Documentazione - Ingegneria del Software

Lorenzo Colombo (1081134)
Carlo Alberto Poggiu (1079843)
Gabriele Merli (1081373)

Software Life Cycle

Nella creazione del nostro software gestionale economico per cantieri, il team ha adottato un approccio di sviluppo agile, perfettamente allineato con la nostra metodologia lavorativa:

Enfasi sul Team e Interazioni: Grande importanza viene data al team, alle competenze dei suoi membri e alle loro interazioni. I compiti sono assegnati in base alle capacità individuali, minimizzando il tempo necessario per acquisire competenze non familiari. Se un membro incontra difficoltà in un compito, il team collabora per trovare una soluzione.

Gerarchia Team: Nel nostro processo di sviluppo, abbiamo optato per un approccio Agile, in quanto rispecchia efficacemente il nostro modo di lavorare. Seguendo il modello del Capo squadra programmatore, poniamo grande enfasi sul valore del team, sulle competenze individuali e sull'interazione tra i membri:

- Il team è al centro della nostra attenzione, valorizzando le abilità di ciascun membro. I compiti vengono distribuiti tenendo conto delle capacità specifiche di ciascuno.
- Qualora un membro del team identifichi una difficoltà o una competenza che eccede le sue capacità nel portare a termine un compito, lo comunica al resto del gruppo. Questo stimola un dialogo collettivo per esplorare e trovare soluzioni insieme.

Preferenza per il Software Funzionante: Si dà più valore a un software funzionante piuttosto che a una documentazione eccessivamente dettagliata. Questo permette di concentrarsi maggiormente sullo sviluppo, accelerando la pubblicazione del prodotto e riducendo l'uso di risorse. La documentazione può essere ampliata o modificata successivamente, se necessario.

Timeboxing nella Fase Alpha: Durante lo sviluppo della versione beta, abbiamo adottato la tecnica del timeboxing, dividendo lo sviluppo in intervalli temporali definiti per l'implementazione di specifiche funzionalità.

Apertura ai Cambiamenti: Siamo disposti a modificare il software in base alle richieste degli utenti o dei finanziatori, purché non contraddica i principi fondamentali del nostro prodotto.

Model-Driven e Data-Driven Architecture: In termini di progettazione, abbiamo adottato un approccio Model-Driven, definendo prima i modelli delle funzionalità desiderate, e un approccio Data-Driven, basato sulle molteplici interazioni con il database e la gestione dei dati. Questo rende possibili ulteriori implementazioni in versioni future senza la necessità di aggiornare o ricompilare il software.

Configuration Management

Per la gestione della configurazione del progetto, abbiamo utilizzato GitHub. Per migliorare la gestione degli aggiornamenti del codice e la possibilità di proseguire programmando in parallelo. Durante l'implementazione, eventuali errori o problemi imprevisti sono stati segnalati e gestiti con comunicazioni interne, venendo poi assegnati dal membro del team che individuava l'errore o dallo SCRUM master a chi aveva le competenze per risolverli. GitHub risulta essere uno strumento fondamentale per coordinare la collaborazione del team.

Organizzazione Team

La struttura del team di sviluppo segue il modello Agile Squad, aderendo al processo di sviluppo software prescelto. I membri del gruppo hanno ricevuto i seguenti incarichi specifici:

Gabriele Merli: Scrum Master
Lorenzo Colombo: Assistente programmazione - Aiuto Documentazione
Carlo Alberto Poggiu: Cura della Documentazione

Per massimizzare l'efficienza delle tempistiche, abbiamo scelto di frazionare i lavori in base ai principali pacchetti di lavoro inizialmente previsti. L'organizzazione dello sprint backlog è stata rispettata nella maggior parte dei casi, essendo stata progettata per facilitare il lavoro collaborativo e parallelo sui diversi componenti del software.

Qualità Software

La qualità del nostro software è stata verificata in maniera incrementale e valutata alla fine di ogni sprint. È stato nostro obiettivo assicurare che, nell'uso da parte degli utenti finali, il software rispettasse i seguenti criteri:

- Precisione: Garantire l'accesso accurato ai dati dei pazienti in conformità con i requisiti funzionali, evitando omissioni o duplicazioni di informazioni rilevanti.
- Affidabilità: Assicurare che il software fornisca risposte coerenti in tempi accettabili, anche nel peggiore degli scenari, senza causare perdite di informazioni inserite.
- Efficienza: Utilizzare le risorse del sistema ospitante in maniera ottimale per permettere un'efficace e celere consultazione dei dati.
- Integrità: Restringere l'accesso e le interazioni con il sistema ai soli utenti autorizzati e qualificati per determinate operazioni tramite l'introduzione del log-in.
- Facilità d'uso: Offrire un'esperienza intuitiva che permetta agli utenti autorizzati di gestire i dati senza necessità di istruzioni complesse, facilitati da un'interfaccia grafica molto intuitiva

Per quanto riguarda la manutenibilità del software, abbiamo definito le seguenti aspettative di qualità:

- Manutenibilità: Il codice dovrebbe essere sufficientemente leggibile e organizzato per consentire la facile identificazione e risoluzione di eventuali malfunzionamenti senza la necessità di riscrivere estese porzioni di codice.
- Testabilità: Assicurare che il collaudo del software per verificare il corretto funzionamento sia rapido ed efficace, facilitato dalla presenza di test automatizzati sia per il database sia per l'eseguibile.
- Flessibilità: Il software dovrebbe presentare una struttura che consente modifiche o aggiunte di nuove funzionalità senza compromettere il funzionamento delle versioni precedenti.

In relazione alla portabilità del software, l'unica esigenza delineata è stata quella di garantire che il software operi correttamente su sistemi diversi da quello di sviluppo.

Requisiti

Specifiche dei requisiti funzionali:

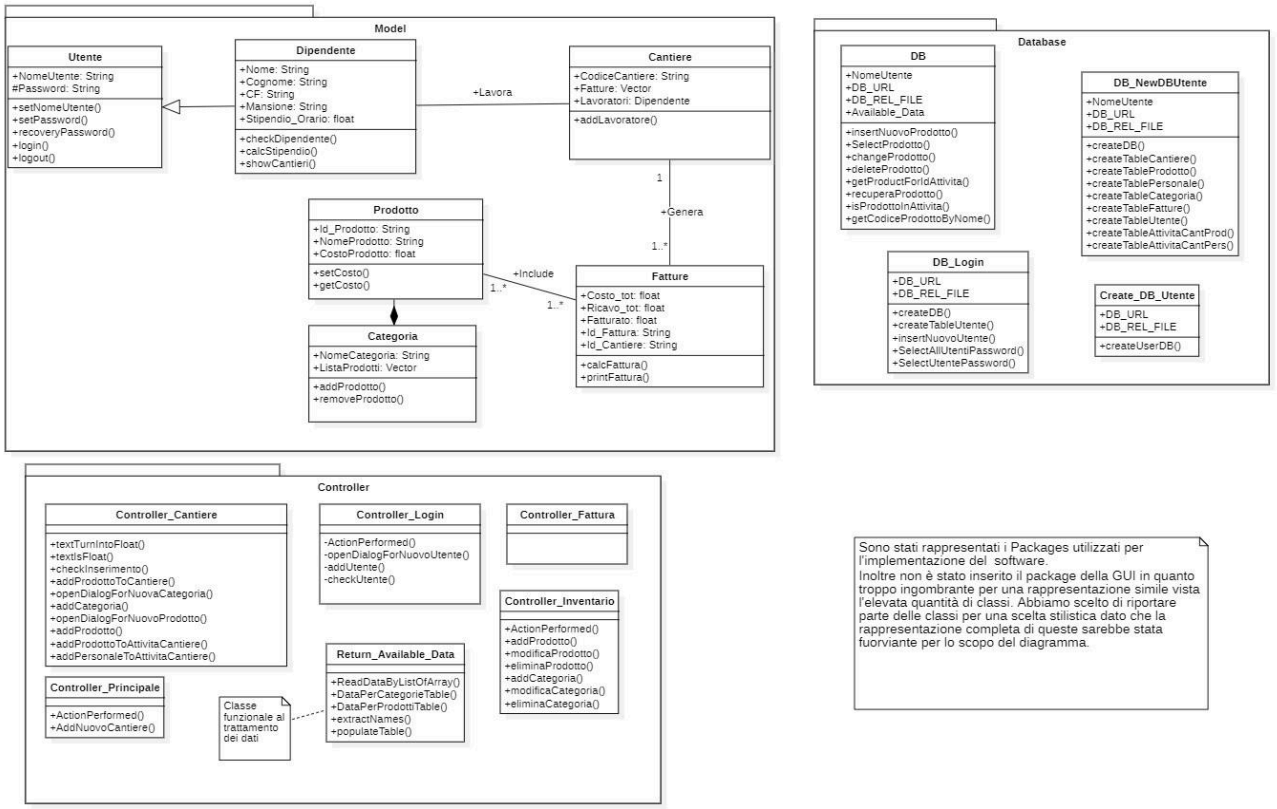
- **Must Have:**
  - Log-In del personale
  - Gestione Inventario
  - Gestione Cantieri
  - Gestione Personale
- **Should Have:**
  - Modifica dei prodotti e delle categorie
  - Modifica del personale
- **Could Have:**
  - Gestione del sistema di fatturazione
- **Won't Have:**
  - Import dati da .txt o file excel

UML

La modellazione del progetto è avvenuta mediante i seguenti modelli UML:

- Casi d'uso
- Attività
- Classe
- Sequenza
- Macchina a stati
- Componenti

Di seguito sono stati rappresentati nel diagramma delle classi i packages utilizzati per l'implementazione del software



Architettura del Software

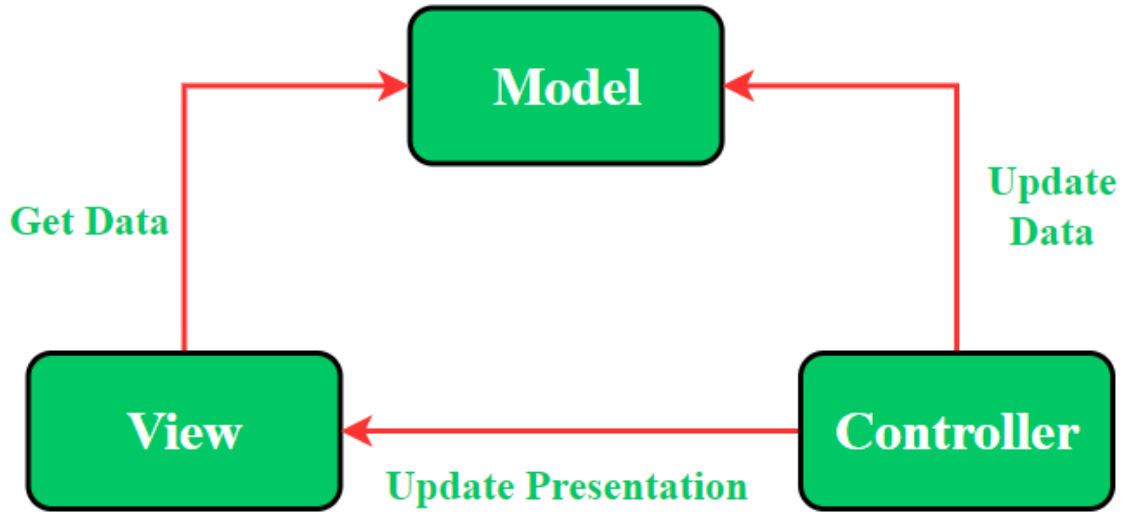
Factory Pattern: Per creare oggetti senza esporre la logica di creazione al client e referenziare il prodotto creato tramite un'interfaccia comune.

Singleton Pattern: Assicura che una classe abbia una sola istanza e fornisce un punto di accesso globale a tale istanza, come per il gestore del database.

Observer Pattern: Permette di notificare e aggiornare automaticamente diversi oggetti (osservatori) in caso di modifica dello stato di un oggetto (soggetto). Questo è utile per

notificare il sistema di gestione del magazzino e delle attività relative ai cantieri. MVC (Model-View-Controller)

Pattern: Separa l'applicazione in tre componenti principali: il modello (dati), la vista (interfaccia utente) e il controller (logica di business), facilitando così la gestione separata del codice in un'architettura a microservizi.



Design del software

La progettazione del software è stata documentata utilizzando il linguaggio UML. Di seguito sono illustrati in maniera esemplificativa il diagramma delle attività, che descrive il flusso delle operazioni eseguite dagli utenti del software e il diagramma dei casi d'uso.
Tutti i diagrammi utilizzati per il design del software sono disponibili nella cartella UML della stessa directory GitHub dov'è presente tutta la documentazione.

DiagrammaSequenza - Modifica Dipendente

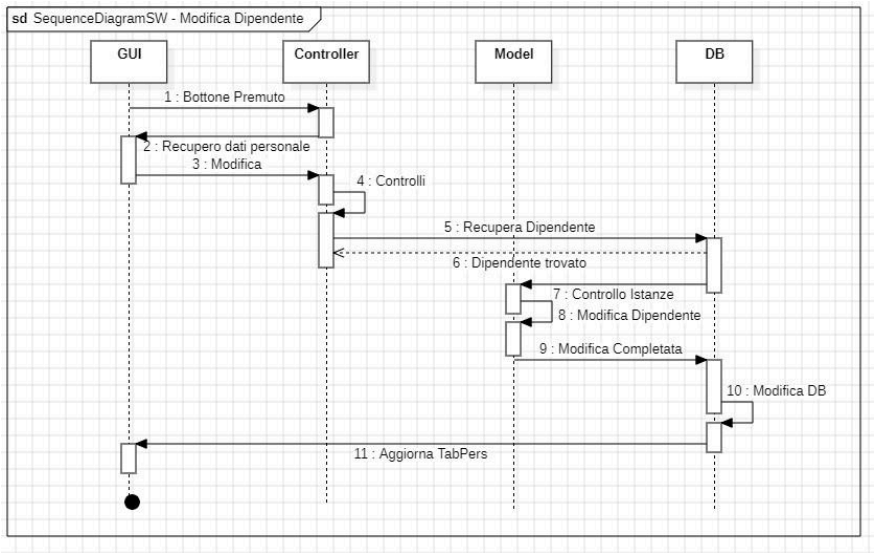
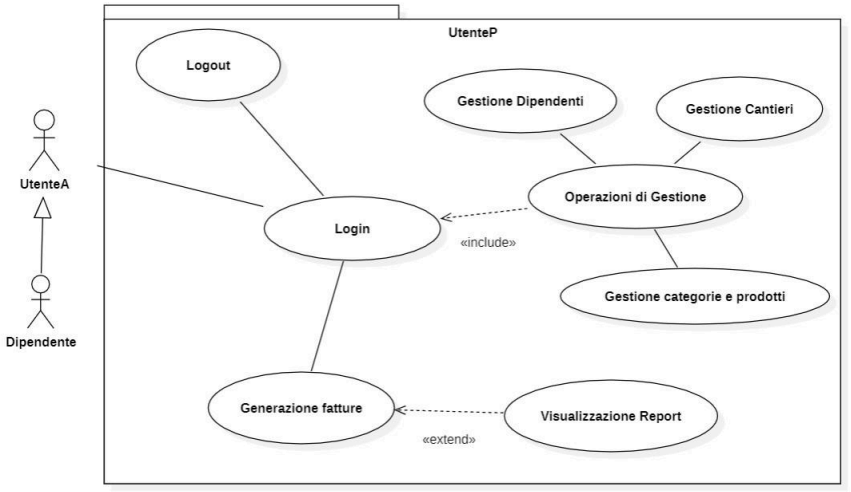
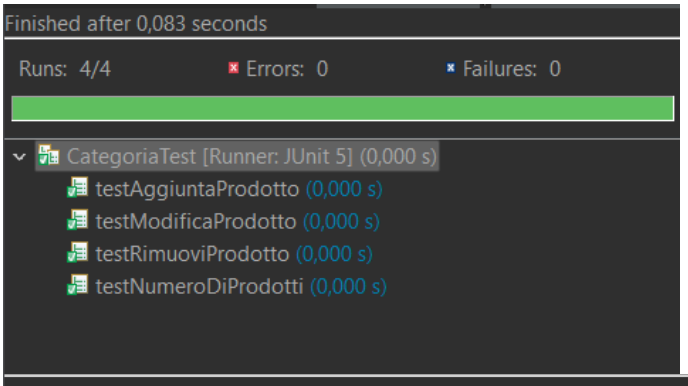


Diagramma Casi d'uso

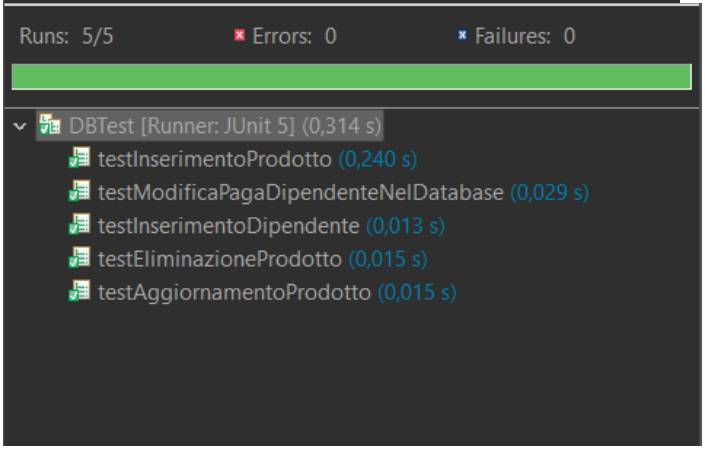


Testing

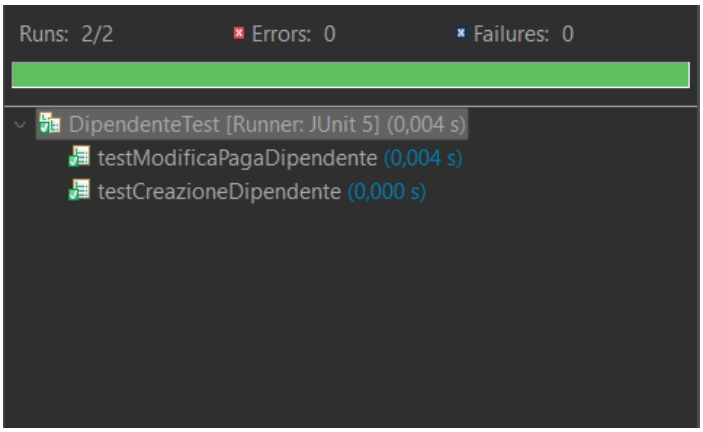
Durante lo sviluppo del software, abbiamo adottato un approccio incrementale nei test di funzionalità. Ogni volta che una nuova funzionalità viene implementata, si procede immediatamente con i relativi test di funzionamento. Questo processo ci ha permesso di identificare e correggere tempestivamente eventuali errori. Per ogni funzionalità testata, abbiamo eseguito la compilazione e l'esecuzione del codice, al fine di analizzare e risolvere gli errori che emergessero durante questi test. La maggior parte dei test è stata effettuata manualmente tramite la sostituzione di codice e la verifica del corretto funzionamento dei metodi implementati, inoltre sono state effettuate le opportune verifiche anche alla gestione del database con un software esterno (DB Browser).



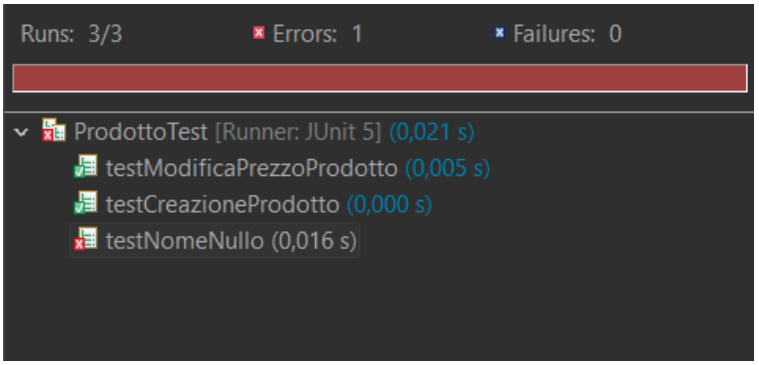
Questo test verifica il corretto funzionamento della classe Categoria nel suo insieme, testando singolarmente i metodi che consentono di aggiungere, rimuovere, modificare prodotti e contare il numero di prodotti associati a una categoria. Utilizza la libreria JUnit per eseguire i test automatici.



Questo test verifica il corretto funzionamento delle operazioni di inserimento, aggiornamento ed eliminazione dei prodotti e dei dipendenti nel database. Utilizza la libreria JUnit per eseguire i test automatici e si appoggia a un database SQLite per verificare i risultati delle operazioni di



database.Questo test verifica il corretto funzionamento della classe Dipendente, testando la creazione di un'istanza di Dipendente con dati di esempio e la modifica della paga oraria di un Dipendente esistente. Utilizza la libreria JUnit per eseguire i test automatici.



Questo test verifica il corretto funzionamento della classe Prodotto, testando la creazione di un'istanza di Prodotto con dati di esempio, la modifica del prezzo di un Prodotto esistente e la gestione di un'eccezione nel caso in cui venga fornito un nome nullo durante la creazione di un Prodotto. Utilizza la libreria JUnit per eseguire i test automatici.

Manutenzione

La manutenzione del software è stata fatta durante il suo sviluppo andando a correggere errori e problematiche.

La tecnica più comune di manutenzione è stata quella di Refactoring diversificando al meglio i nomi utilizzati all'interno del codice per aumentare leggibilità e comprensibilità. Inoltre spesso abbiamo rimosso o sostituito parti di codice inutilizzate/superflue o ripetitive, andando a migliorare ed ottimizzare il funzionamento del programma.

In una futura manutenzione del software si punta ad implementare quelle funzionalità extra che non sono state ancora implementate per la presentazione del progetto a causa di competenze non ben consolidate o ragioni di tempo. Inoltre il team di sviluppo è pronto ad eseguire nuove sessioni di manutenzione in seguito ad eventuali feedback da parte degli utenti che utilizzeranno il software.