstrap* che fornisce delle classi per personalizzare il contenuto. Alle pagine jsp è affiancata un'estensione detta *JSTL* che mette a disposizione dei tag per la visualizzazione dei dati in modo sicuro. Per l'interazione tra modello, controller e view entra in gioco un'ulteriore *framework*, senza il quale, non sarebbe possibile.

L'utente, nell'utilizzo della webapp, interagisce con gli elementi messi a disposizione dalla *View* e richiama delle specifiche. Design Pattern I **Design Pattern** sono soluzioni generali utilizzate per risolvere problemi ricorrenti durante lo sviluppo. Possiamo riconoscere tre famiglie per i *design pattern*:

Comportamentali: definiscono le interazioni tra gli oggetti e distribuiscono le responsabilità.

Creazionali: si occupano di come creare gli oggetti.

Strutturali: provvedono a definire la struttura delle classi, degli oggetti e come essi sono composti.

L'azienda *CWBI* ha applicato i seguenti *design pattern* per la codifica delle loro applicazioni. Tali pattern sono anche presentati nel diagramma.

*Dependency Injection La *Dependency injection* è una tecnica che si occupa di separare la creazione di un oggetto dalla logica di business. Questo *pattern* porta ad avere vantaggi come il riutilizzo, la testabilità e la manutenzione del codice.

*Inversion of Control L' ***Inversion of Control***, detto anche ***IoC*** è un design pattern molto importante ed è uno. Normalmente il flusso di un'applicazione è determinato dagli oggetti e quindi dal codice che la compongono. Con ***IoC*** il flusso è invertito. Un esempio di framework che applica ***IoC*** è ***Spring*** che introduce delle annotazioni come: ***@Component***, ***@Service***, ***@Repository***.
*Decorator [H] [width=0.6]decorator Pattern decorator

Il pattern ***Decorator*** è un pattern strutturale che permette di introdurre nuove funzionalità e comportamenti ad un oggetto.
Component: rappresenta l'interfaccia dell'oggetto da creare;
ConcreteComponent: è l'oggetto a cui verranno aggiunte le nuove caratteristiche;
Decorator: è l'interfaccia dei Decorator che aggiungeranno le nuove funzioni;
ConcreteDecorator: rappresenta gli oggetti Decorator che hanno il compito di aggiungere le nuove funzionalità al Component.
*Data Access Object [H] [width=0.7]DAO Pattern DAO Il pattern ***Data Access Object***, detto anche ***Dao***, è un pattern di design.

Mette a disposizione un'interfaccia che mappa le operazioni sui dati alle chiamate per il database. In generale, facilita l'accesso ai dati.
Il seguente capitolo ha lo scopo di mostrare le tecniche di verifica e validazione del progetto, secondo le linee guida. Verificare un prodotto ha l'obiettivo di controllare se l'introduzione di nuovi elementi nel codice ha generato dei problemi. La validazione serve ad approvare il progetto qualora soddisfatti tutti i requisiti imposti.

Processo di Verifica Il processo di verifica è stato attuato durante tutto lo sviluppo del progetto per verificare le nuove funzionalità. L'approccio all'introduzione di nuovi elementi con le relative funzioni è stato costantemente controllato da ***Roberto Marini***. Quindi, non si procede per codificare il "tutto" perché i requisiti potrebbero essere non soddisfatti e c'è il rischio di perdere tempo.

Debugging Una delle tecniche per verificare il giusto funzionamento dell'applicazione è il ***Debugging***. Effettuando il debug si può individuare il punto di errore. Lo strumento di ***debug*** utilizzato per verificare il progetto è quello messo a disposizione da ***IDE Eclipse***. ***Eclipse*** mette a disposizione un ***breakpoint*** su una riga, l'applicazione si fermerà su quel ***breakpoint*** durante la sua esecuzione e il programmatore potrà vedere lo stato dell'applicazione in quel momento.
[H] [width=0.7]debug IDE Eclipse - Debug Processo di Validazione Il processo di validazione è stato eseguito insieme al processo di verifica.

Oltre alla conformità ai requisiti è ritenuto parte fondamentale di validità del prodotto anche il rispetto dei canoni della qualità. Il tutor ha posto quindi molta attenzione anche su questo aspetto dato che non seguire la struttura designata porta ad un prodotto di qualità inferiore.

Consuntivo finale
Raggiungimento degli obiettivi
Conoscenze acquisite
Valutazione personale
Appendice A
CitazioneAutore della citazione
[type=, title=Acronimi e abbreviazioni, toctitle=Acronimi e abbreviazioni] [type=main, title=Glossario, toctitle=Glossario]

Bibliografia

[heading=subbibliography,title=Riferimenti bibliografici,type=book]
[heading=subbibliography,title=Siti web consultati,type=online]