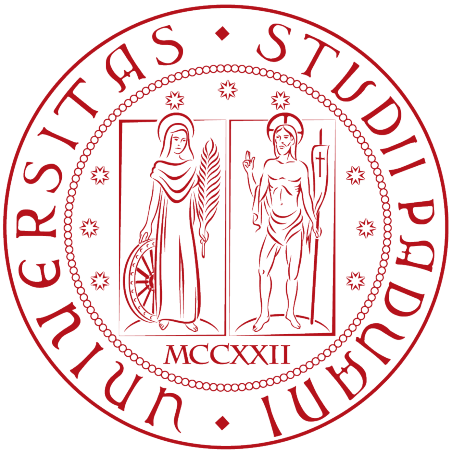
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Modulo web per la gestione di tickets in un contesto bancario - fintech

Tesi di laurea

Relatore

Prof. Paolo Baldan

Laureando

Fabio Pantaleo

Anno Accademico 2022-2023

Ringraziamenti

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

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 durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute. In particolar modo, ringrazio una ragazza speciale che mi è stata vicina durante questi anni.

Padova, Settembre 2023 Fabio Pantaleo

Sommario

L'obiettivo del presente documento è mostrare il lavoro svolto durante il periodo di stage, dal laureando Fabio Pantaleo, presso l'azienda CWBI.
Lo scopo principale del progetto è analizzare uno dei rami del CRM[g] : il ticketing[g]. Il primo passo per lo sviluppo del modulo web relativo al ticketing l'analisi del problema con la conseguente raccolta dei requisiti primari in modo tale da elaborare i casi d'uso della nostra applicazione. Questa prima fase è molto importante per il ciclo di vita del nostro prodotto in quanto rappresenta la base di partenza per la costruzione del nostro modello di dati. Le prime funzionalità individuate sono la creazione, modifica ed eliminazione di un ticket da parte di un utente.
In prossimo passo è andare a definire in che modo l'utente si interfaccia con le funzioni del modulo e quindi con quali componenti, anche visive, deve interagire per raggiungere lo scopo che si è prefissato.
Durante questo periodo è iniziata un ulteriore fase di analisi per introdurre nuove funzionalità, come il commento in tempi asincroni di un ticket da diverse utenti, che arricchiscono il modulo. L'ultimo obiettivo è gestire il tipo di utente che utilizza il modulo per offrire diverse feature in base al livello di autorizzazione che un utente possiede.

L'IDE[g] utilizzato è Eclipse e il linguaggio per lo sviluppo del Model[g] è java, supportato da vari framework (struts2, maven, hibernate, Spring, JEE/ Spring, ecc...). Per il front-end sono utilizzati invece: HTML5, css3, Bootstrap, jsp.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

CWBI una società italiana di sviluppo software specializzata nella fornitura di soluzioni internet e mobile per banche, assicurazioni e industria. Opera nel mercato dell' **Information Communication Technology** e fornisce ai propri clienti un supporto nello studio dei *modelli business* e nella progettazione e realizzazione di software orientati alle ultime tecnologie in questo campo.
CWBI offre una vasta gamma di servizi quali:

- Sviluppo applicazioni e portali web-based
- Sviluppo applicazioni mobile
- Analisi e definizione dei processi organizzativi
- Studi di navigabilità e usabilità

Figura 1.1: CWBI

1.2 Tecnologie utilizzate

Nello sviluppo dei propri prodotti, CWBI si occupa sia della parte di *back-end*[g] sia della parte di *front-end*[g] . Per la prima, è utilizzato Java[g]come linguaggio di programmazione, supportato dai vari *framework*[g]; mentre per la parte destinata alla vista del cliente, sono utilizzati:

- HTML5[g];
- Css[g];
- Boostrap3/5[g];

- JSP[g].

Affiancata anche questa da *framework* come:

- JSTL[g];
- Struts2[g];
- Taconite[g].

Per tracciare gli interventi relativi al codice, l'azienda si avvale di un sistema di *versionamento*[g] con una *repository*[g] in remoto, accessibile grazie a un *toolkit* di Java: SVNKit[g].

1.3 Organizzazione del testo

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

1.4 Struttura

Il testo sarà composto dai seguenti capitoli:

- Introduzione
- Descrizione del sistema attuale
- Descrizione dello stage
- Analisi dei requisiti
- Progettazione e codifica
- Verifica e validazione
- Conclusioni

Capitolo 2

Descrizione del sistema attuale

CWBI ha sviluppato CWGEST[g], un'applicazione usata internamente all'azienda per la gestione, l'organizzazione e il tracciamento delle interazioni con utenti esterni, clienti e non, che supporta il personale offrendo una *way of working*.
L'applicazione è divisa in due menu:

- Amministrativo
- Gestionale

Ogni sezione ha all'interno diversi moduli, rispettivamente:
Amministrativo

- *Administration Module*;
- *User Registration Module*;
- *User Menu Module*;
- *Tracking Module*.

I moduli presenti in questo menu servono per la gestione di *CWGEST* e offrono funzionalità come la registrazione di nuovi utenti per accedere all'applicazione. Tutti questi moduli sono riservati all'utente amministratore e quindi non visibili all' utente generico.
Gestionale

- Ticket
- Offerta
- Consuntivazione
- Progetto Cliente

Alcuni dei moduli di quest' ultimo menu non sono attivi oppure c'è il bisogno, da parte dell'azienda, di eseguire un'operazione di *refactoring*[g] su quelli attualmente in funzione, con l'obiettivo di estendere l'utilizzo dell'applicazione ad agenti esterni come, ad esempio, un cliente.

Capitolo 3

Descrizione dello stage

3.1 Introduzione al progetto

Visto i bisogni dell'azienda, l'obiettivo del modulo **Ticket** è quello di offrire un portale su cui gli utenti registrati, possono aprire, prendere in carico, assegnare ed eliminare dei ticket.
Il modulo è stato pensato sì per i dipendenti interni di CWBI, ma vuole offrire anche ai clienti un modo di segnalare in modo facile e veloce un qualsiasi tipo di problema sulle applicazioni utilizzate. Così anche per CWBI è semplice vedere chi e quando ha inviato una segnalazione, in modo da assegnare un dipendente per trovare una soluzione.
Inoltre per rendere più interattivo il gestionale ed avere un riscontro su quali operazioni sono state effettuate sul ticket, ogni utente potrà lasciare dei commenti in modo da far capire a chi prenderà in carico il ticket, a quale fase della soluzioni è arrivato.
Lavorando su un'applicazione già esistente, una fase molto importante che il progetto prevede sono le attività di refactoring di moduli preesistenti affinché si adattino al nuovo modulo Ticket.

3.2 Obiettivi

Lo stage definisce delle tappe fondamentali da raggiungere, sia per quanto riguarda lo sviluppo di un prodotto, ma soprattutto la formazione della persona è uno dei traguardi principali che l'azienda, il tutor aziendale e lo stagista hanno volontà di completare. Gli obiettivi quindi sono:

- la formazione del tirocinante affinché posso affrontare gli studi e il mondo del lavoro in un'ottica diversa, con nuove conoscenze e con la consapevolezza di un **metodo** di lavoro che aiuterà ad affrontare i futuri problemi attraverso pensiero logico e strategico;
- lo sviluppo del modulo Ticket per supportare e facilitare i rapporti tra azienda e cliente.

3.3 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:

Durata in ore	**Attività**
40	Formazione iniziale e introduzione tecnologie utilizzata JAVA/-JEE lato server
40	Formazione soluzione *baseapp*[g] 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 3.1: Tabella della pianificazione del lavoro

Capitolo 4

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 a questi un'etichetta che li identifica in base alla loro natura.

4.1 Ciclo di vita di un Ticket

Lo studio del ciclo di vita di un ticket è fondamentale per capire le interazioni che questo deve avere con le altre classi della webapp e per comprendere gli elementi fondamentali che compongono la nostra entità. Un ticket può essere nello stato di:

- APERTO
- CHIUSO

Lo stato APERTO viene assegnato al ticket la prima volta che viene aperto e rimarrà così fino allo conclusione delle attività previste nel ticket. Lo stato CHIUSO verrà assegnato quando, chi ha in carico il ticket, ha concluso tutte le operazioni previste e quindi cambia lo stato in chiuso. Quando un ticket viene chiuso potrà essere aperto nuovamente qualora, chi se ne occupa, ritiene che le attività non siano state svolte completamente o se il cliente non rimane soddisfatto delle soluzioni applicate. È importante specificare che un ticket può essere preso in carico da più tecnici durante il suo ciclo di vita. Infatti le operazioni richieste del cliente possono richiedere diversi ruoli; il ticket verrà chiuso da chi lo prende in carico per ultimo.

4.2 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Essendo il progetto finalizzato alla creazione di un tool per l'automazione di un processo, le interazioni da parte dell'utilizzatore devono essere ovviamente ridotte allo stretto necessario. Per questo motivo i diagrammi d'uso risultano semplici e in numero ridotto.

4.2.1 Attori

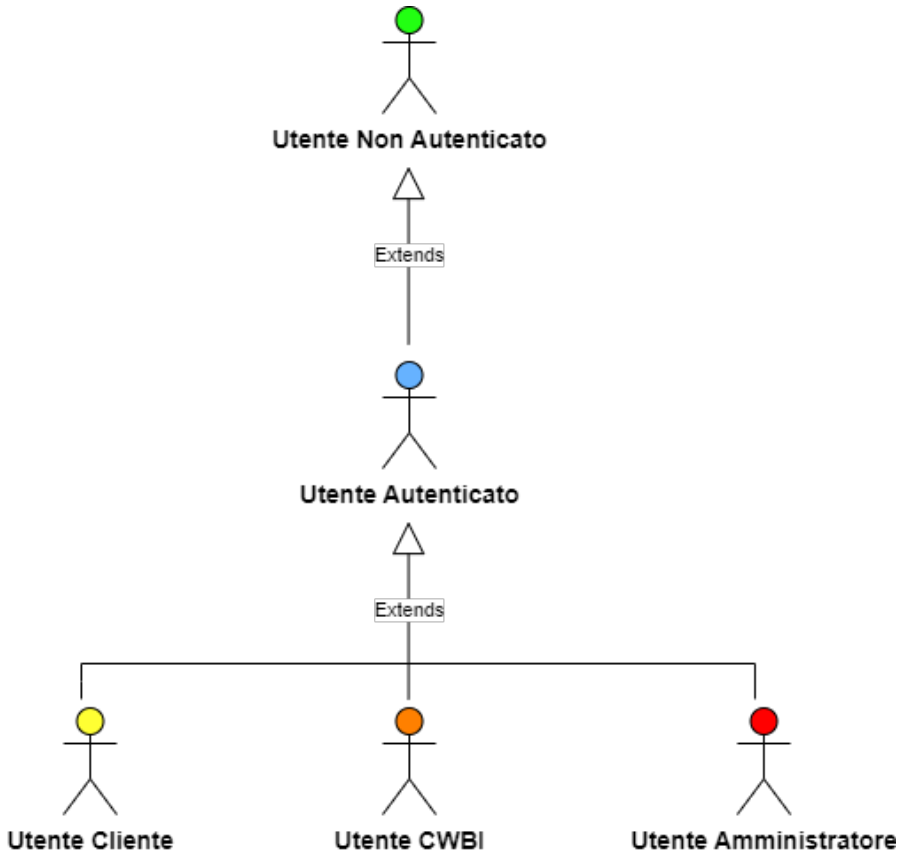


Figura 4.1: Gerarchia degli attori

Utente Non Autenticato: utente che ancora non ha effettuato l'accesso alla webapp.
Utente Autenticato: utente che ha effetuato 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 che hanno la funzione di amministrare la webapp e i contenuti.

4.2.2 Elenco

UC01 - Autenticazione

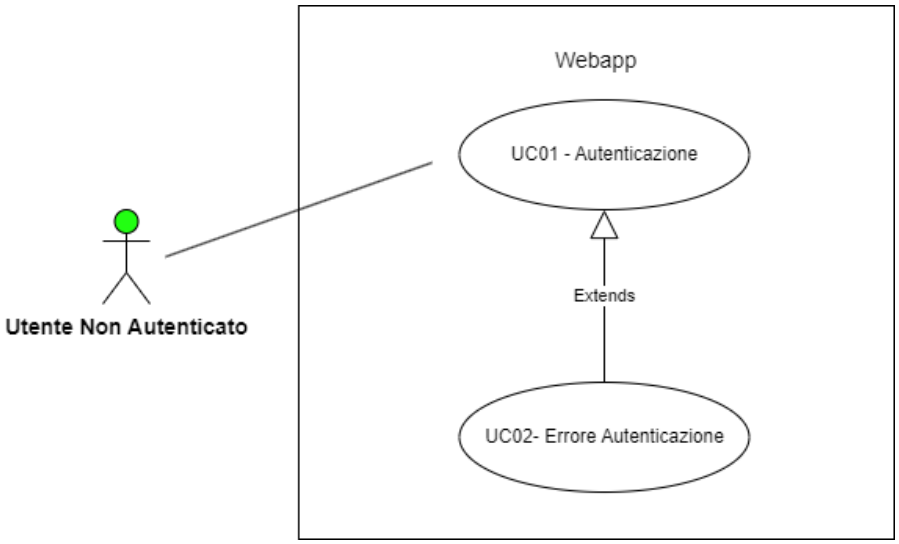


Figura 4.2: UC01

Attore primario	Utente non autenticato
Precondizioni	L'utente non è autenticato.
Postcondizioni	L'utente è autenticato.
Scenario principale	L'utente accede alla webapp
Estensioni	Se l'accesso non va a buon fine, si verifica UC02.

Tabella 4.1: UC01

UC02 - Errore Autenticazione

Attore primario	Utente non autenticato
Precondizioni	L'utente sta tentando di autenticarsi.
Postcondizioni	L'operazione fallisce.

Tabella 4.2: UC02

Scenario principale	1. Si verificano problemi con l'accesso alla webapp; 2. Viene mostrato un errore che informa l'utente del fallimento dell'operazione.

Tabella 4.3: UC02

UC03 - Registrazione Nuovo Utente

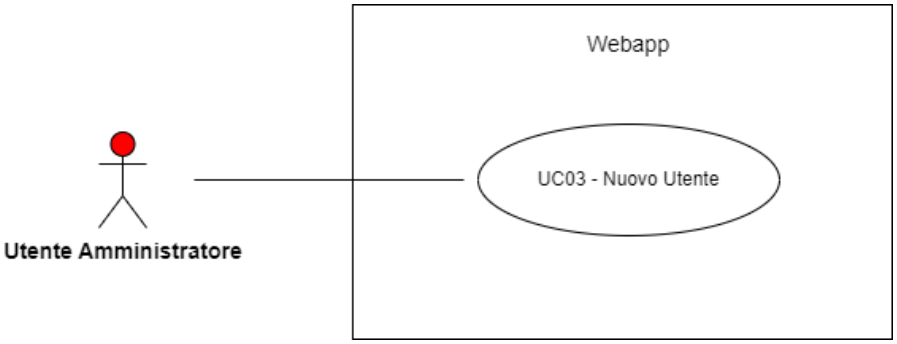


Figura 4.3: UC03

Attore primario	Utente Amministratore
Precondizioni	L'utente amministratore vuole registrare un nuovo utente
Postcondizioni	L'utente amministratore ha registrato un nuovo utente
Scenario principale	L'utente è nel modulo di registrazione utente della webapp

Tabella 4.4: UC03

UC04 - Accesso Modulo Ticket

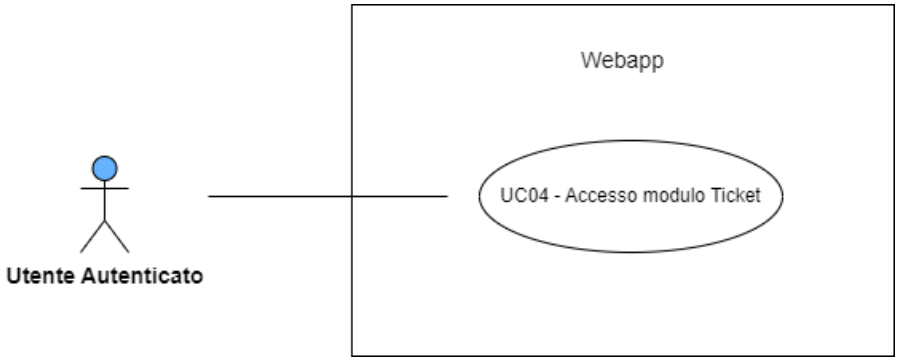


Figura 4.4: UC04

Attore primario	Utente autenticato
Precondizioni	L'utente è nella webapp
Postcondizioni	L'utente è nel modulo ticket
Scenario principale	L'utente accede al modulo ticket e alle sue funzionalità

Tabella 4.5: UC04

UC05 - Visualizza Lista Ticket

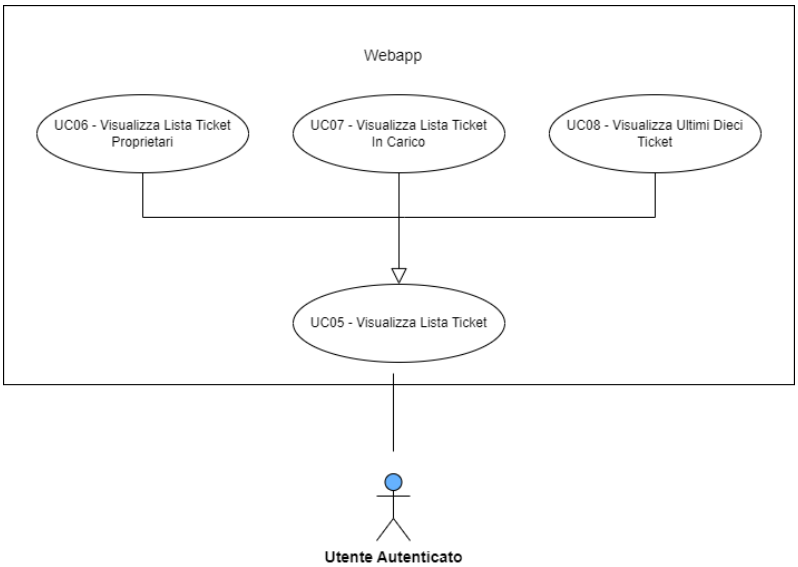


Figura 4.5: UC05 - UC06 - UC07 - UC08

UC06 - Visualizza Lista Ticket Proprietari

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina principale della webapp
Postcondizioni	L'utente visualizza la lista dei ticket aperti da lui.
Scenario principale	L'utente sceglie di visualizzare i ticket proprietari.

Tabella 4.6: UC06

UC07 - Visualizza Lista Ticket In Carico

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina di visualizzazione dei ticket.
Postcondizioni	L'utente visualizza la lista dei ticket presi in carico.
Scenario principale	L'utente sceglie di visualizzare i ticket presi in carico.

Tabella 4.7: UC07

UC08 - Visualizza Lista Degli Ultimi Dieci Ticket

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina di visualizzazione dei ticket
Postcondizioni	L'utente visualizza la lista degli ultimi dieci ticket aperti.
Scenario principale	L'utente sceglie di visualizzare gli ultimi dieci ticket aperti.

Tabella 4.8: UC08

UC05.1, UC05.2

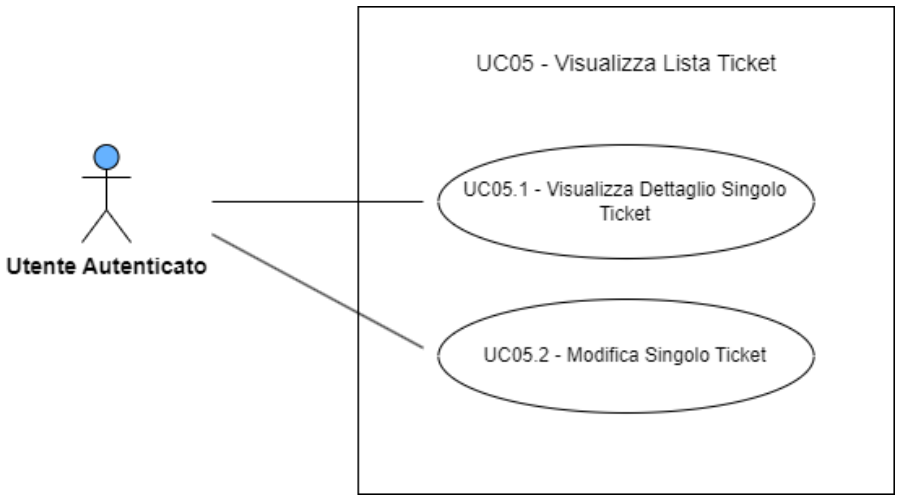


Figura 4.6: UC05.1 - UC05.2

UC05.1

Attore primario	Utente autenticato
Precondizioni	L'utente ha visualizzato la lista dei ticket
Postcondizioni	L'utente visualizza il dettaglio del ticket
Scenario principale	L'utente seleziona il bottone "Dettaglio" per visualizzare i dettagli del ticket

Tabella 4.9: UC05.1

UC05.2

Attore primario	Utente autenticato
Precondizioni	L'utente ha visualizzato la lista dei ticket
Postcondizioni	L'utente modifica del ticket
Scenario principale	L'utente seleziona il bottone "Modifica" per modificare il ticket

Tabella 4.10: UC05.2

UC05.1 - Dettaglio

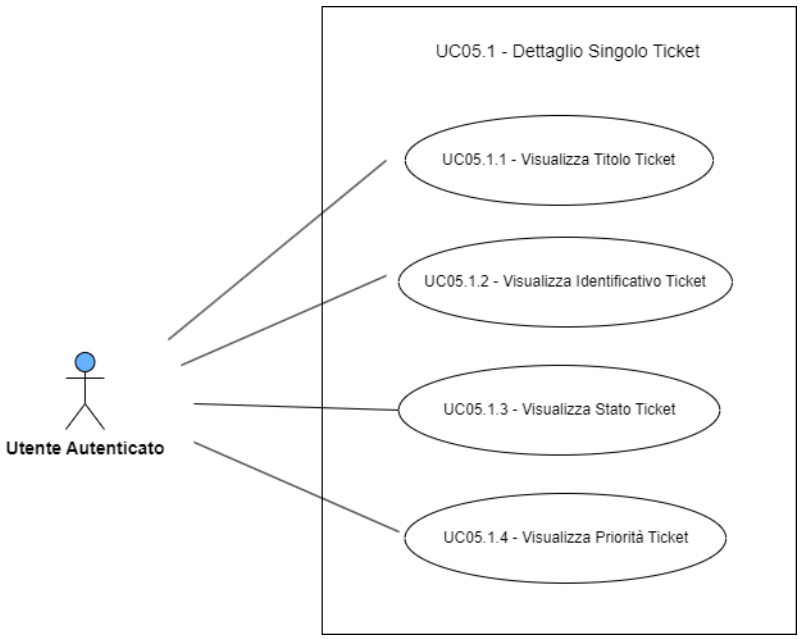
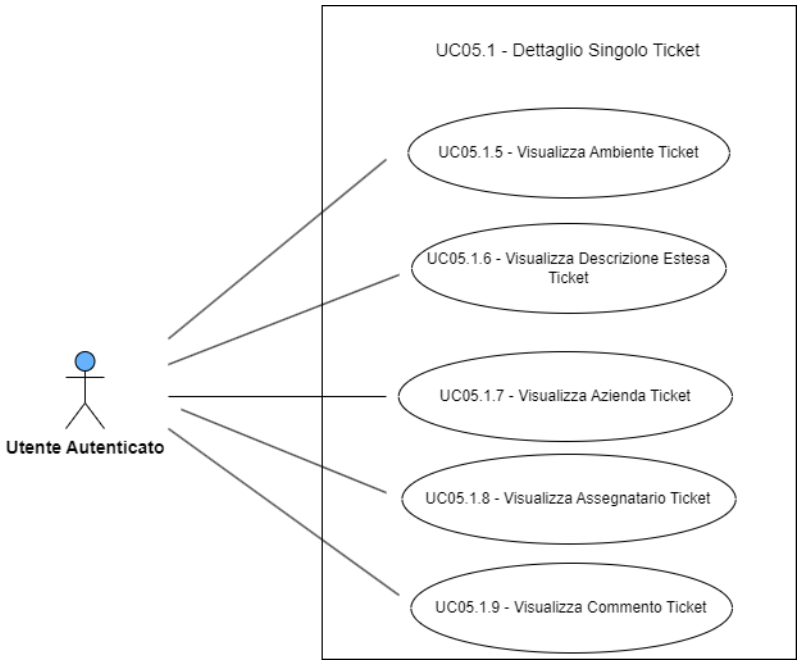


Figura 4.7: UC05.1 - Dettaglio



Attore primario	Utente autenticato
Precondizioni	L'utente ha selezionato da una lista il dettaglio di un ticket
Postcondizioni	L'utente visualizza i dettagli del ticket
Scenario principale	L'utente visualizza tutte le informazioni del ticket: titolo, descrizione, data di apertura, ecc...

Tabella 4.11: UC05.1 - Dettaglio

UC09 - Apertura Nuovo Ticket

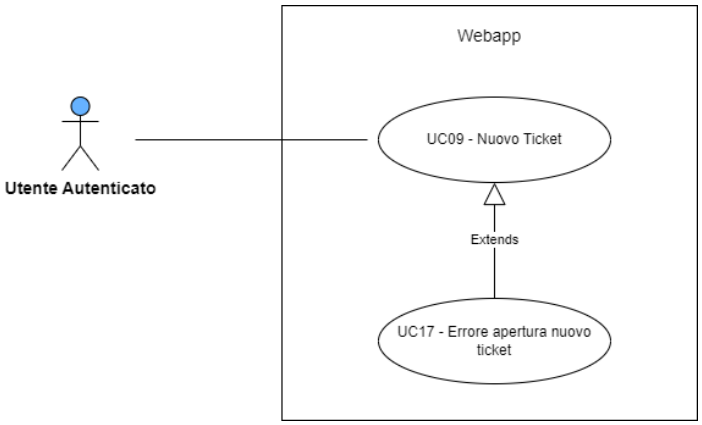


Figura 4.8: UC09

Attore primario	Utente autenticato
Precondizioni	L'utente crea un nuovo ticket compilando i campi: azienda, progetto, titolo, descrizione, data di scadenza, ambiente, priorità, stato, allegato.
Postcondizioni	L'utente apre un nuovo ticket, compilando tutti i campi
Scenario principale	L'utente, accedendo al modulo ticket, seleziona la creazione di un nuovo ticket
Estensioni	Se la creazione di un ticket non va a buon fine, si verifica UC17.

Tabella 4.12: UC09

UC10 - Ricerca Ticket

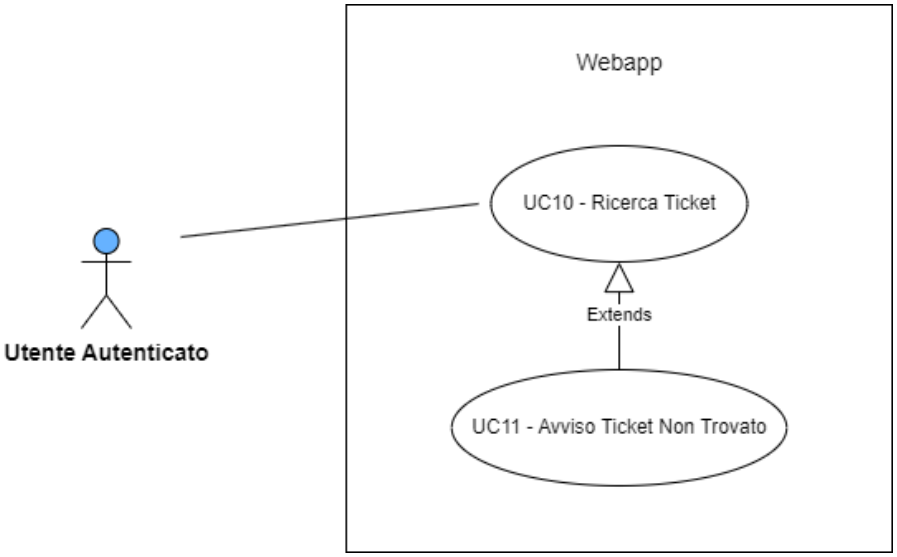


Figura 4.9: UC10 - UC11

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina di ricerca di ticket.
Postcondizioni	L'utente ha ricercato e trovato uno o più ticket.
Scenario principale	L'utente ricerca i ticket
Estensioni	Se la ricerca non dà nessun risultato, si verifica UC11.

Tabella 4.13: UC10 - UC11

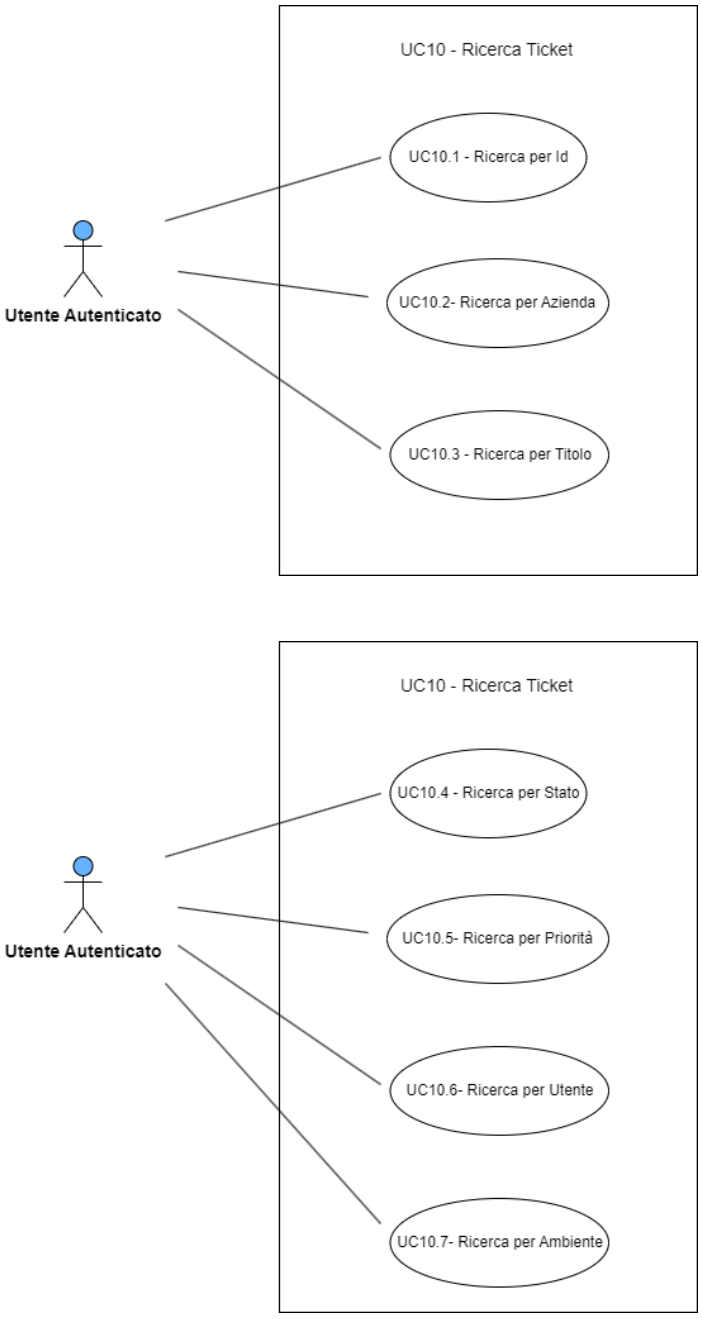


Figura 4.10: Filtri

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina di ricerca di ticket.
Postcondizioni	L'utente ha ricercato attraverso dei filtri.
Scenario principale	L'utente seleziona i filtri con cui effettuare la ricerca.

Tabella 4.14: UC10 - Filtri

UC11 - Ticket Non Trovato

Attore primario	Utente autenticato
Precondizioni	L'utente ha effettuato la ricerca secondo dei filtri
Postcondizioni	Viene visualizzato un avviso perché non è stato trovato nessun ticket.
Scenario principale	La ricerca non è andata a buon fine

Tabella 4.15: UC11

UC12, UC13 - Download Allegato

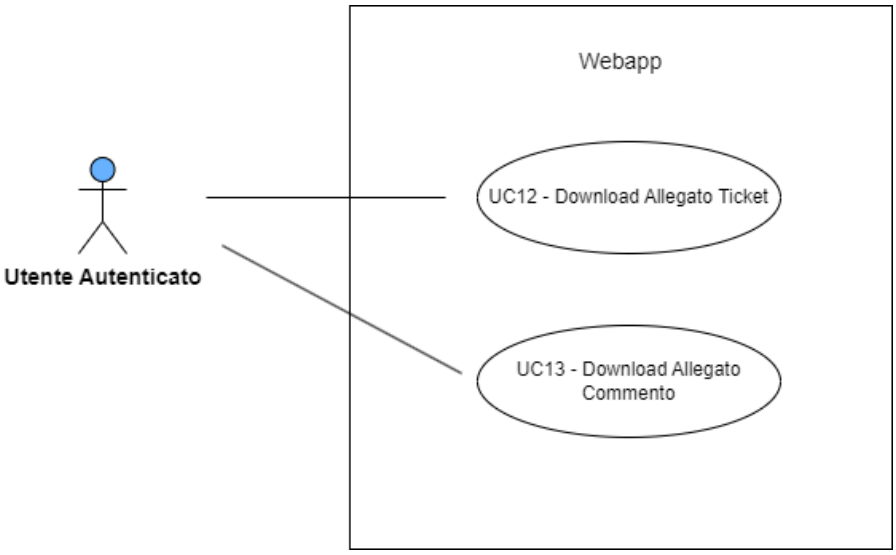


Figura 4.11: UC12 - UC13

UC12 - Download Allegato del Ticket

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina di dettaglio del Ticket
Postcondizioni	Viene scaricato l'allegato collegato al ticket.
Scenario principale	L'utente vuole scaricare l'allegato del ticket

Tabella 4.16: UC12

UC13 - Download Allegato del Commento

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina di dettaglio del Ticket
Postcondizioni	Viene scaricato l'allegato collegato al commento.
Scenario principale	L'utente vuole scaricare l'allegato del commento

Tabella 4.17: UC13

UC14 - Nuovo Commento

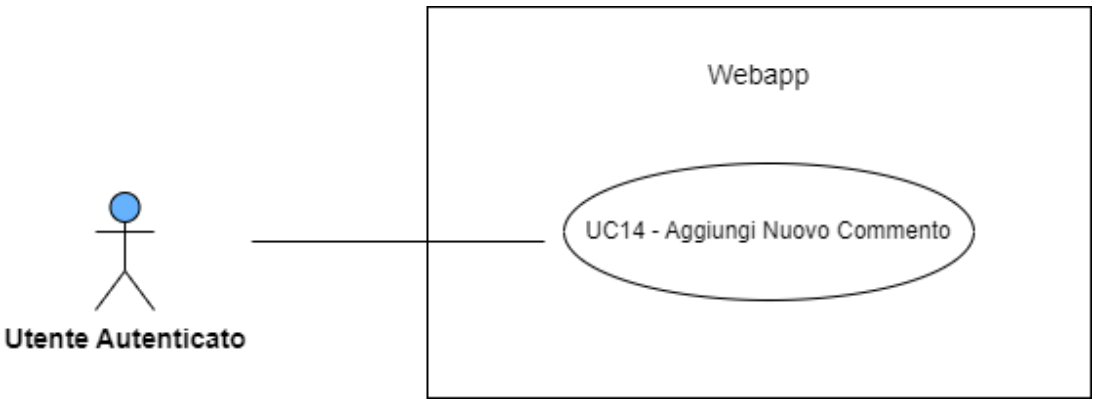


Figura 4.12: UC14

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina di dettaglio del Ticket.
Postcondizioni	L'utente ha pubblicato un commento.
Scenario principale	L'utente vuole aggiungere un commento al ticket che verrà visualizzato nella sezione "Commenti".

Tabella 4.18: UC14

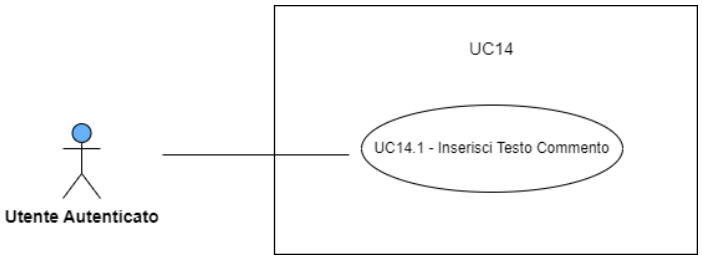


Figura 4.13: UC14.1

Attore primario	Utente autenticato
Precondizioni	L'utente è nella pagina del commento.
Postcondizioni	L'utente ha inserito il testo del commento

Tabella 4.19: UC14.1

UC15, UC16 - Eliminazione

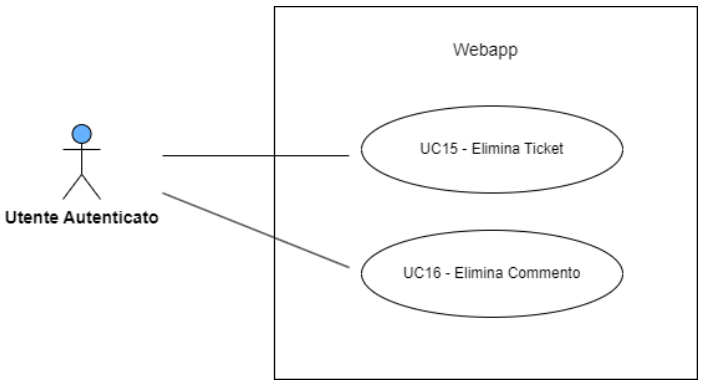


Figura 4.14: UC15 - UC16

UC15

Attore primario	Utente autenticato
Precondizioni	L'utente è nel modulo Ticket
Postcondizioni	L'utente ha eliminato un Ticket.
Scenario principale	L'utente vuole eliminare il ticket selezionato

Tabella 4.20: UC15

UC16

Attore primario	Utente autenticato
Precondizioni	L'utente è nel modulo Ticket
Postcondizioni	L'utente ha eliminato un commento.
Scenario principale	L'utente vuole eliminare un commento dalla pagina di dettaglio di un Ticket

Tabella 4.21: UC16

UC17

Attore primario	Utente autenticato
Precondizioni	L'utente ha provato ha creare un nuovo ticket
Postcondizioni	L'utente non è riuscito a creare il nuovo ticket
Scenario principale	L'utente cerca di creare un nuovo ticket, lasciando vuoti alcuni campi

Tabella 4.22: UC17

4.3 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 in rapporto agli use case.
Il requisito è così descritto:

- **codice identificativo**: ogni codice identificativo è univoco e definito seguendo lo standard di codifica **R[Importanza][Tipologia] [Codice]** il significato delle cui voci è:
  - **Importanza**:

1	Requisito obbligatorio: irrinunciabile per l'azienda e i clienti
2	Requisito desiderabile: non strettamente necessario ma a valore aggiunto riconoscibile.
3	Requisito opzionale: relativamente utile oppure contrattabile più avanti nel progetto.

  - **Tipologia**:

F	Funzionale
P	Prestazionale
Q	Qualitativo
V	Vincolo

  - **Codice**: identificatore univoco del requisito in forma gerarchica.
- **classificazione**: viene riportata l'importanza del requisito per facilitare la lettura;
- **descrizione**;
- **fonte**: origine del requisito.

4.3.1 Requisiti funzionali

Tabella 4.23: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione
R1F1	L'utente ha la possibilità di effetuare 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
R1F5	L'utente amministratore può registrare un nuovo utente specificandone i permessi
R1F6	L'utente loggato, accedendo al modulo ticket, può visualizzare la pagina di Home
R1F7	L'utente loggato, accedendo al modulo ticket, può visualizzare la pagina di Menu
R1F8	L'utente loggato può visualizzare i ticket aperti da lui
R1F9	L'utente loggato può visualizzare i ticket presi in carico
R1F10	L'utente loggato può visualizzare gli ultimi 10 ticket aperti
R1F11	L'utente loggato può creare un nuovo ticket
R1F11.1	L'utente loggato, quando crea un nuovo ticket, deve selezionare l'azienda di riferimento
R1F11.2	L'utente loggato, deve selezionare il progetto su cui aprire il ticket
R1F11.3	L'utente loggato deve inserire il titolo del ticket
R1F11.4	L'utente loggato può inserire una descrizione del ticket
R1F11.5	L'utente loggato deve selezionare l'ambiente su cui aprire il ticket
R1F11.6	L'utente loggato deve selezionare la priorità del ticket

Requisito	Descrizione
R1F11.7	L'utente loggato deve selezionare lo stato del ticket
R1F11.8	L'utente loggato può inserire la data di scadenza entro il quale il ticket deve essere completato
R1F11.9	L'utente loggato può inserire un allegato
R1F12	Il modulo visualizza la pagina di selezione dell' azienda
R1F12.1	Il modulo visualizza solo e soltanto la lista delle aziende a cui è collegato l'utente loggato
R1F13	Il modulo visualizza la pagina di creazione del ticket
R1F13.1	visualizza solo e soltanto i progetti collegati all'azienda selezionata
R1F13.2	Il modulo fornisce 3 ambienti su cui aprire il ticket: SVIL, PROD, TEST
R1F13.3	Il modulo fornisce 4 gradi di priorità (1,2,3,4)
R1F13.4	Il modulo fornisce 2 stati del ticket: APERTO - CHIUSO
R1F13.5	Il modulo fornisce di poter allegare file di tipo: pdf/txt
R1F13.6	Il modulo fornisce la possibilità di personalizzare la descrizione del ticket
R1F13.7	Il modulo fornisce di annullare la creazione di un nuovo ticket
R1F14	L'utente loggato può modificare un ticket
R1F14.1	L'utente loggato, in fase di modifica, deve selezionare l'utente assegnatario
R1F15	L'utente loggato può eliminare un ticket
R1F16	L'utente loggato può visualizzare il dettaglio del ticket
R1F16.1	Il modulo visulizza il titolo del ticket
R1F16.2	Il modulo visualizza l'id del ticket
R1F16.3	Il modulo visualizza l'etichetta stato del ticket
R1F16.4	Il modulo visualizza l'etichetta priorità del ticket

Requisito	Descrizione
R1F16.5	Il modulo visualizza l'etichetta ambiente del ticket
R1F16.6	Il modulo visualizza la descrizione estesa del ticket
R1F16.7	Il modulo visualizza l'azienda del ticket
R1F16.8	Il modulo visualizza l'assegnatario del ticket
R1F16.9	Il modulo visualizza la descrizione del ticket
R1F16.10	Il modulo visualizza l'assegnatario del ticket
R1F17	L'utente loggato può scaricare il file allegato al ticket
R1F18	L'utente loggato può cambiare lo stato del ticket dal dettaglio
R1F19	L'utente può visualizzare la lista dei commenti del ticket
R1F19.1	L'utente loggato può scaricare il file allegato al singolo commento
R1F20	L'utente loggato può creare un nuovo commento per un ticket
R1F21	L'utente loggato può modificare un commento
R1F22	L'utente loggato può eliminare un commento
R1F23	L'utente loggato può allegare un file al commento
R1F24	L'utente loggato può visualizzare la pagina di ricerca
R1F25	L"utente loggato può ricercare i ticket
R1F25.1	L'utente loggato può ricercare i ticket per identificativo
R1F25.2	L'utente loggato può ricercare i ticket per azienda
R1F25.3	L'utente loggato può ricercare i ticket per titolo
R1F25.4	L'utente loggato può ricercare i ticket per stato
R1F25.5	L'utente loggato può ricercare i ticket per priorità
R1F25.6	L'utente loggato può ricercare i ticket per utente assegnato
R1F25.7	L'utente loggato può ricercare i ticket per ambiente

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

4.3.2 Requisiti di vincolo

Tabella 4.24: Tabella del tracciamento dei requisiti di vincolo

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

Tabella 4.25: Tabella del tracciamento dei requisiti di vincolo

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

Capitolo 5

Progettazione e codifica

Il capitolo inizialmente presenta gli strumenti e le tecnologie analizzate e utilizzate per la realizzazione del prodotto. Successivamente si vede l'effettiva creazione delle classi del progetto, affiancate da una struttura preesistente fondamentale per le classi che offre affinché il modulo funzioni e svolga la sua funzione.

5.1 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 funzionalità per la personalizzazione delle pagine. È studiata e conosciuta con il corso di Tecnologie Web.

Bootstrap3/5

Bootstrap è un framework che fornisce delle classi le quali raggruppano uno o più attributi del css, per la creazione di pagine web *responsive*.
Bootstrap è utilizzato nello stile *inline* di *HTML* e le sue classi vengono inserite all'interno del *tag* "class" di HTML di un elemento. In questo modo, specificando la classe di Boostrap che si vuole utilizzare, verrà applicato un certo stile all'elemento selezionato. Si possono concatenare più classi per un certo elemento.
La differenza tra le due versioni di *Boostrap* 3 e 5 è nella gamma di funzionalità che offrono. La versione 5 è la più recente e molte più funzionalità di quelle precedenti, adattandosi alle nuove feature di *HTML5*.
Ad oggi si cerca di migrare dalle versioni più vecchie a quella più recente.

Servlet

Le *servlet* permettono di soddisfare delle *request HTTP* proveniente da web. Ogni servlet viene richiamata e caricata una sola volta e poi resta in memoria per rispondere alle chiamate successive.

JSP

Le *JavaServer Page* rappresentano una tecnologia fondamentale per la realizzazione di pagine web dinamiche. Infatti forniscono dei *tag* speciali con i quali possono essere richiamate delle funzioni specifiche in modo da rendere la pagina dinamica. I file *JSP* sono caratterizzati dell'estensione .jsp e costituiscono le vere e proprie pagine web visualizzate dall'utente in quanto permettono la codifica in *HTML* e *XML*.

JSTL

JavaServer Pages Standard Tag Library è una libreria che estende JSP offrendo nuove funzionalità per applicazioni web in JAVA EE.

Apache Struts

Apache Struts è un *framework oper-source* che supporta lo sviluppo di applicazioni web in Java con il pattern MVC. Infatti *Struts* ha il compito di organizzare le richieste del client e richiamare le funzionalità della logica di business.
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 vengono specificati anche gli ***interceptor*** per le *Action* delle nostre classi. La specifica degli *interceptor* è una fase importante dello sviluppo di un'applicazione web.

JQUERY Taconite

JQUERY Taconite permette di aggiornare DOM multipli utilizzando il risultato di una singola chiamata *AJAX*.
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 non deve preoccuparsi di implementazione ma è *Hibernate* che si occupa del collegamento al database e di eseguire le operazioni *CRUD*, andando a generare query e leggerne il risultato.

Spring

Spring un framework volto ad aiutare lo sviluppo di applicazioni più o meno complesse attraverso la sua architettura modulare. Spring è diviso in cinque livelli e in questo modo si possono escludere le parti non necessarie per l'applicazione.
L'elemento principale di *Spring* è il *Core Container* che ha il compito di creazione e gestione di tutti gli oggetti dell'applicazione, detti anche ***beans***.

Wro4j

Wro4j è uno strumento per l'ottimizzare le risorse web e velocizzare il caricamento delle pagine. Il suo compito è quello di organizzare le risorse, come i file *css* e *js*, raggrupparli e farli scaricare tutti in una sola volta alla pagina web.
Normalmente un browser può scaricare al massimo due risorse contemporaneamente e questo limite porta ad un caricamento della pagina lento in vista di molte risorse da scaricare. *Wro4j* elimina questo problema comprendendo tutte le risorse in un'unica risorsa.

SVNKit

SVNKit è uno *toolkit Open-Source* e permette l'accesso il remoto e in locale a delle *repository* per le applicazioni Java. Funge anche da sistema di versionamento.

Apache Maven

Apache Maven è uno strumento per la gestione delle dipendenze tra un progetto Java e le versioni delle librerie che servono e si occupa anche di effettuare il download di tali risorse.
Le relazioni tra progetto e librerie sono definite in un file *XML* chiamato ***POM***.

Apache Tomcat

Apache Tomcat è un server web che permette l'esecuzione di applicazioni web. Supporta le specifiche di *JSP* e *servlet*.
Esistono diverse versioni per i server *Tomcat* e si può scegliere la versione che offre le funzionalità giuste per la propria applicazione.

DBeaver

È un'applicazione che si occupa di gestire i database. Si possono creare nuovi database, creare tabelle, manipolare i dati, ecc...

5.2 Progettazione

La fase di progettazione è una fase cruciale per la realizzazione del progetto. Prima ancora di iniziare la progettazione del lavoro assegnato è importante analizzare l'ambiente già esistente, capirne il funzionamento e individuare se ci sono elementi che potrebbero servire al nostro scopo.
Una tecnica per una buona progettazione è concentrarsi su un elemento alla volta. Prendere in considerazione tutti gli elementi del progetto potrebbe sembrare più efficace

e veloce, ma così facendo si può perdere il focus della funzione delle classi del prodotto. Quindi l'idea è quella di analizzare e progettare una singola classe, vedere se funziona correttamente e da lì procedere ed andare avanti con lo studio per la realizzazione della prossima classe.

5.2.1 Base di Dati

Per la progettazione della base di dati si è partiti da una struttura già esistente nell'azienda e si sono inseriti i nuovi elementi del progetto. Si è eseguito un *refactoring* di alcune delle tabelle preesistenti per correggerle ed adattarle a quelle nuove, senza però modificarne attributi fondamentali per le altre parti della webapp. Anche qui c'è stato allora una fase di profonda e attenta analisi per avere una base di dati consistente.

Tabelle preesistenti

Il progetto utilizza tabelle preesistenti per il suo scopo. Le tabelle sono:

- **Cliente**: questa tabella rappresenta l'entità cliente e ha tutti gli attributi necessari per definirne lo scopo all'interno della webapp. Il cliente non è la singola persona ma bensì l'azienda a cui poi saranno collegati sia i dipendenti sia i progetti richiesti da questa. Gli altri attributi presenti servono per le altre parti della webapp
- **User**: gli user sono tutte le persone coinvolte nell'azienda, interne ed esterne. Quindi troveremo sia il personale di CWBI sia il personale delle aziende clienti.
- **Progetto**: questa tabella rappresenta un progetto dell'azienda. Avrà tutti gli attributi necessari e sarà collegata ad un cliente. Un progetto può essere collegato a più aziende diverse.
- **ProgettoCliente**: questa tabella rappresenta la relazione tra progetto e cliente. Viene chiamata in causa quando si dovrà scegliere l'azienda e il progetto su cui si vorrà aprire il ticket.
- **ProgettoUser**: questa tabella rappresenta la relazione tra progetto e user. Ogni progetto può avere associato uno o più utenti.

Tabelle introdotte

- **Ticket**: questa tabella rappresenta l'entità ticket con tutti gli attributi che lo caratterizzano. Sarà collegata all'entità ProgettoCliente in quanto il ticket sarà aperto per uno specifico progetto di una specifica azienda.
- **TicketItem**: questa tabella rappresenta i commenti presenti in ogni ticket. Un commento potrà avere un solo ticket di riferimento, cioè quello in cui è stato scritto.

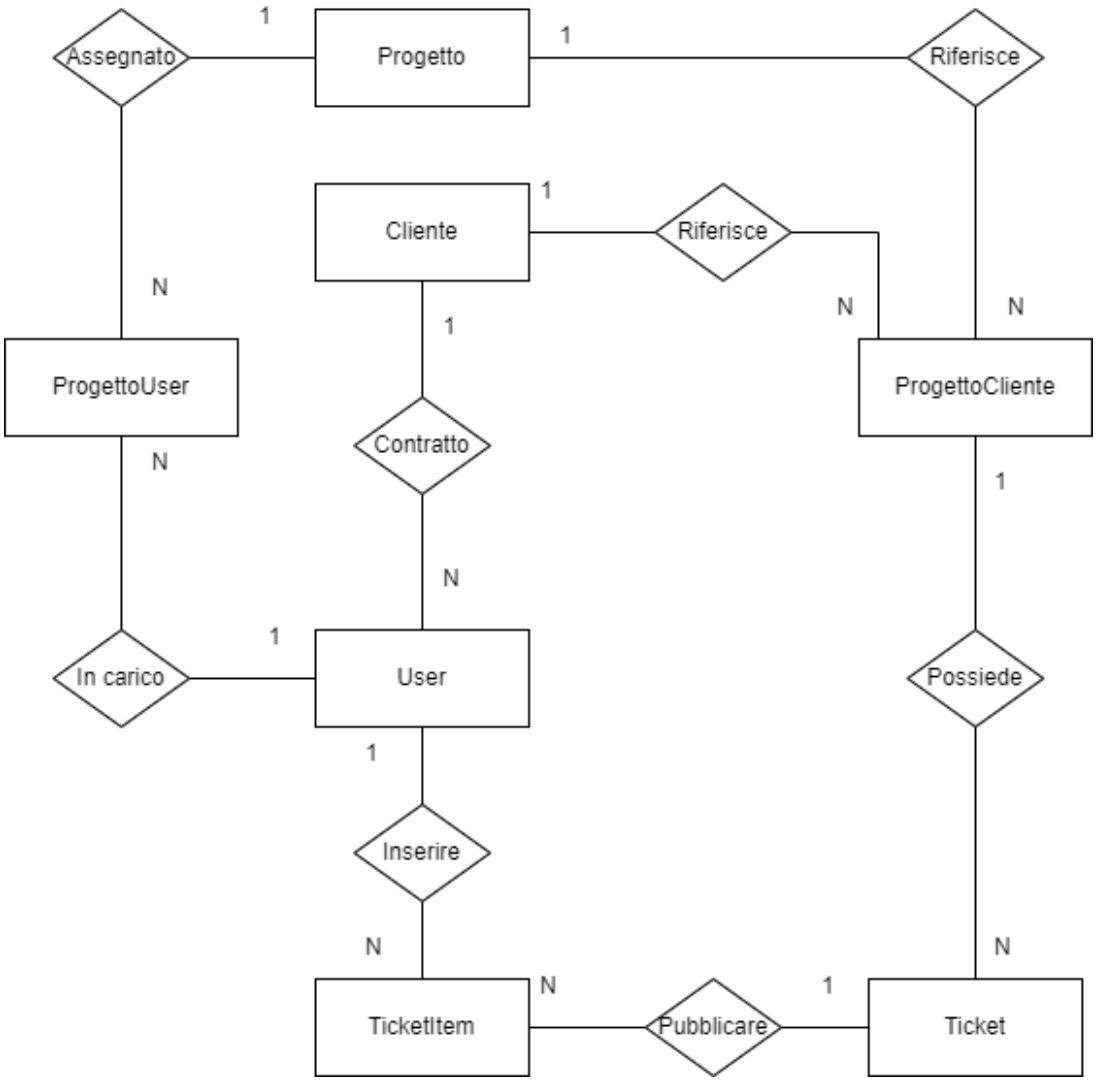


Figura 5.1: Base di Dati - Relazioni delle tabelle del progetto

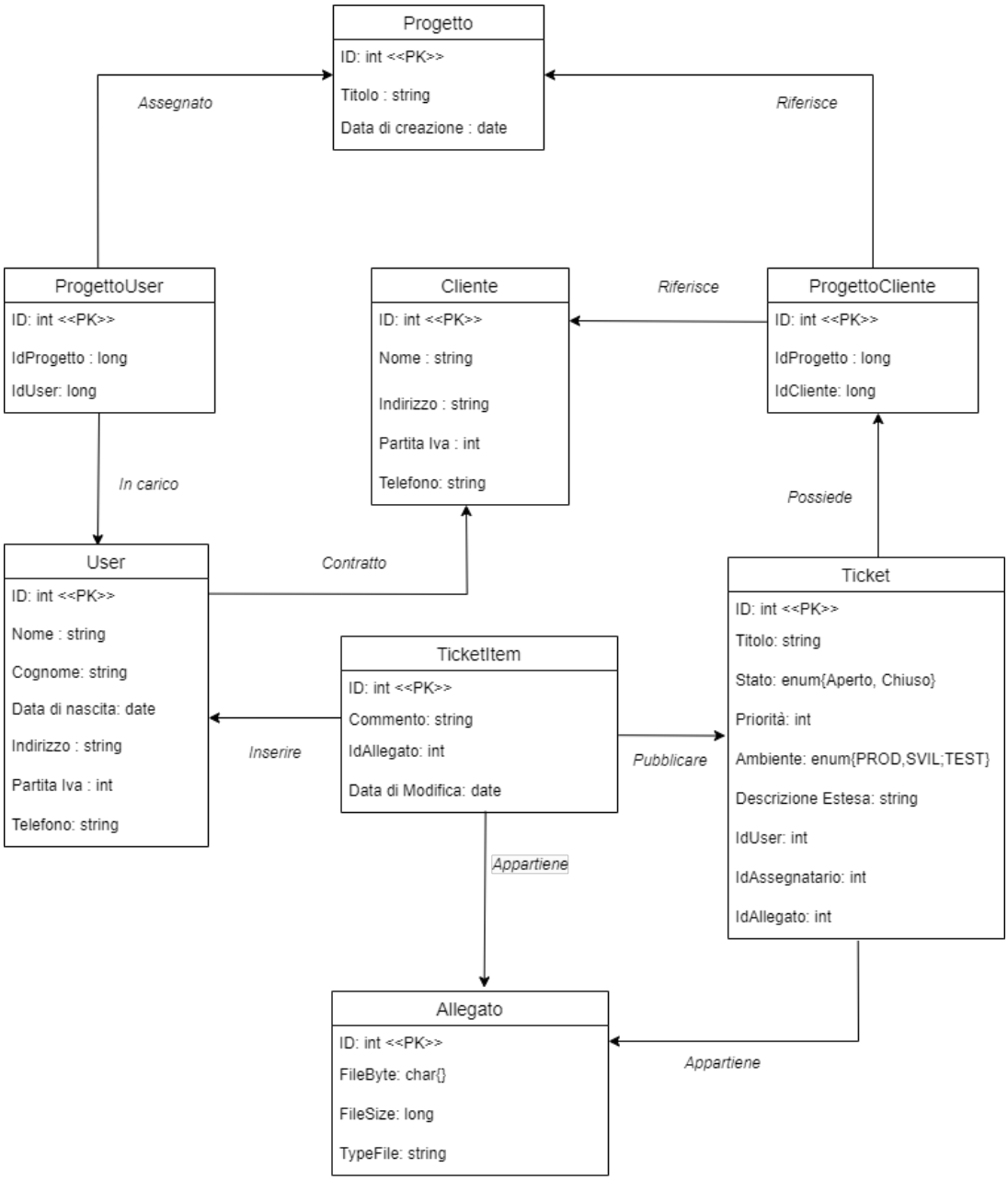


Figura 5.2: Base di Dati - Tabelle del progetto

Analisi delle tabelle

Cliente	
Id	Identificativo univoco di ogni cliente.
Nome	Nome dell'azienda.
Indirizzo	Indirizzo della sede principale dell'azienda.
Partita Iva	Partita Iva dell'azienda
Telefono	Telefono dell'azienda

Tabella 5.1: Tabella Cliente

ProgettoCliente	
Id	Identificativo univoco di ogni ProgettoCliente.
IdProgetto	Id del progetto a cui si riferisce.
IdCliente	Id dell'azienda a cui si riferisce.

Tabella 5.2: Tabella ProgettoCliente

Progetto	
Id	Identificativo univoco di ogni Progetto.
Titolo	Titolo del progetto.
Data di creazione	Data in cui è stato aperto il progetto.

Tabella 5.3: Tabella Progetto

ProgettoUser	
Id	Identificativo univoco di ogni ProgettoUser.
IdProgetto	Id del progetto a cui si riferisce.
IdUser	Id dell'utente a cui si riferisce.

Tabella 5.4: Tabella ProgettoUser

User	
Id	Identificativo univoco di ogni User.
Nome	Nome dell'utente.
Cognome	Cognome dell'utente.
Data di nascita	Data di nascita dell'utente.
indirizzo	Indirizzo dell'utente.
Partita Iva	Partita Iva dell'utente
Telefono	Telefono dell'utente

Tabella 5.5: Tabella User

Ticket	
Id	Identificativo univoco di ogni Ticket.
Titolo	Titolo del ticket.
Stato	Stato del Ticket.
Priorità	Priorità che un ticket ha. Parte da un minimo di 1, quindi poco urgente, ad un massimo di 4, urgente.

Tabella 5.6: Tabella Ticket

Ambiente	Ambiente del Ticket. Un ticket può essere aperto in base all'ambiente in cui si sta testando l'applicazione e si trova il problema. Si hanno tre diversi ambienti: PROD, SVIL, TEST.
Descrizione	Descrizione del Ticket. Utile per approfondire il problema che si è riscontrato.
IdUser	Id dell'utente che ha aperto il ticket.
IdAssegnatario	Id dell'utente a cui è stato assegnato il ticket. Può essere cambiato durante il ciclo di vita del ticket.
IdAllegato	Id dell'allegato caricato al ticket. Può essere cambiato durante il ciclo di vita del ticket.

Tabella 5.7: Tabella Ticket

TicketItem	
Id	Identificativo univoco di ogni Commento.
Commento	Contenuto del commento.
IdAllegato	Id dell'allegato caricato al commento.
Data di Modifica	Data in cui è stato modificato il ticket.

Tabella 5.8: Tabella TicketItem

Allegato	
FileByte	Contenuto del file caricato codificato in un vettore di caratteri (char[])
FileSize	Dimensione del file caricato.
TypeFile	Tipo del file caricato.

Tabella 5.9: Tabella Allegato

5.2.2 Architettura

Lo sviluppo dell'applicazione avviene secondo il *pattern* architetturale **MVC** [g] (*Model - View - Controller*).
Questo *pattern* permette di dividere e rendere modulabile 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 una schema definito.
- **Controller**: ha il compito di gestire la comunicazione tra il modello e la vista ed elaborare gli input dell'utente per poi fornire in output un determinato risultato.

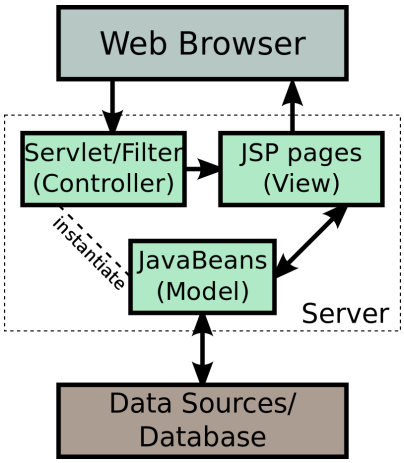


Figura 5.3: Schema MVC

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

- **Manutenzione**: la suddivisione in componenti rende la manutenzione dell'applicazione più semplice, andando a concentrarsi sulla parte interessata.
- **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 un parte dell'applicazione, bisognerebbe eseguire la diagnosi sul complessivo. Mentre la suddivisione in componenti permette di eseguire i test più velocemente prendendo in considerazione la parte su cui si vuole eseguirli.
- **Separazione delle responsabilità**: ogni componente ha un compito ben preciso e non andrà a interessarsi delle parti di codice che non sono sotto la sua responsabilità.

Model

Si parte da un modello preesistente e strutturato secondo gli standard aziendali. Infatti questa parte è divisa in più livelli, ognuno dei quali ha un diverso scopo e diverse funzionalità. In generale, la struttura presente è illustrata come segue:

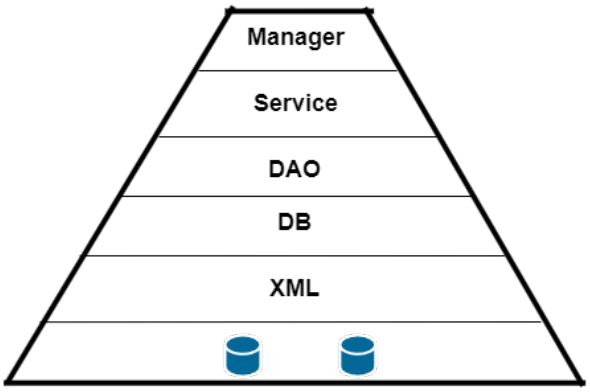


Figura 5.4: Struttura Model CWBI

Come detto in precedenza alcune delle classi già presenti sono utilizzate per supportare le nuove entità introdotte con il progetto. Per la codifica del nostro nuovo Model ci basiamo sulla figura *5.4*, partendo dal basso e andando verso l'alto.

Il primo passo per la creazione di una nuova classe è mappare i suoi attributi all'interno del file xml che conterrà la mappatura e specificherà la struttura della tabella che sarà costruita poi sul database.
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 webapp. Importante specificare che gli attributi inseriti in questa classe saranno uguali agli attributi specificati nella tabella *ticket-a* nel file di mappatura. La classe sarà composta quindi dagli attributi, i costruttori e le funzioni di *get* degli attributi.
Le nomenclatura *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, secondo il pattern ***Dao***: le operazioni *CRUD*.
La classe *TicketDaoHibernate* invece è l'implementazione dell'interfaccia *TicketDao*, che implementa quindi le funzioni presenti.
In realtà, l'esecuzione effettiva della funzione CRUD richiesta, non avviene dentro TicketDaoHibernate ma all'interno di ogni funzione implementata, viene richiamata un'altra funzione che ha lo stesso scopo di quella in cui sta venendo richiamata ma è implementata in un'altra classe *Hibernate*, padre di *TicketDaoHibarnate* che effettuerà le operazioni classiche CRUD.

In generale, *TicketDaoHibarnate* implementa le funzioni dell'interfaccia a cui si riferisce, ma le operazioni CRUD vengono sempre effettuate dall'*Hibernate* padre.

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, come ad esempio la funzione di ricerca.
TicketServiceImpl invece è l'implementazione delle funzioni di TicketService. All'interno di ogni funzione, il dato viene manipolato e alla fine, verrà richiamata la funzione della classe TicketDao specifica per il contesto. Si può dire che all'interno dell'implementazione di una funzione *service* il dato viene manipolato e personalizzato affinché, ad esempio la funzione di ricerca, ci fornisca il risultato desiderato.

È importante notare che la funzione di ricerca non è una funzione *CRUD* e viene implementata nel service. Infatti nella sua implementazione, questa funzione richiamerà la funzione di *TicketDaoHibernate* che effettuerà la **lettura** del dato corrispondete ai parametri di ricerca.

Allora tutte le funzioni all'interno delle applicazioni si riducono sempre a delle semplice 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* (View) e la parte di *back-end* (Model), oltre a gestire anche gli input degli utenti.

Il *controller* ha il compito di *istanziare* le classi del Model, richiamarne le funzioni per avere un risultato e inviarlo poi al *front-end* in modo da essere visualizzato. All'interno dell'applicazione le classi che fungono da *controller* sono marcate dall'annotazione *@Controller* di Spring.
Un'altra annotazione Spring presente è *@Autowired* che serve per indicare le dipendenze dei *bean* (classi. Infatti all'interno dei controller troveremo degli oggetti *Service* che saranno utilizzati per le operazioni sul Model. Quindi quando verrà istanziata il controller, anche gli oggetti all'interno di esso verranno creati.

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*. Quindi quando l'utente compilerà i campi, andrà a popolare gli attributi di questa classe.
  Le caratteristiche di TicketForm saranno adeguate alla controparte dell'oggetto Ticket nel modello. Infatti il *controller* che si occupa della creazione e della modifica, dovrà costruire l'oggetto Ticket partendo dall'oggetto TicketForm appena popolato.

- *TicketSearchForm*: questa classe identifica i campi di input presenti alla ricerca di un ticket. L'utente per effettuare la ricerca di un ticket potrà scegliere o compilare dei filtri , che saranno rappresentati da *TicketSearchForm*.
  Le caratteristiche di tale classe sono redatte in base ai tipi di filtri che si vogliono fornire all'utente e devono essere adeguati per le proprietà dell'oggetto su cui si sta facendo la ricerca.

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 pagine della webapp che non hanno form al loro interno. Il dettaglio del Ticket sarà gestito da *TicketAction* in quanto non possiede nessun tipo di campo da compilare.
  Si trovano diverse funzioni, come ad esempio la funzione di caricamento della pagina di dettaglio di un Ticket. In questa funzione viene utilizzato l'oggetto *TicketService* per richiamare la funzione di ricerca per id e trovare l'oggetto *Ticket* corrispondente e stampare i dati a schermo.

- *TicketFormAction*: questo *controller* si occupa di gestire le *Action* che riguardano la pagina di creazione e modifica di un *Ticket* attraverso la manipolazione dell'oggetto *TicketForm*. Prendiamo come esempio la creazione di un nuovo Ticket.
  Quando si entra nella pagina di creazione di un ticket, i campi dell'oggetto *TicketForm* sono inizializzati vuoti dall'Action *input* e devono essere compilati dall'utente. Al salvataggio, viene richiamata un'altra *Action* che si occupa di prendere i valori presenti nell'oggetto *TicketForm* e creare un nuovo oggetto *Ticket* con i dati prelevati.
  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 *TicketSearchForm* per le proprie funzionalità. Quando si entra nella pagina di ricerca, i campi dell'oggetto *TicketSearchForm* che rappresentano i filtri di ricerca, vengono lasciati vuoti e sarà l'utente poi riempirli. Quando si effettua la ricerca, viene richiamata la *Action* di *search*, dove si costruisce un nuovo oggetto *Ticket* a seconda delle caratteristiche (filtri) dell'oggetto *TicketSearchForm*.
  Viene quindi utilizzato l'oggetto *TicketService* per richiamare la funzione di ricerca che prende in input un oggetto *Ticket* e confronta quale dei ticket presenti nel database ha i valori uguali a quelli del ticket passato. Vengono così trovati tutti i ticket corrispondenti ai filtri inseriti.

View

L'ultimo componente dell'architettura è la ***View*** che ha il compito di visualizzare i dati secondo una logica e fornire la possibilità all'utente di interagire con il modello. Le pagine che compongono la webapp sono file *jsp* che permettono di scrivere codice con standard HTML o XML, ma anche di integrare le funzionalità di Java rendendo i contenuti dinamici.

La ***View*** è supportata anche da diversi *framework* come *Bootstrap* che fornisce delle classi per personalizzare il contenuto della pagina andando a codificare tali classi direttamente nel attributo "class" dei tag di HTML.
Alle pagine jsp è affiancata un'estensione detta *JSTL* che mette a disposizione dei tag per la visualizzazione dei dati in modo dinamico.
Per l'interazione tra modello, controller e view entra in gioco un'ulteriore *framework*, senza il quale, non sarebbe possibile il funzionamento della webapp cos' com'è stata pensata e codificata da CWBI: ***Struts***.

L'utente, nell'utilizzo della webapp, interagisce con gli elementi messi a disposizione dalla *View* e richiama delle specifiche *Action* di controller specifici. Questa interazione è possibile grazie a *Struts* che permette di associare un file jsp ad una Action di un controller, andando a configurare il file *struts.xml*.

5.3 Design Pattern

I ***Design Pattern*** sono soluzioni generali utilizzate per risolvere problemi ricorrenti durante lo sviluppo di un'applicazione. Esistono diversi *design pattern* ed ognuno di loro ha uno scopo preciso durante la codifica del prodotto.
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 presenti nel progetto in quanto basato su una struttura ben definita e solida.

Dependency Injection

La *Dependency injection* è una tecnica che si occupa di separare la creazione di un oggetto dal suo effettivo utilizzo. Quindi quando un oggetto vuole utilizzare un servizio/oggetto, non deve preoccuparsi di come questo servizio/oggetto è composto o creato in quanto li verrà *iniettato* dall'esterno. Quindi le dipendenze di un oggetto con i componenti o i servizi che lo compongono sono risolte e iniettate da una classe chiamata ***Injectors***.
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 dei modi per applicare la *dependency injection*.
Normalmente il flusso di un'applicazione è determinato dagli oggetti e quindi dal codice che la compongono. Con *IoC* il controllo del flusso è affidato ad un *framework* che si occuperà degli oggetti e delle loro dipendenze.
Un esempio di framework che applica *IoC* è *Spring* che introduce delle annotazioni come: *@Component*, *@Service*, *@Repository* o *@Controller*.

Decorator

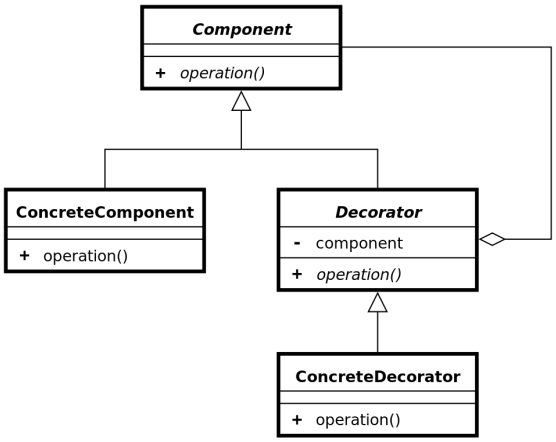


Figura 5.5: Pattern decorator

Il pattern ***Decorator*** è un pattern strutturale che permette di introdurre nuove funzionalità e comportamenti ad un oggetto senza cambiarne la struttura. L'introduzione di questi nuovi elementi viene effettuata a *run-time*. Come si vede dalla *figura 5.5* gli elementi che compongono il *decorator* sono:

- **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 *ConcreteComponent*.

Data Access Object

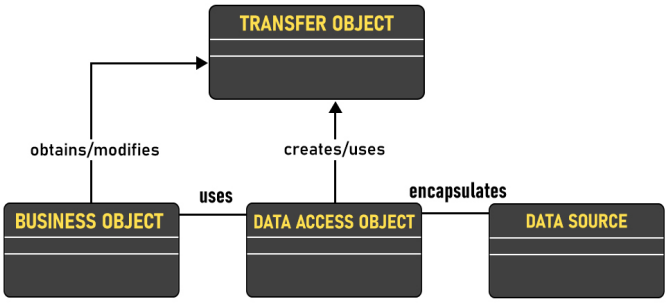


Figura 5.6: Pattern DAO

Il pattern ***Data Access Object***, detto anche ***Dao***, è un pattern architetturale che permette di dividere il livello di business dell'applicazione dalla fonte da cui arrivano i dati, per esempio un database.
Mette a disposizione un'interfaccia che mappa le operazione sui dati alle chiamate per il database. In generale, facilita l'utilizzo delle funzioni *CRUD* separando i bisogni dell'applicazione dal come questi bisogni dovranno essere soddisfatti. In questo modo il livello business e il database evolveranno separatamente senza conoscere i dettagli l'uno dell'altro.

5.4 Codifica

Capitolo 6

Verifica e validazione

Capitolo 7

Conclusioni

7.1 Consuntivo finale

7.2 Raggiungimento degli obiettivi

7.3 Conoscenze acquisite

7.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia