

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 da parte del laureando Fabio Pantaleo presso l'azienda CWBI^[g].

Lo scopo principale del progetto è stata l'analisi e l'implementazione di uno dei rami del CRM^[g] : il ticketing^[g].

Il primo passo per lo sviluppo del modulo web relativo al ticketing consisteva nell'analisi del problema, è stata poi seguita dalla raccolta dei requisiti primari, al fine di elaborare i casi d'uso per l'applicazione. Questa fase iniziale ha avuto grande importanza per il ciclo di vita del prodotto, poiché rappresentava la base di partenza per la costruzione del modello di dati.

Dopo la conclusione di questa prima fase di analisi, il mio lavoro si è concentrato sul mappare le tabelle principali nel database e implementare le classi individuate secondo le indicazioni dei requisiti.

Il passo successivo è stato definire e codificare le funzioni del modulo con cui l'utente si sarebbe interfacciato, realizzando anche le componenti front-end del modulo, in modo da seguire lo standard aziendale.

Il mio periodo di tirocinio è terminato con l'esecuzione dei test volti a verificare la corretta esecuzione delle feature da me introdotte, nonché a garantire il rispetto dei requisiti.

Indice

1	Introduzione	1
1.1	L'azienda	1
1.2	Lo stage	1
1.2.1	Bisogni dell'azienda	1
1.2.2	Il progetto	2
1.3	Tecnologie utilizzate	2
1.4	Organizzazione del testo	3
1.5	Struttura	3
2	Descrizione dello stage	4
2.1	Sistema attuale	4
2.2	Introduzione al progetto	5
2.3	Obiettivi	5
2.4	Pianificazione	5
3	Analisi dei requisiti	7
3.1	Ciclo di vita di un Ticket	7
3.2	Casi d'uso	7
3.2.1	Attori	8
3.2.2	Elenco	9
3.3	Tracciamento dei requisiti	22
3.3.1	Requisiti funzionali	23
3.3.2	Requisiti di vincolo	26
4	Progettazione e codifica	28
4.1	Tecnologie e strumenti	28
4.2	Progettazione	30
4.2.1	Base di Dati	31
4.2.2	Architettura	37
4.3	Design Pattern	41
5	Verifica e validazione	44
5.1	Processo di Verifica	44
5.1.1	Debugging	44
5.2	Processo di Validazione	45
6	Prodotto finale	46
6.1	Pagina iniziale	46
6.1.1	Home Ticket	46

6.1.2	Menu Ticket	47
6.2	Nuovo Ticket	48
6.3	Ricerca Ticket	49
6.4	Dettaglio Ticket	50
6.5	Commento Ticket	52
7	Conclusioni	53
7.1	Consuntivo finale	53
7.2	Raggiungimento degli obiettivi	53
7.3	Valutazione degli strumenti utilizzati	54
7.4	Miglioramenti e future estensioni	54
	Glossario	56
	Bibliografia	58

Elenco delle figure

1.1	CWBI	1
3.1	Gerarchia degli attori	8
3.2	UC01	9
3.3	UC03	10
3.4	UC04	11
3.5	UC05 - UC06 - UC07 - UC08	11
3.6	UC05.1 - UC05.2	13
3.7	UC05.1 - Dettaglio	14
3.8	UC09	15
3.9	UC10 - UC11	16
3.10	Filtri	17
3.11	UC12 - UC13	18
3.12	UC14	19
3.13	UC14.1	20
3.14	UC15 - UC16	20
4.1	Base di Dati - Relazioni delle tabelle del progetto	32
4.2	Base di Dati - Tabelle del progetto	33
4.3	Schema MVC	37
4.4	Struttura Model CWBI	38
4.5	Pattern decorator	42
4.6	Pattern DAO	43
5.1	IDE Eclipse - Debug	45
6.1	Home Ticket	47
6.2	Menu Ticket	47
6.3	Nuovo Ticket - Step 1	48
6.4	Nuovo Ticket - Step 2	49
6.5	Ricerca Ticket	50
6.6	Parte iniziale Dettaglio Ticket	51
6.7	Parte Centrale Dettaglio Ticket	51
6.8	Commenti Dettaglio Ticket	51
6.9	Commenti Dettaglio Ticket	52

Elenco delle tabelle

2.1	Tabella della pianificazione del lavoro	6
3.1	UC01	9
3.2	UC02	9
3.3	UC02	10
3.4	UC03	10
3.5	UC04	11
3.6	UC06	12
3.7	UC07	12
3.8	UC08	12
3.9	UC05.1	13
3.10	UC05.2	13
3.11	UC05.1 - Dettaglio	15
3.12	UC09	15
3.13	UC10 - UC11	16
3.14	UC10 - Filtri	18
3.15	UC11	18
3.16	UC12	19
3.17	UC13	19
3.18	UC14	20
3.19	UC14.1	20
3.20	UC15	21
3.21	UC16	21
3.22	UC17	21
3.23	Tabella del tracciamento dei requisiti funzionali	23
3.24	Tabella del tracciamento dei requisiti funzionali	26
3.25	Tabella del tracciamento dei requisiti di vincolo	27
4.1	Tabella Cliente	34
4.2	Tabella ProgettoCliente	34
4.3	Tabella Progetto	34
4.4	Tabella ProgettoUser	35
4.5	Tabella User	35
4.6	Tabella Ticket	35
4.7	Tabella Ticket	36
4.8	Tabella TicketItem	36

4.9 Tabella Allegato 36

Capitolo 1

Introduzione

1.1 L'azienda

CWBI è 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 Lo stage

1.2.1 Bisogni dell'azienda

L'azienda ha sviluppato un'applicazione web per la gestione del personale e della clientela. La webapp è divisa in diversi moduli e solo alcuni di questi sono stati integrati e perfezionati in modo da poter essere utilizzati attivamente dal personale. Gli altri moduli presenti sono stati introdotti nell'applicazione ma non sviluppati in quanto non essenziali nel breve periodo.

Lo scopo dello stage riguardava l'introduzione di un nuovo modulo: ***Ticket***. Questa nuova feature è stata richiesta dall'azienda per gestire le segnalazioni dei propri

clienti. Infatti, prima dello sviluppo del modulo *Ticket*, l'azienda si interfacciava con le problematiche riscontrate dagli utenti utilizzando degli strumenti sì utili, ma non adatti allo scopo. Ad esempio, per tener traccia di una segnalazione, veniva utilizzato un documento Word, che non forniva dettagli utili per capire la vera natura del problema; inoltre i file potevano essere persi o eliminati da un momento all'altro.

CWBI ha quindi richiesto lo sviluppo del nuovo modulo per gestire al meglio le interazioni con i propri clienti, offrendo a quest'ultimi la possibilità di creare un nuovo ticket in qualsiasi momento.

1.2.2 Il progetto

Il progetto consisteva nella realizzazione del modulo *Ticket*, integrandone tutte le funzionalità richieste dal tutor.

Prima di iniziare la codifica delle classi del progetto, c'è stata una fase di studio dell'architettura aziendale, in modo da comprendere come le entità presenti sono state introdotte e come interagivano tra di loro. Inoltre l'azienda faceva utilizzo di diversi framework per la realizzazione di applicazioni, quindi è stato fondamentale studiarne il funzionamento per poi utilizzarli durante l'implementazione del nuovo modulo *Ticket*. Una volta conclusa questa fase si è svolta l'analisi dei requisiti del progetto per individuare tutte le funzionalità che il modulo doveva mettere a disposizione. Contemporaneamente sono stati identificati gli attributi dell'oggetto *Ticket* e le relative feature.

In una fase successiva, si è proceduto con la mappatura sul database delle nuove entità introdotte e sono state sviluppate le classi necessarie al modulo *Ticket*, con particolare attenzione all'integrazione più ottimale possibile con le altre classi. Il *refactoring* dei modelli già presenti è stata un'attività importante dell'implementazione del nuovo modulo. Questi modelli infatti hanno subito delle modifiche, sia a livello di funzionalità sia a livello di parametri, dato che risultavano incompleti e non compatibili con l'integrazione dell'entità *Ticket*.

Una volta conclusa la codifica del back-end è stato sviluppato anche il front-end. In questo scenario è stato utile osservare la grafica degli altri moduli, già ampiamente sviluppati dal personale CWBI, per rispettare appunto gli standard grafici dell'azienda. Solamente alcune pagine della webpp sono state oggetto di una revisione del layout al fine di garantire una visualizzazione ottimale delle specifiche dei ticket, le quali richiedevano una struttura differente rispetto a quella di solito adottata.

1.3 Tecnologie utilizzate

Nello sviluppo dei propri prodotti, CWBI si occupa sia della parte di *back-end*^[9] sia della parte di *front-end*^[9]. Per la prima, è utilizzato Java^[8] come linguaggio di programmazione, supportato dai vari *framework*^[9]; mentre per la parte destinata alla vista del cliente, sono utilizzati:

- HTML5^[8];
- CSS^[8];
- Bootstrap3/5^[8];
- JSP^[8].

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].

1.4 Organizzazione del testo

Riguardo la stesura del testo 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*.

1.5 Struttura

Il testo sarà composto dai seguenti capitoli:

- Introduzione
- Descrizione dello stage
- Analisi dei requisiti
- Progettazione e codifica
- Verifica e validazione
- Prodotto finale
- Conclusioni

Capitolo 2

Descrizione dello stage

2.1 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 diverse 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.

2.2 Introduzione al progetto

Visti i bisogni dell'azienda, l'obiettivo del modulo **Ticket** era quello di offrire un portale su cui gli utenti registrati potevano aprire, prendere in carico, assegnare ed eliminare un ticket.

Il modulo è stato pensato per i dipendenti interni di CWBI, ma voleva offrire anche ai clienti la possibilità di segnalare in modo facile e veloce un qualsiasi tipo di problema sulle applicazioni utilizzate. Anche per CWBI sarebbe stato agevole identificare il mittente e la data di invio di una segnalazione, al fine di assegnare il compito a un dipendente per risolvere il problema.

Inoltre per rendere più interattivo il gestionale ed avere un riscontro su quali operazioni sono state effettuate sul ticket, ogni utente poteva commentare, in modo da far capire al personale addetto a prendere in carico il ticket, a quale fase della soluzione si era arrivati.

Lavorando su un'applicazione già esistente, un'operazione cruciale che il progetto ha previsto sono state le attività di refactoring di moduli preesistenti per adattarli al nuovo modulo *Ticket*.

2.3 Obiettivi

Lo stage ha definito delle tappe fondamentali da raggiungere, sia per quanto riguarda lo sviluppo di un prodotto, ma anche e soprattutto per la formazione della persona. Questa infatti rappresentava uno dei traguardi principali che l'azienda, il tutor aziendale e lo stagista volevano raggiungere. Gli obiettivi quindi erano:

- la formazione del tirocinante affinché potesse 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 potrà aiutarlo ad affrontare i problemi futuri attraverso una *way of working* concreta;
- lo sviluppo del modulo Ticket per supportare e facilitare i rapporti tra azienda e cliente.

2.4 Pianificazione

Il lavoro si è svolto nelle 300 ore obbligatorie per il tirocinio formativo e si è suddiviso 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 in:

Durata in ore	Attività
40	Formazione iniziale e introduzione tecnologie utilizzata JAVA/-JEE lato server
40	Formazione soluzione <i>baseapp</i> ^[9] con apprendimento <i>framework</i> di lavoro
68	Analisi e raccolta requisiti progetto marketing
122	Sviluppo soluzione (Realizzazione soluzione software in java <i>back-end</i> e sviluppo <i>front-end</i>)
15	Test e supporto UAT
15	Documentazione progetto

Tabella 2.1: Tabella della pianificazione del lavoro

Capitolo 3

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.

3.1 Ciclo di vita di un Ticket

Lo studio del ciclo di vita di un ticket è stato essenziale per capire le interazioni che questo oggetto avrebbe dovuto avere con le altre classi della webapp e per comprendere gli elementi fondamentali che avrebbero composto la nostra entità.

Un ticket può essere in stato di:

- APERTO
- CHIUSO

Lo stato APERTO era assegnato al ticket la prima volta che veniva creato e rimaneva tale fino alla conclusione delle attività previste. Lo stato CHIUSO invece era assegnato quando la persona incaricata del ticket aveva concluso tutte le operazioni, veniva così cambiato lo stato da aperto a chiuso. Quando un ticket veniva chiuso poteva essere aperto nuovamente qualora si riteneva che le attività non erano state svolte completamente oppure se il cliente non rimaneva soddisfatto delle soluzioni applicate. È importante specificare che un ticket poteva essere preso in carico da più tecnici durante il suo ciclo di vita. Infatti le operazioni assegnate dal cliente potevano richiedere diversi ruoli; il ticket veniva chiuso da chi lo avrebbe preso in carico per ultimo.

3.2 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 UML^[g] 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 modulo dedicato al solo ticketing, 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.

3.2.1 Attori

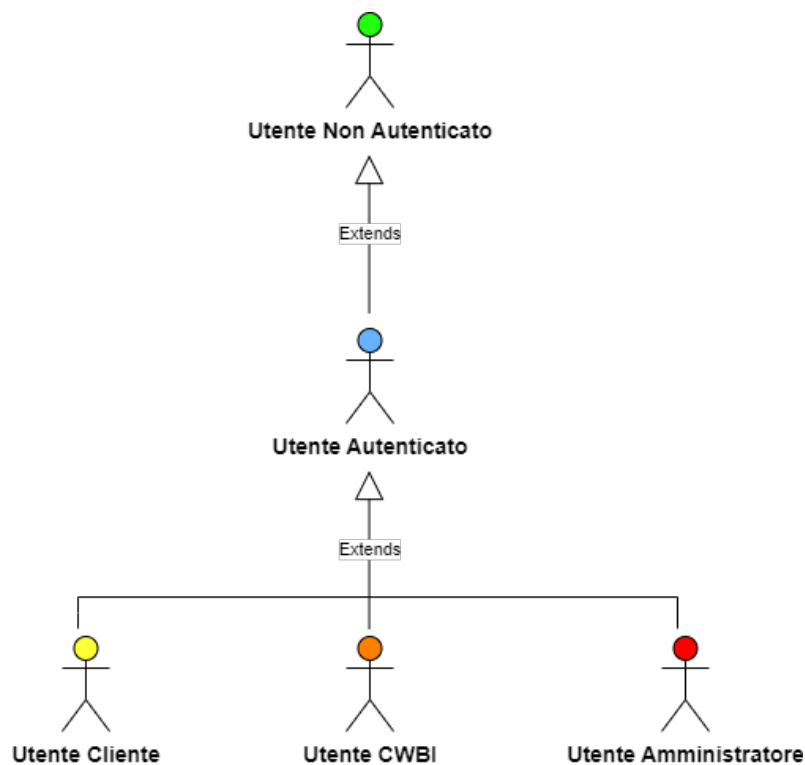


Figura 3.1: Gerarchia degli attori

- **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 che hanno la funzione di amministrare la webapp e i contenuti.

3.2.2 Elenco

UC01 - Autenticazione

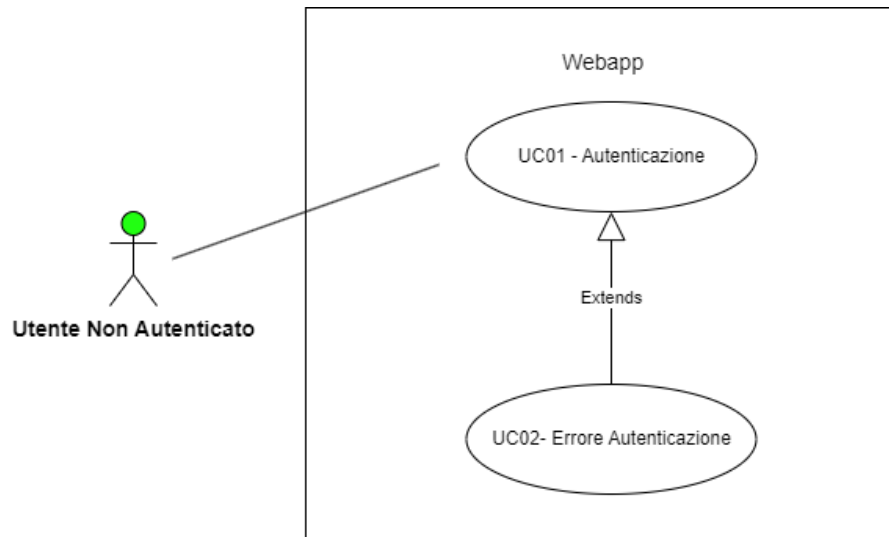


Figura 3.2: UC01

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 3.1: UC01

UC02 - Errore Autenticazione

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

Tabella 3.2: 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 3.3: UC02

UC03 - Registrazione Nuovo Utente

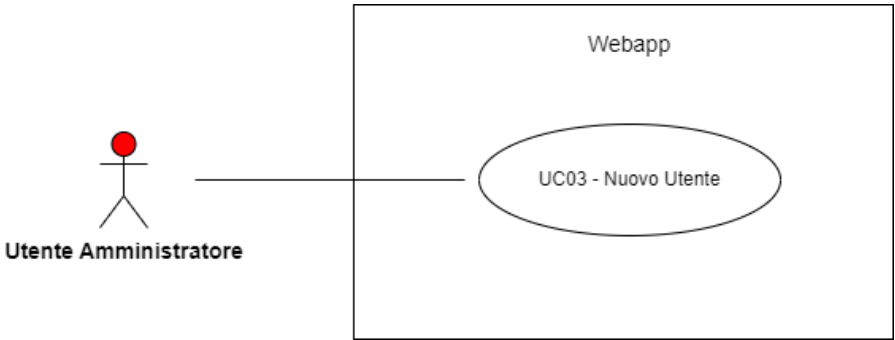
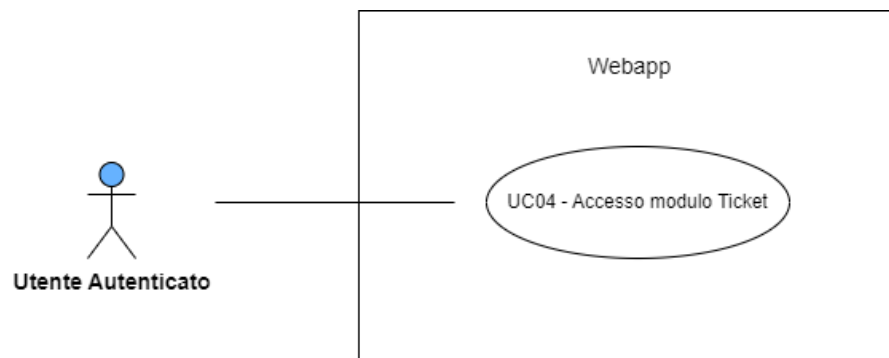


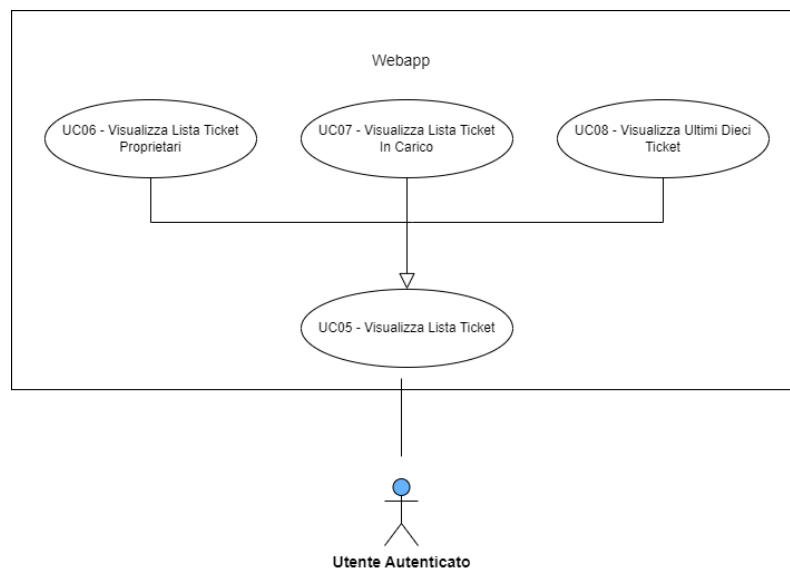
Figura 3.3: UC03

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 3.4: UC03

UC04 - Accesso Modulo Ticket**Figura 3.4:** 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 3.5: UC04**UC05 - Visualizza Lista Ticket****Figura 3.5:** 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 3.6: 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 3.7: 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 3.8: UC08

UC05.1, UC05.2

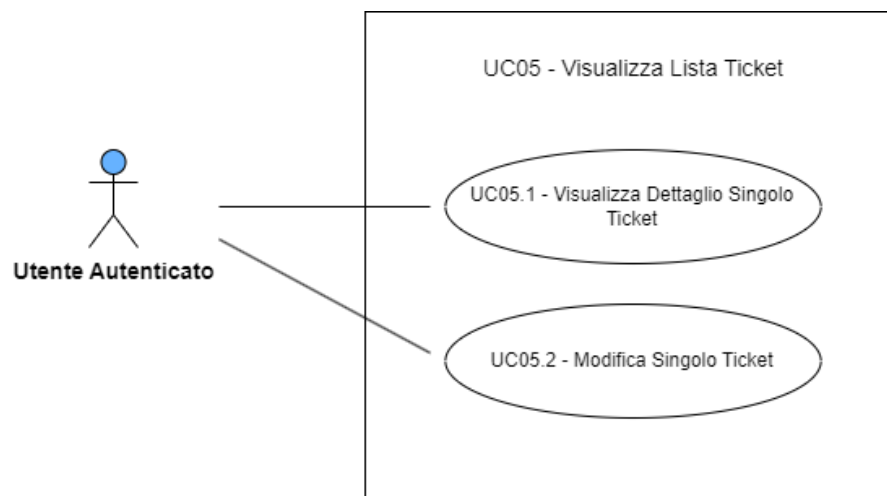


Figura 3.6: UC05.1 - UC05.2

UC05.1

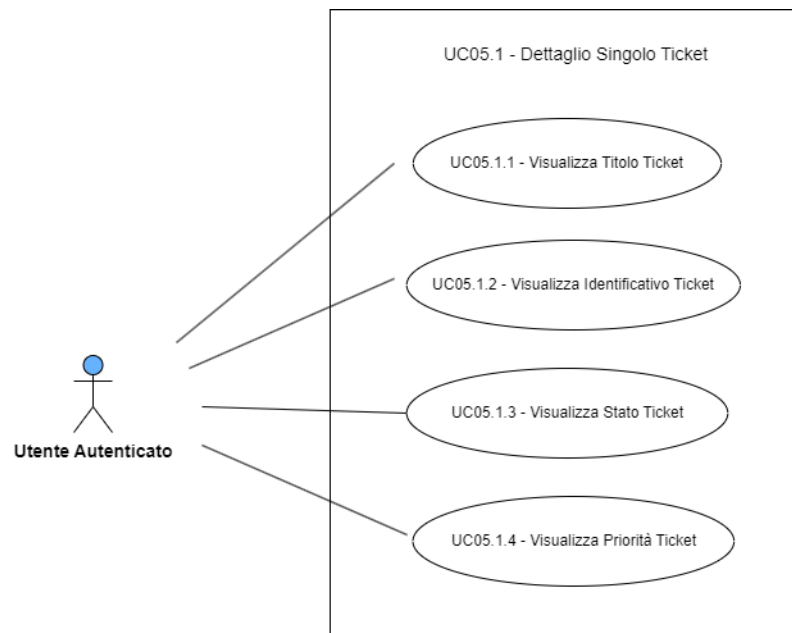
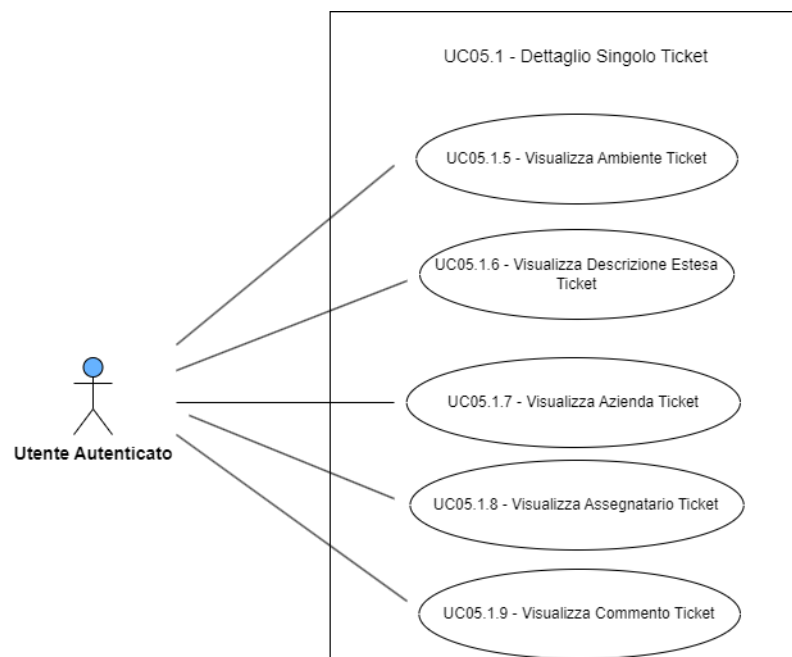
Attore primario	Utente autenticato
Precondizioni	L'utente ha visualizzato la lista dei ticket
Postcondizioni	L'utente visualizza il dettaglio del ticket
Scenario principale	L'utente seleziona il bottone "Dettaglio" per visualizzare i dettagli del ticket

Tabella 3.9: 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 3.10: UC05.2

UC05.1 - Dettaglio**Figura 3.7:** 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 3.11: UC05.1 - Dettaglio

UC09 - Apertura Nuovo Ticket

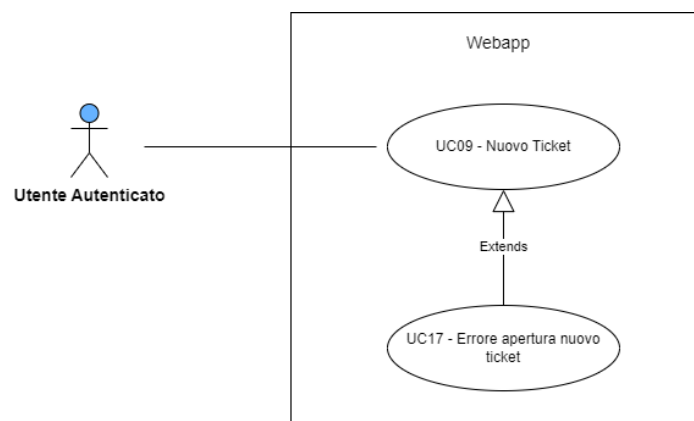
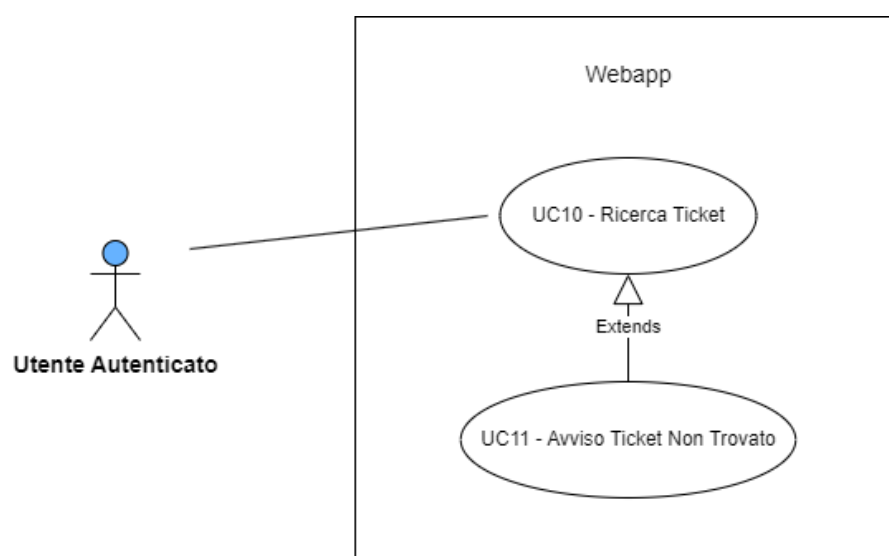


Figura 3.8: UC09

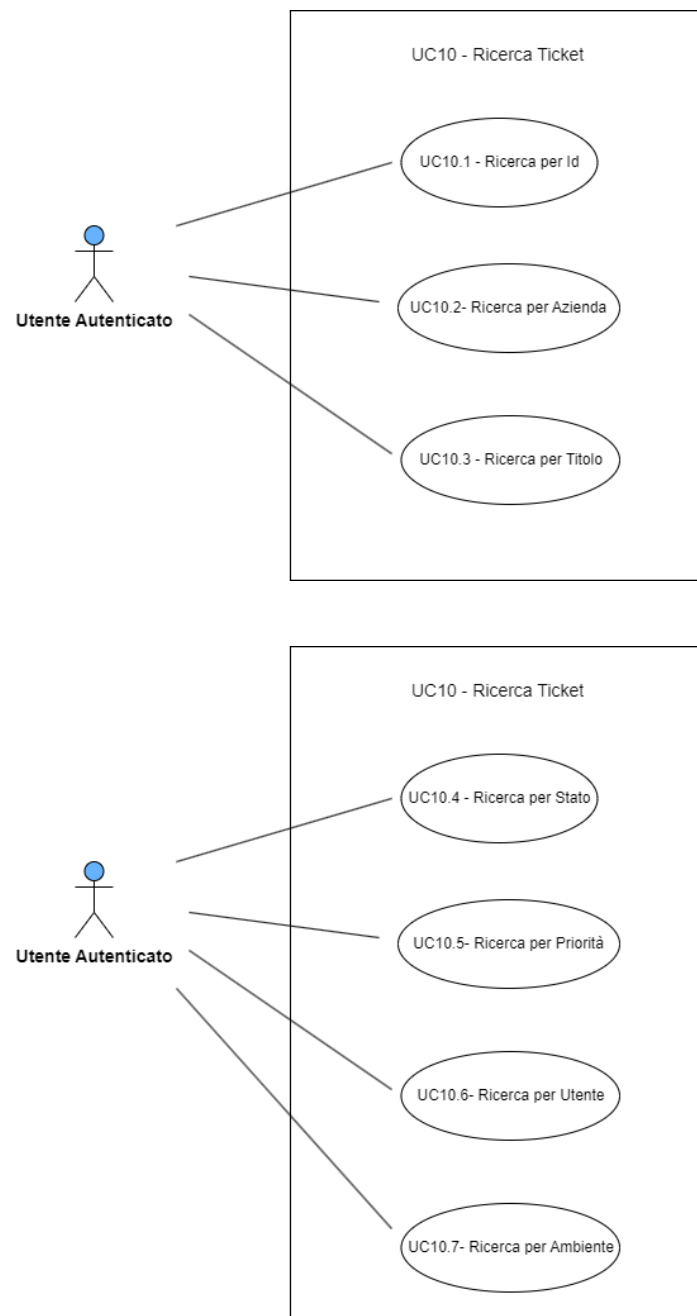
Attore primario	Utente autenticato
Precondizioni	L'utente crea un nuovo ticket compilando i campi: azienda, progetto, titolo, descrizione, data di scadenza, ambiente, priorità, stato, allegato.
Postcondizioni	L'utente apre un nuovo ticket, compilando tutti i campi
Scenario principale	L'utente, accedendo 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 3.12: UC09

UC10 - Ricerca Ticket**Figura 3.9:** 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.

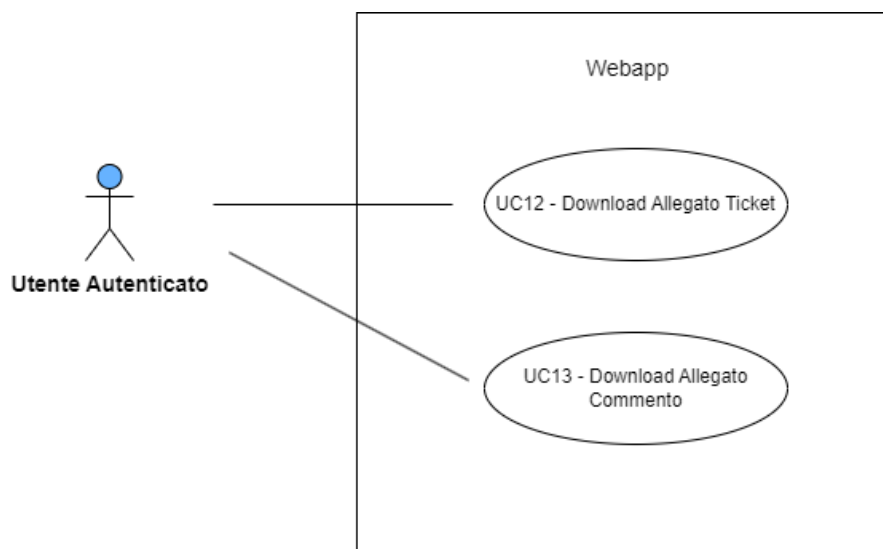
Tabella 3.13: UC10 - UC11

**Figura 3.10:** 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 3.14: 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

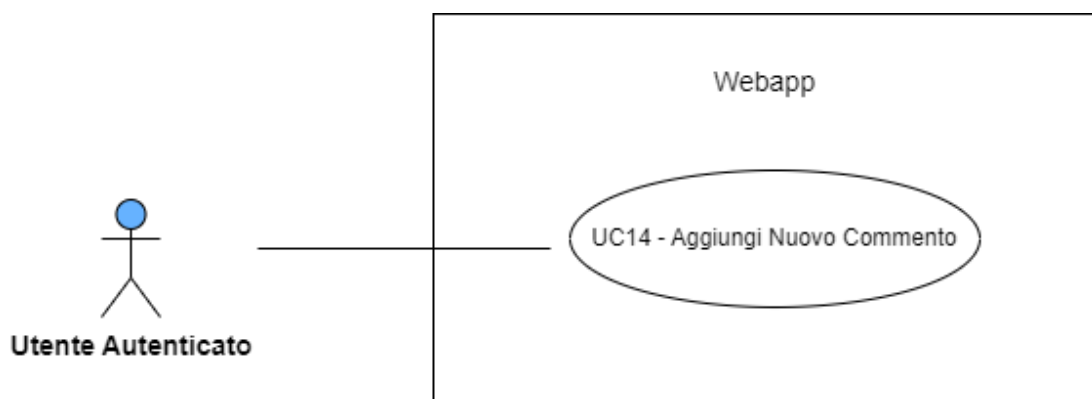
Tabella 3.15: UC11**UC12, UC13 - Download Allegato****Figura 3.11:** UC12 - UC13

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 3.16: 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 3.17: UC13**UC14 - Nuovo Commento****Figura 3.12:** 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 3.18: UC14

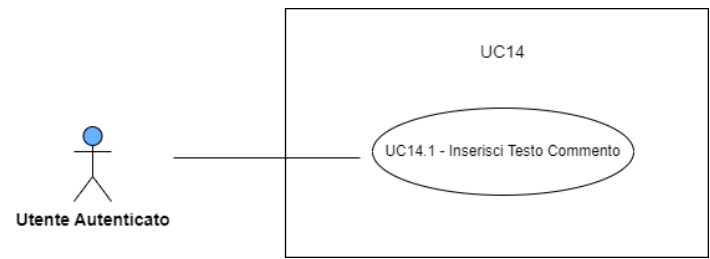


Figura 3.13: UC14.1

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

Tabella 3.19: UC14.1

UC15, UC16 - Eliminazione

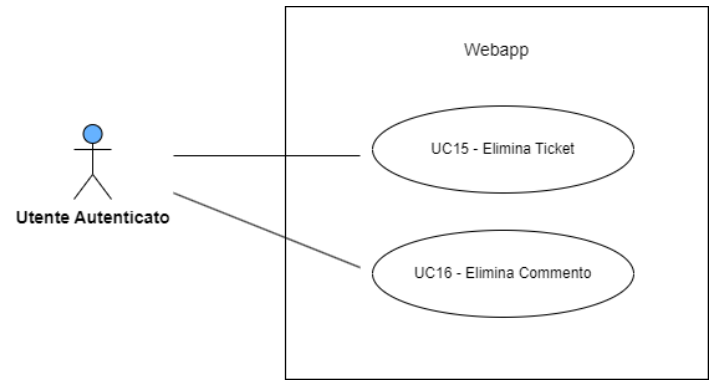


Figura 3.14: UC15 - UC16

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 3.20: 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 3.21: UC16**UC17**

Attore primario	Utente autenticato
Precondizioni	L'utente ha provato a 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 3.22: UC17

3.3 Tracciamento dei requisiti

Dopo un'attenta analisi dei requisiti e degli use case 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 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.

3.3.1 Requisiti funzionali

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
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

Tabella 3.23: Tabella del tracciamento dei requisiti funzionali

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 visualizza 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

Tabella 3.24: Tabella del tracciamento dei requisiti funzionali

3.3.2 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

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

Tabella 3.25: Tabella del tracciamento dei requisiti di vincolo

Capitolo 4

Progettazione e codifica

Il capitolo inizialmente presenta gli strumenti e le tecnologie analizzate e utilizzate per la realizzazione del prodotto. Successivamente espone l'effettiva creazione delle classi del progetto, affiancate da una struttura preesistente, fondamentale al funzionamento del modulo.

4.1 Tecnologie e strumenti

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

HTML5

Tecnologia standard per la creazione di pagine web, studiata attraverso 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. Studiata attraverso 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 Bootstrap 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 *Bootstrap* 3 e 5 è nella gamma di funzionalità che offrono. La versione 5 è la più recente e possiede molte più funzionalità di quelle precedenti, adattandosi alle nuove feature di *HTML5*.

Ad oggi si cerca di migrare dalle versioni più vecchie a quelle più recenti.

Servlet

Le *servlet* permettono di soddisfare delle *request HTTP* proveniente dal 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 dall'estensione *.jsp* e costituiscono le vere e proprie pagine web visualizzate dall'utente, sono infatti codificate in *HTML* e *XML*^[9].

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 open-source* che supporta lo sviluppo di applicazioni web in Java con il pattern MVC. *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 viene 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*^[9].

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 dell'implementazione ma è *Hibernate*^[9] 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 creare e gestire gli oggetti dell'applicazione, detti anche *beans*.

Wro4j

Wro4j è uno strumento per 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 dalla pagina web.

Di solito, un browser può scaricare al massimo due risorse simultaneamente, il che può causare un rallentamento del caricamento della pagina quando ci sono numerose risorse da scaricare. *Wro4j* elimina questo problema comprendendo tutte le risorse in un unico file.

SVNKit

SVNKit è uno *toolkit Open-Source* e permette l'accesso in 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 essenziali, 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.xml*.

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 adeguate alla propria applicazione.

DBeaver

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

4.2 Progettazione

La fase di progettazione è stata una fase cruciale per la realizzazione del progetto. Prima ancora di iniziare la progettazione del lavoro assegnato è stato importante analizzare l'ambiente già esistente, capirne il funzionamento e individuare eventuali elementi che potevano servire al nostro scopo.

Una tecnica per una buona progettazione è stata concentrarsi su un elemento alla volta. Prendere in considerazione tutti gli elementi del progetto poteva sembrare più

efficace e veloce, ma così facendo si sarebbe perso il focus della funzione delle classi del prodotto.

Quindi l'idea è stata quella di analizzare e progettare una singola classe, controllarne il corretto funzionamento e poi procedere ed andare avanti con lo studio per la realizzazione delle prossime classi.

4.2.1 Base di Dati

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

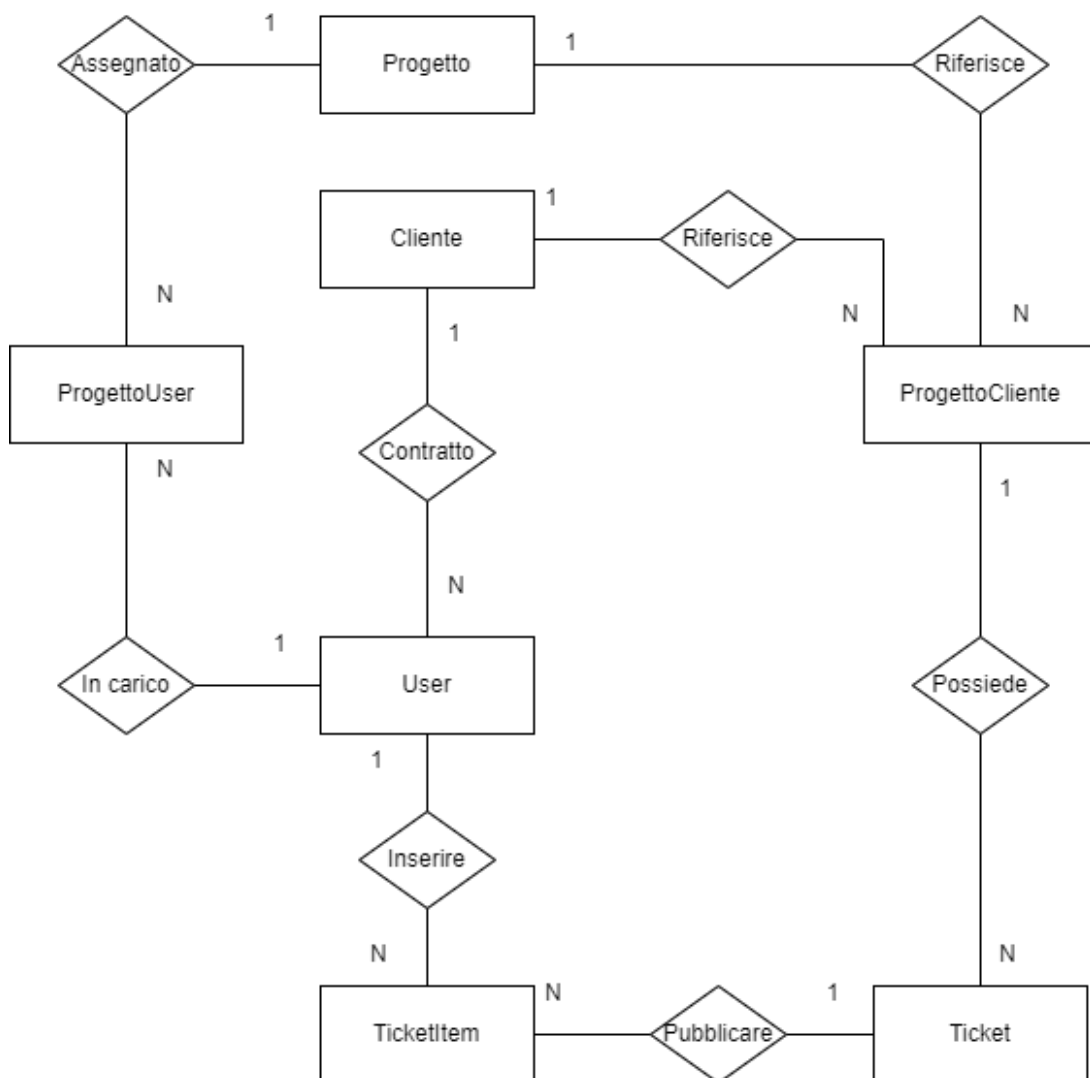
Tabelle preesistenti

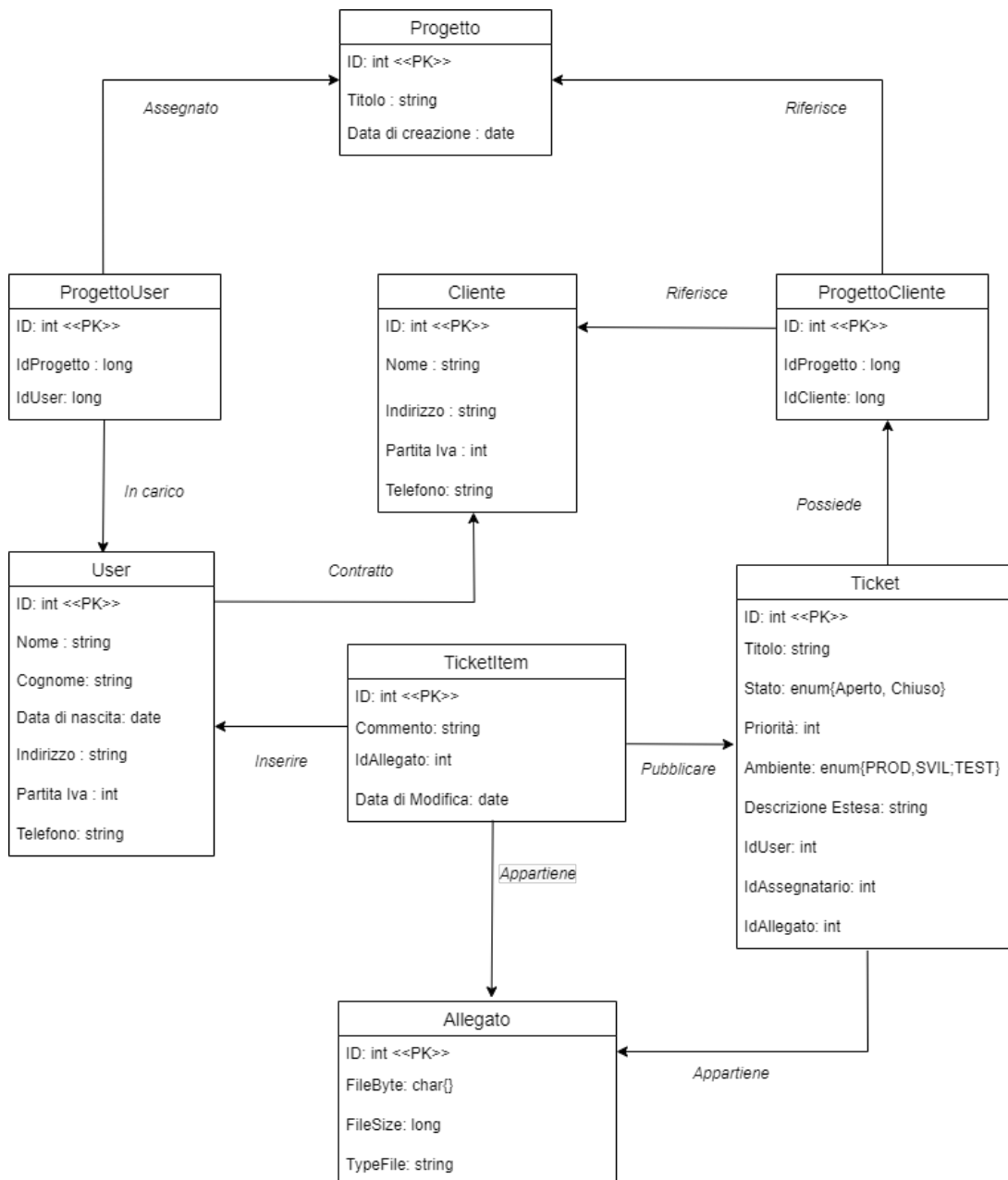
Il progetto utilizzava tabelle preesistenti per il suo scopo. Le tabelle erano le seguenti:

- **Cliente:** questa tabella rappresentava l'entità cliente e aveva tutti gli attributi necessari per definirne lo scopo all'interno della webapp. Il cliente non era la singola persona ma bensì l'azienda, a cui poi erano collegati sia i dipendenti sia i progetti richiesti da questa. Alcuni attributi presenti servono per le altre componenti della webapp;
- **User:** gli user erano tutte le persone coinvolte nell'azienda, interne ed esterne. Quindi la tabella comprendeva sia il personale di CWBI sia il personale delle aziende clienti;
- **Progetto:** questa tabella rappresentava un progetto dell'azienda. Aveva tutti gli attributi necessari ed era collegata ad un cliente. Un progetto poteva essere collegato a più aziende diverse;
- **ProgettoCliente:** questa tabella rappresentava la relazione tra progetto e cliente. Veniva chiamata in causa quando si doveva scegliere l'azienda e il progetto su cui si voleva aprire il ticket;
- **ProgettoUser:** questa tabella rappresentava la relazione tra progetto e user. Ad ogni progetto si potevano associare uno o più utenti.

Tabelle introdotte

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

**Figura 4.1:** Base di Dati - Relazioni delle tabelle del progetto

**Figura 4.2:** Base di Dati - Tabelle del progetto

Analisi delle tabelle

Cliente

<i>Id</i>	Identificativo univoco di ogni cliente.
<i>Nome</i>	Nome dell'azienda.
<i>Indirizzo</i>	Indirizzo della sede principale dell'azienda.
<i>Partita Iva</i>	Partita Iva dell'azienda
<i>Telefono</i>	Telefono dell'azienda

Tabella 4.1: Tabella Cliente**ProgettoCliente**

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

Tabella 4.2: Tabella ProgettoCliente**Progetto**

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

Tabella 4.3: Tabella Progetto

ProgettoUser

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

Tabella 4.4: Tabella ProgettoUser**User**

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

Tabella 4.5: Tabella User**Ticket**

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

Tabella 4.6: Tabella Ticket

<i>Ambiente</i>	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.
<i>Descrizione</i>	Descrizione del Ticket. Utile per approfondire il problema che si è riscontrato.
<i>IdUser</i>	Id dell'utente che ha aperto il ticket.
<i>IdAssegnatario</i>	Id dell'utente a cui è stato assegnato il ticket. Può essere cambiato durante il ciclo di vita del ticket.
<i>IdAllegato</i>	Id dell'allegato caricato al ticket. Può essere cambiato durante il ciclo di vita del ticket.

Tabella 4.7: Tabella Ticket**TicketItem**

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

Tabella 4.8: Tabella TicketItem**Allegato**

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

Tabella 4.9: Tabella Allegato

4.2.2 Architettura

Lo sviluppo dell'applicazione è avvenuto 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 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 fornire in output un determinato risultato.

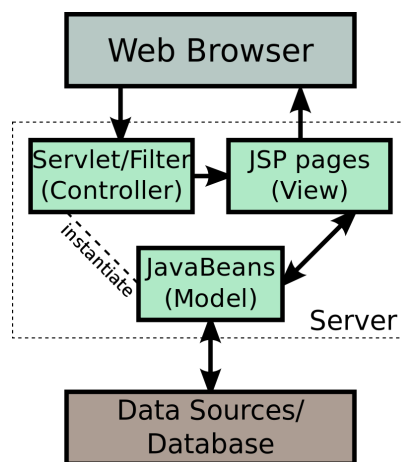


Figura 4.3: Schema MVC

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 una 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 iniziava da un modello preesistente e strutturato secondo gli standard aziendali. Questa parte era divisa in più livelli, ognuno dei quali possedeva un diverso scopo e diverse funzionalità. In generale, la struttura era composta come segue:

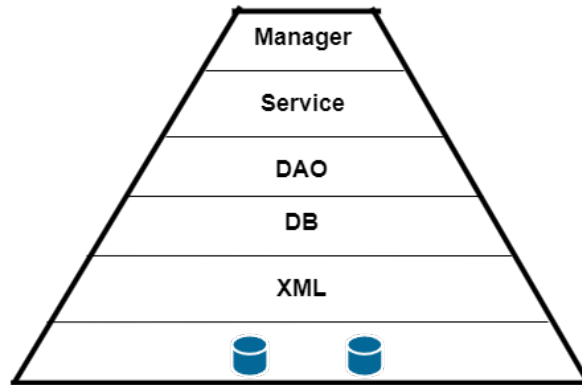


Figura 4.4: Struttura Model CWBI

Come detto in precedenza alcune delle classi già presenti sono state utilizzate per supportare le nuove entità introdotte con il progetto. Nella codifica del nuovo modello, ci si è orientati utilizzando come riferimento la figura 5.4.

Il primo passo per la creazione di una nuova classe è stato mappare i suoi attributi all'interno del file xml che conteneva la mappatura e specificava la struttura della tabella che sarebbe stata costruita poi sul database. Prendiamo come esempio la classe *Ticket*. È stato creato il file *ticket.xml* e all'interno è stata specificata la tabella *ticket-a* con tutti gli attributi.

Dopo è avvenuta la codifica della classe *TicketDB.java* che rappresentava l'oggetto vero e proprio utilizzato nella webapp. Importante specificare che gli attributi inseriti in questa classe dovevano essere uguali agli attributi specificati nella tabella *ticket-a* nel file di mappatura. La classe era quindi composta dagli attributi, i costruttori e le funzioni di *get* degli attributi.

La nomenclatura *DB* dopo il nome della classe è uno standard dell'azienda inserito per ogni nuovo modello che si introduce.

Successivamente sono state create due classi: *TicketDao* e *TicketDaoHibernate*.

TicketDao rappresentava l'interfaccia in cui erano presenti le firme di tutte le funzioni per la manipolazione dei dati secondo il pattern **Dao**^[9]: le operazioni **CRUD**^[9].

La classe *TicketDaoHibernate* invece implementava le funzioni presenti nell'interfaccia *TicketDao*. In realtà, l'esecuzione effettiva della funzione CRUD richiesta non avveniva in *TicketDao-Hibernate*, ma all'interno di ogni implementazione, veniva richiamata un'altra funzione con lo stesso scopo, implementata in una classe *Hibernate* separata. Questa classe, che funge da classe padre di *TicketDaoHibernate*, era responsabile dello svolgimento delle operazioni CRUD.

In generale, *TicketDaoHibernate* implementava le funzioni dell'interfaccia a cui si riferiva, ma le operazioni CRUD erano sempre effettuate dall'*Hibernate* padre.

Il prossimo passo è stato creare le classi service: `TicketService` e `TicketServiceImpl`. La classe `TicketService` era l'interfaccia al cui interno trovavamo le firme delle funzioni che la webapp consentiva di svolgere, come la funzione di ricerca. `TicketServiceImpl` invece era l'implementazione delle funzioni di `TicketService`. All'interno di ogni funzione il dato veniva manipolato e alla fine si richiamava la funzione della classe `TicketDao` specifica per il contesto. Si può dire che all'interno dell'implementazione di una funzione *service* il dato era manipolato e personalizzato con l'obiettivo di fornire il risultato desiderato.

È importante sottolineare che la funzione di ricerca non rientrava tra le operazioni *CRUD* e veniva implementata nel service. Infatti nella sua implementazione, questa funzione richiamava la funzione di `TicketDaoHibernate` che avrebbe effettuato la **lettura** del dato corrispondente ai parametri di ricerca.

Di conseguenza, tutte le funzioni all'interno dell'applicazione finivano per ridursi a operazioni *CRUD*.

L'ultima fase consisteva nella codifica delle classi *manager*. Queste classi non sempre erano necessarie e fungevano da supporto per le classi service. Nel progetto **non** sono state inserite classi *manager*.

Controller

Lo scopo dei *controller* all'interno dell'applicazione era 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* aveva 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 fungevano da *controller* erano marcate dall'annotazione `@Controller` di Spring.

Un'altra annotazione Spring presente era `@Autowired` che serviva per indicare le dipendenze dei *bean* (classi). Infatti all'interno dei controller si potevano trovare degli oggetti *Service* utilizzati per le operazioni sul Model. Quindi, quando veniva istanziato il controller, si creavano anche gli oggetti all'interno di esso.

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

Cartella Form

- **TicketForm**: questa classe identificava i campi di input presenti alla creazione o alla modifica di un ticket, detti appunto *Form*. Quindi quando l'utente compilava i campi, popolava gli attributi di questa classe. Le caratteristiche di **TicketForm** dovevano essere adeguate alla controparte dell'oggetto Ticket nel modello. Infatti il *controller* che si occupava della creazione e della modifica, aveva il compito di costruire l'oggetto Ticket partendo dall'oggetto TicketForm appena popolato.

- **TicketSearchForm**: questa classe identificava i campi di input presenti alla ricerca di un ticket. L'utente per effettuare la ricerca di un ticket poteva scegliere o compilare dei filtri, rappresentati da **TicketSearchForm**.
Le caratteristiche di tale classe erano redatte in base ai tipi di filtri che si volevano fornire all'utente e dovevano essere adeguati per le proprietà dell'oggetto su cui si stava effettuando la ricerca.

Cartella Action

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

- : come si può leggere, la classe non presenta la nomenclatura *Form*. Infatti questa *Action* si occupava di gestire le pagine della webapp che non possedevano form al loro interno. Il dettaglio del Ticket era gestito da **TicketAction** in quanto non possedeva nessun tipo di campo da compilare.
Erano presenti diverse funzioni, come ad esempio la funzione di caricamento della pagina di dettaglio di un Ticket. In questa funzione veniva 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 occupava di gestire le *Action* che riguardavano 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 entrava nella pagina di creazione di un ticket, i campi dell'oggetto **TicketForm** erano inizializzati vuoti dall'*Action* *input* e dovevano essere compilati dall'utente. Al salvataggio, veniva richiamata un'altra *Action* che si occupava di prendere i valori presenti nell'oggetto **TicketForm** e creare un nuovo oggetto *Ticket* con i dati prelevati.
Alla fine, si utilizzava l'oggetto **TicketService** per salvare il nuovo oggetto *Ticket*.
- **TicketSearchFormAction**: l'ultimo *controller* era utilizzato per le pagine di ricerca di un ticket e utilizzava l'oggetto **TicketSearchForm** per le proprie funzionalità.
Quando si entrava nella pagina di ricerca, i campi dell'oggetto **TicketSearchForm** che rappresentavano i filtri di ricerca, erano inizializzati vuoti ed era l'utente poi a riempirli. Quando si effettuava la ricerca, veniva richiamata la *Action* di *search*, dove si costruiva un nuovo oggetto *Ticket* a seconda delle caratteristiche (filtri) dell'oggetto **TicketSearchForm**.
Veniva quindi utilizzato l'oggetto **TicketService** per richiamare la funzione di ricerca che prendeva in input un oggetto *Ticket* e confrontava quale dei ticket presenti nel database aveva i valori uguali a quelli del ticket passato. Venivano così trovati tutti i ticket corrispondenti ai filtri inseriti.

View

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

La **View** era supportata anche da diversi *framework* come *Bootstrap* che forniva delle classi per personalizzare il contenuto della pagina andando a codificare tali classi direttamente nel attributo "class" dei tag di HTML.

Alle pagine jsp era 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 entrava in gioco un'ulteriore *framework*, senza il quale, non era possibile il funzionamento della webapp cos' com'è stata pensata e codificata da CWBI: **Struts**.

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

4.3 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

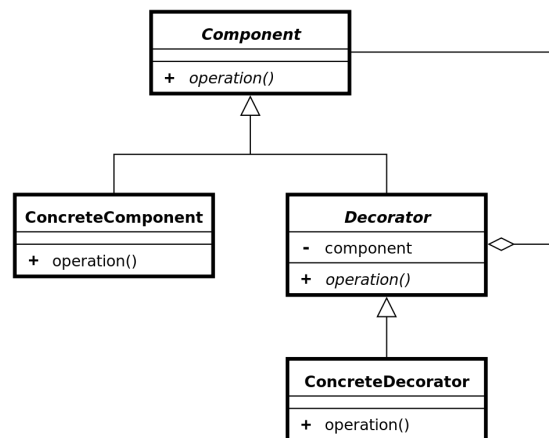


Figura 4.5: Pattern decorator

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

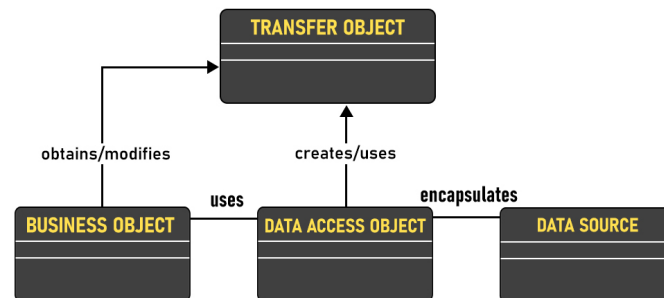


Figura 4.6: Pattern DAO

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 operazioni 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.

Capitolo 5

Verifica e validazione

Il seguente capitolo ha lo scopo di mostrare le tecniche di verifica e validazione del progetto, secondo le linee guida apprese durante il corso di Ingegneria del Software. Verificare un prodotto ha l'obiettivo di controllare se l'introduzione di nuovi elementi nel codice ha generato dei problemi e se rispetta i requisiti designati.

La validazione serve ad approvare il progetto qualora soddisfatti tutti i requisiti imposti.

5.1 Processo di Verifica

Il processo di verifica è stato attuato durante tutto lo sviluppo del progetto per verificare le nuove funzionalità e comportamenti introdotti.

L'approccio all'introduzione di nuovi elementi con le relative funzioni è stato costantemente controllato da *Roberto Martina*, responsabile di stage. Infatti la tecnica più efficiente per lo sviluppo del codice era codificare una delle parti di webapp, verificare che il codice appena introdotto funzionava correttamente nel suo insieme e soltanto dopo collegarlo alle altre parti di codice già sviluppate.

Quindi, non si procedeva per codificare il "tutto" perché i requisiti potevano risultare non soddisfatti e c'era il rischio di perdere l'obiettivo durante lo sviluppo; ma era rigoroso procedere per passi e per ognuno verificarne la correttezza.

5.1.1 Debugging

Una delle tecniche per verificare il giusto funzionamento dell'applicazione è stato il **Debugging**. Effettuando il debug significava controllare a *run-time* come si comportava l'applicazione durante l'interazione con l'utente. Dopo aver introdotto una nuova funzione per l'applicazione si poteva effettuare il debug per verificare che la funzione introdotta funzionava nel corretto modo e non introduceva nuovi problemi per gli altri elementi già presenti.

Lo strumento di *debug* utilizzato per verificare il progetto è quello messo a disposizione da **IDE Eclipse**. *Eclipse* mette a disposizione sia il *debug* del codice, ma anche il *debug* per il server su cui viene eseguita la webapp. Una feature di Eclipse è l'introduzione dei *breakpoint* che permettono di verificare il codice linea per linea.

Inserendo un *breakpoint* su una riga, l'applicazione si fermerà su quel *breakpoint* durante la sua esecuzione e il programmatore con i tasti F, F10, F11, F12 potrà proseguire per ogni riga in modo da individuare la posizione esatta del possibile errore.

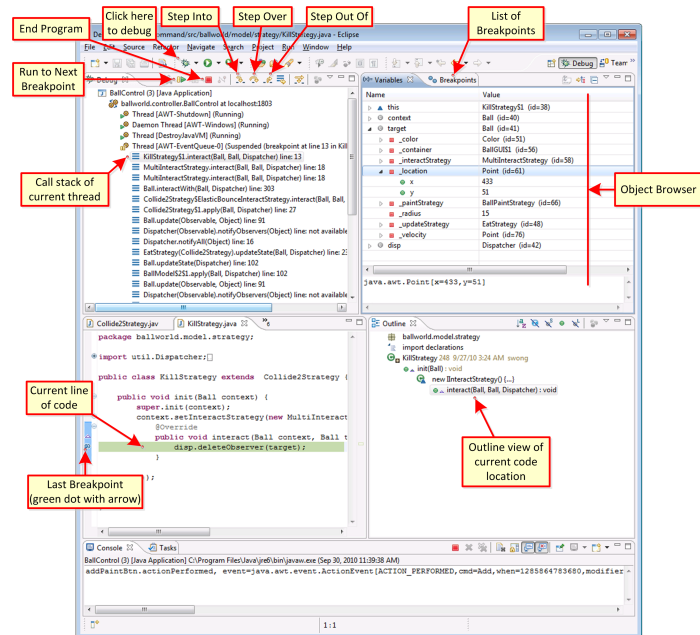


Figura 5.1: IDE Eclipse - Debug

5.2 Processo di Validazione

Il processo di validazione è stato eseguito insieme al tutor di tirocinio *Roberto Martina*. Durante gli ultimi giorni sono stati eseguiti tutti i test e confermata la validità della webapp prodotta rispetto ai requisiti definiti all'inizio dello stage.

Oltre alla conformità ai requisiti è stato ritenuto parte fondamentale di validità del prodotto anche il rispetto dei canoni della struttura aziendale presente.

Il tutor ha posto quindi molta attenzione anche su questo aspetto dato che non seguire la struttura designata portava ad un uso errato dei *pattern* e dei *framework* utilizzati. La struttura del codice aziendale era costruita con la consapevolezza di agevolare il programmatore nella creazione di nuove classi.

Capitolo 6

Prodotto finale

6.1 Pagina iniziale

Una volta effettuato il *login* l'utente poteva scegliere il modulo a cui voleva accedere della webapp. L'utente, accedendo al modulo Ticket, veniva indirizzato in una pagina che era composta da due sezioni:

- **Home Ticet;**
- **Menu Ticket.**

6.1.1 Home Ticket

La prima pagina visualizzata era la pagina di **Home Ticket**. In questa pagina erano presenti tre liste, ognuna con una modalità di visualizzazione di ticket diversa:

- **Aperti da me:** visualizzava tutti i ticket aperti dall'utente che stava accedendo la pagina;
- **Assegnati a me:** visualizzava tutti i ticket assegnati all'utente che stava accedendo la pagina. Infatti un dipendente CWBI visualizzava i ticket assegnati a lui.
- **Recenti:** visualizzava gli ultimi dieci ticket aperti, in generale.

I miei ticket

Aperti da me (3)

Assegnati a me (0)

Recenti (3)

ID	Creazione	Ultima modifica	Descrizione	Tipo	Priorità	Stato
56	20/07/2023 - 08:24:21	20/07/2023 - 08:24:38	Gestione delle immagini profilo utente	EVOLUTIVA	1	OPEN
55	20/07/2023 - 08:20:03	20/07/2023 - 08:26:28	Allineamento CSS	ADATTATIVA	3	OPEN
54	20/07/2023 - 07:24:08	20/07/2023 - 07:24:08	Sistema di ticketing	EVOLUTIVA	1	OPEN

Ambiente	Data di scadenza	Assegnatario	Progetto	
SVIL	06/07/2023 - 09:30:00	AMISTA' Michael (amistamichael@gmail.com)	Gameometry	<div>Modifica</div> <div>Dettagli</div>
TEST	27/06/2023 - 09:35:00	PANTALEO Fabio (pantaleo.fabio01@gmail.com)	Gameometry	<div>Modifica</div> <div>Dettagli</div>
PROD	20/07/2023 - 17:20:00	Non assegnato	Ticket	<div>Modifica</div> <div>Dettagli</div>

Figura 6.1: Home Ticket

6.1.2 Menu Ticket

La seconda sezione disponibile all’entrata nel modulo Ticket era la pagina che mostra il menù, in cui si poteva scegliere di aprire un nuovo ticket oppure di effettuare una ricerca.

Menu Ticket

MENU TICKET	
ticket-new	>
Ricerca Ticket Progetto	>

Figura 6.2: Menu Ticket

6.2 Nuovo Ticket

La creazione di un nuovo ticket si divideva in due step:

- 1. Si doveva scegliere l'azienda che stava aprendo il nuovo ticket;
- 2. In base all'azienda scelta, venivano visualizzati i progetti disponibili su cui aprire il ticket; si compilavano gli altri campi per l'apertura.

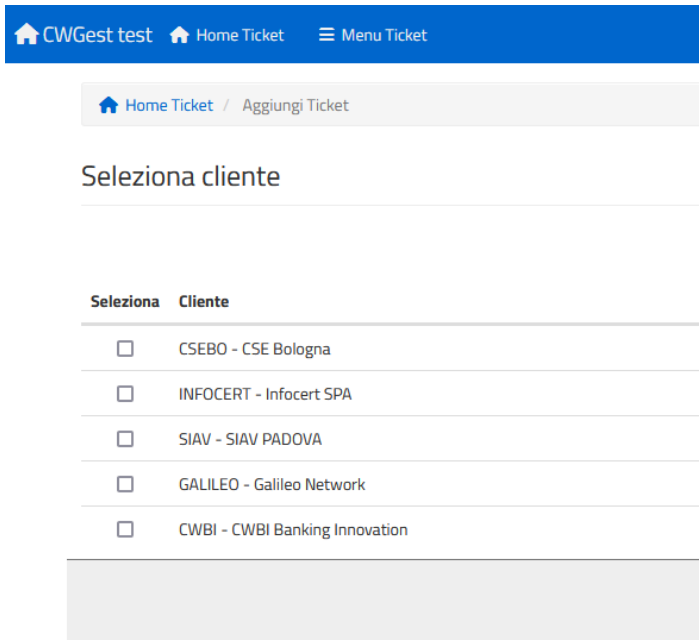


Figura 6.3: Nuovo Ticket - Step 1

Nuovo ticket

Dettagli*

Progetto*:
2023.20 - Gameometry

Proiettto

Priorità*:
1

priorità

Ambiente:
PROD

ambiente

Descrizione *

descrizione breve

B

I

U

14

A

descrizione lunga

Avanzamento del ticket:

Tipologia di manutenzione: EVOLUTIVA

tipologia di manutenzione

Stato: OPEN

stato

Data di scadenza:

data di scadenza

Allegato:

sfoglia

Figura 6.4: Nuovo Ticket - Step 2

6.3 Ricerca Ticket

La pagina di ricerca Ticket offriva un spazio in cui erano visualizzati tutti i ticket secondo i criteri di ricerca. I filtri selezionabili si trovavano sul menu a sinistra della pagina e per effettuare la ricerca bastava premere sul pulsante "Cerca". Per ogni ticket presente nella lista erano presenti i campi delle caratteristiche che lo rappresentavano. Inoltre erano disponibili i pulsanti di modifica e di dettaglio.

RICERCA TICKET

Istituto: 2272 - CWBI

ID:

Descrizione:

Tipo: Tutti

Priorità: Tutti

Ambiente: Tutti

Stato: Tutti

Assegnatario: Tutti

Progetto: Tutti

from-date-time: X

to-date-time: X

Cerca

Reset

Ricerca Ticket Progetto

+ Nuovo

Show 1 - 3 of 3 records

PrimoPrec1SuccUltimo

ID	Creazione	Ultima modifica	Descrizione	Tipo	Priorità	Stato	Ambiente	Data di scadenza	Assegnatario	Progetto	
56	20/07/2023 - 08:24:21	20/07/2023 - 08:24:38	Gestione delle immagini profilo utente	EVOLUTIVA	1	OPEN	SVIL	06/07/2023 - 09:30:00	AMISTA' Michael (amistamichael@gmail.com)	Gameometry	Modifica Dettagli
55	20/07/2023 - 08:20:03	20/07/2023 - 08:26:28	Allineamento CSS	ADATTATIVA	3	OPEN	TEST	27/06/2023 - 09:35:00	PANTALEO Fabio (pantaleo.fabio01@gmail.com)	Gameometry	Modifica Dettagli
54	20/07/2023 - 07:24:08	20/07/2023 - 07:24:08	Sistema di ticketing	EVOLUTIVA	1	OPEN	PROD	20/07/2023 - 17:20:00	Non assegnato	Ticket	Modifica Dettagli

PrimoPrec1SuccUltimo

Figura 6.5: Ricerca Ticket

6.4 Dettaglio Ticket

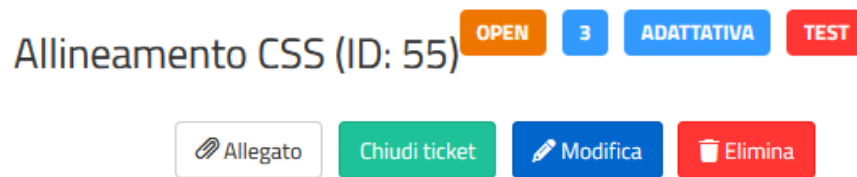
Il dettaglio di un ticket era visualizzato attraverso il pulsante di "Dettaglio" presente in ogni riga delle liste di ticket. La pagina di dettaglio, come dice il nome, visualizzava le caratteristiche del ticket in modo dettagliato. La parte iniziale della pagina era composta da:

Parte sinistra

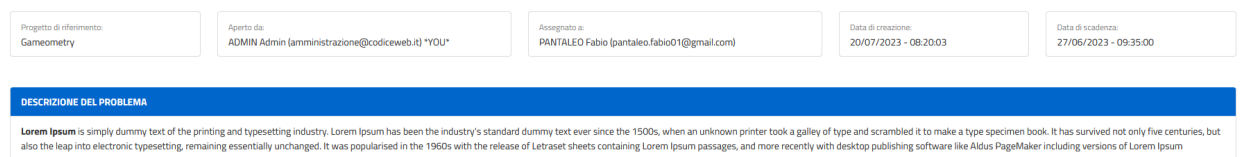
- Titolo del Ticket;
- Stato;
- Priorità;
- Tipo;
- Ambiente.

Parte destra

- Download dell'allegato;
- Chiudi/Apri;
- Modifica;
- Elimina;

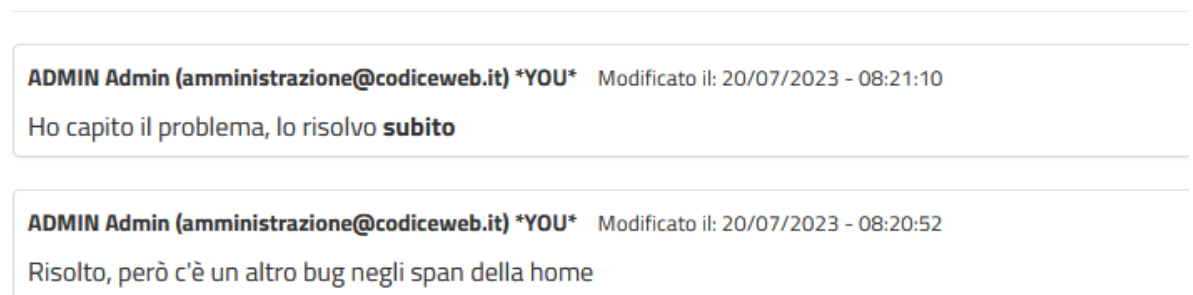
**Figura 6.6:** Parte iniziale Dettaglio Ticket

La parte centrale era composta dalle rimanenti caratteristiche del ticket.

**Figura 6.7:** Parte Centrale Dettaglio Ticket

A fine pagina era posizionata la sezione dei commenti in cui erano presenti tutti i commenti inseriti dagli utenti, con la possibilità di pubblicare un nuovo commento.

Note (2)





**Figura 6.8:** Commenti Dettaglio Ticket

6.5 Commento Ticket

Questa pagina serviva per lasciare una nota per un ticket.

Nuova nota

Descrizione*:

B *I* U  14 ▼ **A** ▼    ▼

Prendo in **carico** l'attività

Descrizione

Allegato:


 sfoglia

Figura 6.9: Commenti Dettaglio Ticket

Capitolo 7

Conclusioni

7.1 Consuntivo finale

La pianificazione redatta ad inizio stage sul *Piano di Lavoro* ha subito delle variazioni della variazione rispetto a quante ore sono state effettivamente dedicate per ogni attività.

La formazione iniziale e lo studio delle tecnologie utilizzate in azienda hanno richiesto più di 40 ore, come previsto. Infatti la comprensione della struttura aziendale è stata una delle parti cruciali per iniziare lo sviluppo del progetto. Quindi non solo sono stati affrontati e appresi le nuove tecnologie e framework già presenti, ma è stato fondamentale capire come questi elementi interagiscono tra di loro nell'architettura dell'azienda.

Un'altra attività che ha richiesto più di quanto previsto è stata lo sviluppo della soluzione ma non per la complessità di codifica del codice, bensì per la *way of working* da intraprendere. Essenziale era avere le idee chiare e procedere per passi; non immaginare fin da subito l'intero prodotto con tutte le sue parti, ma lavorare su una singola componente per volta e svilupparne le caratteristiche. Per le attività rimanenti le ore previste sono state più che sufficienti e quelle non utilizzate sono state dedicate alle attività già citate sopra.

Le 300 ore totali previste sono state quindi rispettate grazie soprattutto agli strumenti messi a disposizione dall'azienda che facilitano il lavoro rendendolo veloce ed intuitivo.

7.2 Raggiungimento degli obiettivi

Gli obiettivi definiti con il tutor *Roberto Martina* ad inizio stage sono stati ampiamente raggiunti. Lo studio e la comprensione della struttura aziendale sono stati eccellenti e cruciali per il raggiungimento degli altri obiettivi, come lo sviluppo della webapp.

Tutte le componenti e funzionalità previste per il modulo sviluppato sono state integrate e testate per verificarne il corretto funzionamento, così come i requisiti obbligatori raccolti nella fase di "analisi dei requisiti" sono stati soddisfatti.

7.3 Valutazione degli strumenti utilizzati

L'*IDE* utilizzato è Eclipse, già personalmente conosciuto in una versione molto precedente a quella utilizzata e riscoperto ancora più diretto e di facile utilizzo, con la possibilità di integrare nuove componentistiche per utilizzo professionale.

Per la visualizzazione dell'applicazione sul browser è stato utilizzato *Tomcat* che si è rivelato un applicativo perfetto per lo sviluppo di applicazioni web. Alla creazione di un server *Tomcat* c'era la possibilità di scegliere tra diverse opzioni in base alla versioni di Java e dei framework presenti. Configurabile facilmente con i file *jar* essenziali per il corretto funzionamento del progetto.

In generale gli strumenti utilizzati sono stati adeguati per lo sviluppo di un'applicazione Java e li prenderò sicuramente in considerazione anche per progetti futuri.

7.4 Miglioramenti e future estensioni

Il modulo della webapp sviluppato, come detto precedentemente, ha soddisfatto tutti i presupposti elaborati e concordati per lo stage con il tutor. Anche il lato utente è stato ampiamente trattato, adattandolo alle linee guida dell'azienda per quanto riguarda il *front-end*.

Tuttavia una crescita cruciale che dovrebbe affrontare il modulo riguarda il rispetto delle norme di accessibilità attuali. Infatti la maggior parte degli elementi di *front-end* non sono allineati per quanto riguarda l'accessibilità, la quale negli ultimi anni è diventata la regola fondamentale che ogni applicazione web cerca di integrare.

Un ulteriore miglioramento deve essere l'ottimizzazione delle interazioni tra il modulo sviluppato e quelli già presenti. L'obiettivo punta al rifacimento di alcune delle classi presenti all'interno della *webapp*, in modo da avere una più solida connessione con il nuovo modulo.

Durante la fase di sviluppo è stata ideata una nuova feature da integrare nel progetto ma lasciata in disparte per una futura estensione del modulo. L'idea era quella di fornire all'utente un'interfaccia di messaggistica per discutere con gli altri utenti sui diversi ticket aperti in tempo reale. L'introduzione di questa funzionalità richiedeva tuttavia un'ulteriore fase di analisi dei requisiti che avrebbe portato ad uno slittamento della data di fine progetto.

Glossario

A

- **Ajax**: tecnica per lo sviluppo di pagine dinamiche che non richiedono la ricarica della pagina.

B

- **Baseapp**: applicazione sviluppata dall'azienda CWBI.
- **Bootstrap**: framework utilizzato per il design delle pagine web.

C

- **CRM**: sistema utilizzato dalle azienda per gestire i rapporti con i clienti.
- **CRUD**: acronimo che racchiude le quattro operazione principali di un'applicazione: *create, read, update, delete*.
- **Css**: linguaggio usato per gestire il design e la presentazione delle pagine web.
- **CWBI**: acronimo di **Codice Web Banking Innovation**.
- **CWGEST**: applicazione sviluppata dall'azienda CWBI.

D

- **DAO**: acronimo di **Data Access Object**, è un **design pattern** per lo sviluppo di applicazioni.

H

- **Hibernate**: framework che gestisce il rapporto tra database e applicazione Java.
- **HTML**: Acronimo di HyperText Markup Language permette di immaginare e formattare pagine collegate fra di loro attraverso link.

J

- **JSP**: acronimo di JavaServer Page.
- **JSTL**: acronimo di JSP Standard Tag Library, è un'estensione di JSP.

M

- **MVC**: pattern architetturale per lo sviluppo di un'applicazione.

R

- **Refactoring**: processo che prevede l'ottimizzazione del codice di un software.
- **Repository**: è una memoria in cui vengono memorizzati i file del sistema di versionamento.

S

- **Sistema di versionamento**: è un sistema per tener traccia delle modifica di un file o software.
- **Struts**: framework utilizzato per la comunicazione tra front-end e back-end.
- **SVNKit**: sistema di versionamento di Java.

T

- **Taconite**: framework utilizzato per le chiamate Ajax.
- **Ticketing**: sistema per gestire ticket.

U

- **UML**: acronimo di **Unified Modeling Language**, utilizzato per rappresentare l'architettura di un software.

X

- **XML**: linguaggio di markup per la rappresentazione di dati strutturati.

Bibliografia

- <https://www.cwbi.eu/it;>
- <https://www.html.it/guide/guida-html5;>
- <https://getbootstrap.com;>
- https://it.wikipedia.org/wiki/JavaServer_Pages;
- <https://www.html.it/pag/17162/servlet-e-jsp;>
- <https://it.wikipedia.org/wiki/JSTL;>
- https://it.wikipedia.org/wiki/Apache_Struts;
- <https://struts.apache.org;>
- <http://malsup.com/jquery/taconite/overview;>
- <https://svnkit.com;>
- https://it.wikipedia.org/wiki/Use_Case_Diagram;
- <https://hibernate.org/orm;>
- <https://spring.io/projects/spring-framework;>
- https://it.wikipedia.org/wiki/Spring_Framework;
- <https://wro4j.readthedocs.io/en/stable/GettingStarted;>
- <https://maven.apache.org;>
- https://it.wikipedia.org/wiki/Apache_Maven;
- <https://tomcat.apache.org;>
- <https://dbeaver.io;>
- <https://www.html.it/pag/18299/il-pattern-mvc;>
- <https://en.wikipedia.org/wiki/Model>
- <https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/che-cosa-sono-i-design-pattern;>
- https://it.wikipedia.org/wiki/Design_pattern;

- <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>;
- https://en.wikipedia.org/wiki/Dependency_injection;
- https://en.wikipedia.org/wiki/Inversion_of_control;
- <https://it.wikipedia.org/wiki/Decorator>;
- <https://italiancoders.it/decorator-pattern>;
- https://en.wikipedia.org/wiki/Data_access_object;
- <https://www.baeldung.com/java-dao-pattern>;
- <https://www.geeksforgeeks.org/data-access-object-pattern>.