

“Ingegneria del Software” 2021-2022

Docente: Prof. Angelo Furfaro

Organigramma Aziendale

Data	17/09/2024
Documento	Documento Finale – D3

Team Members		
Nome e Cognome	Matricola	E-mail address
Silvia Costantino	213341	cstslv99r71f112g@studenti.unical.it

Sommario

List of Challenging/Risky Requirements or Tasks	2
A. Stato dell'Arte.....	3
B. Raffinamento dei Requisiti.....	3
<i>B.1 Servizi (con prioritizzazione)</i>	3
<i>B.2 Requisiti non Funzionali</i>	4
<i>B.3 Scenari d'uso dettagliati</i>	4
<i>B.4 Excluded Requirements</i>	6
<i>B.5 Assunzioni</i>	7
<i>B.6 Use Case Diagrams</i>	7
C. Architettura Software	9
<i>C.1 The static view of the system: Component Diagram</i>	9
<i>C.2 The dynamic view of the software architecture: Sequence Diagram</i>	10
D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)	12
E. Scelte Progettuali (Design Decisions)	13
F. Progettazione di Basso Livello	14
G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs).....	16
Appendix. Prototype	18

List of Challenging/Risky Requirements or Tasks

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
Aggiungere dipendenti con due ruoli	22/08/2024	26/08/2024	La gestione dei dipendenti che posseggono più ruoli è stata effettuata attraverso una finestra di selezione interattiva che permette all'utente di indicare se un dipendente ha un altro ruolo nell'azienda. In caso di risposta positiva, il dipendente viene aggiunto anche all'altra unità organizzativa corrispondente al secondo ruolo. Questo approccio ha richiesto l'aggiornamento dell'interfaccia utente e della logica di gestione delle unità per supportare l'associazione di un singolo dipendente a più ruoli e unità organizzative.
Salvataggio dell'organigramma in un file	26/08/2024	02/09/2024	Si è implementato il salvataggio utilizzando un file .txt e la sfida maggiore è stata la gestione della formattazione e struttura dei dati per mantenere la gerarchia e le relazioni tra le unità organizzative. Inoltre, è stato sviluppato un metodo personalizzato per convertire l'organigramma in un formato leggibile da un file .txt
Caricamento dell'organigramma da un file	02/09/2024	06/09/2024	La sfida principale nel caricamento di un file di testo in un organigramma è stata la corretta interpretazione della struttura gerarchica e dei dati formattati. Si è, dunque, implementato un parser che legge il file di testo e ricostruisce l'organigramma, identificando le unità organizzative e le relazioni gerarchiche sulla base delle indentazioni e delle convenzioni di formattazione utilizzate. È stato, inoltre, necessario definire un formato standard per il file di testo.

A. Stato dell'Arte

L'organigramma aziendale è uno strumento cruciale per visualizzare la struttura interna di un'organizzazione, evidenziando ruoli, responsabilità e relazioni gerarchiche.

Recentemente, si è assistito a un'evoluzione significativa di questi organigrammi, in risposta alla crescente complessità delle organizzazioni e alla necessità di maggiore agilità. I modelli tradizionali, come l'organigramma gerarchico, sono stati affiancati da strutture più flessibili come gli organigrammi matriciali, che combinano gestione per funzione e per progetto, e quelli orizzontali, che riducono i livelli gerarchici per favorire la collaborazione. La digitalizzazione ha rivoluzionato la gestione degli organigrammi: strumenti avanzati permettono di creare rappresentazioni dinamiche, aggiornabili in tempo reale, e spesso integrate con i sistemi di gestione delle risorse umane. Questi organigrammi digitali non solo rappresentano la struttura, ma offrono anche funzionalità interattive, consentendo un'esplorazione dettagliata delle competenze e delle prestazioni dei dipendenti. In prospettiva, gli organigrammi stanno evolvendo verso soluzioni adattative, che si aggiornano automaticamente in base ai cambiamenti interni, e inclusivi, rappresentando la diversità e promuovendo la trasparenza. Queste innovazioni rendono l'organigramma uno strumento sempre più potente per ottimizzare la gestione delle risorse e supportare decisioni strategiche all'interno delle organizzazioni.

B. Raffinamento dei Requisiti

B.1 Servizi (con prioritizzazione)

I servizi che il sistema dovrà offrire affinché avvenga il corretto funzionamento dell'applicazione sono descritti di seguito, con annessa relativa importanza e complessità:

1. Creazione e Aggiornamento dell'Organigramma (Alta, Media)
 - a. Questo requisito funzionale rende possibile all'utente la creazione dell'organigramma e l'aggiunta o rimozione delle unità organizzative.
2. Gestione dei Dipendenti (Alta, Alta)
 - a. Tale requisito permette l'inserimento e la rimozione dei dipendenti nelle diverse unità organizzative.
3. Gestione delle Relazioni Gerarchiche (Alta, Media)
 - a. Questo requisito serve per la creazione e la modifica delle relazioni gerarchiche tra unità organizzative (ogni unità organizzativa deve poter avere una o più sotto-unità organizzativa).

4. Salvataggio e Caricamento dell'Organigramma (Alta, Alta)
 - a. Tale requisito permette il salvataggio dell'organigramma in una memoria secondaria per poi caricarlo successivamente.
5. Visualizzazione dell'Organigramma (Alta, Bassa)
 - a. Questo requisito rende possibile la visualizzazione a video dell'intero organigramma.

B.2 Requisiti non Funzionali

1. Usabilità Interfaccia Utente
 - a. Tale requisito è importante per avere un facile e intuitivo uso dell'applicazione.
2. Scalabilità
 - a. L'applicazione deve essere in grado di gestire un numero crescente di unità organizzative e dipendenti senza perdita di prestazioni.
3. Accessibilità
 - a. Questo requisito è necessario affinché l'applicazione possa essere accessibile a tutti gli utenti.
4. Performante
 - a. Il sistema deve garantire un buon livello di prestazioni, anche con un numero elevato di dipendenti e unità organizzative.

B.3 Scenari d'uso dettagliati

- Creazione di un nuovo organigramma:

1. L'applicazione viene avviata, mostrando il menu iniziale con le opzioni "Crea nuovo organigramma" o "Apri organigramma esistente".
2. L'utente seleziona "Crea nuovo organigramma" per iniziare a creare un nuovo organigramma.
3. L'applicazione chiude il menu principale e apre una finestra nel quale viene richiesto l'inserimento del nome dell'azienda; dopo l'invio, apre una seconda finestra che richiede il nome della radice dell'organigramma.
4. Si apre così la schermata principale dove è presente il nome dell'azienda (in alto, al centro), la prima unità organizzativa e, sulla destra, una tabella che mostrerà i nomi dei dipendenti con i rispettivi ruoli.

5. Inoltre, in questa schermata, l'utente trova i pulsanti: "Aggiungi Unità Organizzativa", "Rimuovi Unità Organizzativa", "Aggiungi Dipendente", "Rimuovi Dipendente" e "Salva Organigramma".

- Aggiunta di sotto-unità:

1. Dopo aver creato un'unità principale, l'utente decide di aggiungere sotto-unità.
2. L'utente clicca su "Aggiungi Unità Organizzativa" per iniziare a costruire l'organigramma: compare una finestra di dialogo dove l'utente seleziona l'unità organizzativa padre e, subito dopo, l'utente inserisce il nome dell'unità appena creata.
3. L'unità viene visualizzata nella schermata principale, connessa all'unità superiore indicata.

- Salvataggio dell'organigramma:

1. L'utente ha terminato di creare o modificare l'organigramma e desidera salvarlo.
2. L'utente clicca su "Salva Organigramma" nella barra dei pulsanti.
3. L'applicazione chiede all'utente di specificare il nome del file di salvataggio e la directory in cui desidera salvare l'organigramma.
4. L'utente inserisce le informazioni richieste e clicca su "Salva". Il file viene salvato e un messaggio di conferma viene mostrato all'utente.

- Caricamento di un organigramma esistente:

1. L'utente avvia l'applicazione e seleziona "Apri organigramma esistente" dal menu iniziale.
2. L'applicazione apre una finestra di dialogo che permette all'utente di cercare e selezionare un file di testo dove è inserito un organigramma precedentemente salvato.
3. L'utente seleziona il file desiderato e clicca su "Apri".
4. L'applicazione carica l'organigramma selezionato e lo visualizza nel pannello centrale, permettendo all'utente di continuare a modificarlo o visualizzarlo.

- Aggiunta di un dipendente:

1. L'utente seleziona l'unità organizzativa in cui desidera aggiungere un nuovo dipendente nell'organigramma.
2. L'utente clicca sul pulsante "Aggiungi Dipendente" presente nella barra degli strumenti o nel menu contestuale.
3. Compare una finestra di dialogo che richiede all'utente di inserire il nome e il cognome del dipendente.
4. Dopo viene chiesto all'utente di selezionare l'unità organizzativa dove verrà inserito il dipendente e di scrivere quale ruolo svolge.
5. A questo punto, appare una nuova finestra di selezione che chiede: "Il dipendente ha un altro ruolo?" con le opzioni "Sì" e "No".
 - Se l'utente seleziona "No": Il processo di aggiunta del dipendente si conclude, e il dipendente viene visualizzato sotto l'unità organizzativa selezionata nell'organigramma.
 - Se l'utente seleziona "Sì":
 1. L'applicazione chiede all'utente di inserire il nuovo ruolo.
 2. Dopo aver inserito il ruolo, l'utente viene guidato a selezionare un'altra unità organizzativa in cui aggiungere il dipendente con questo nuovo ruolo.
 3. L'utente conferma la selezione, e il dipendente viene aggiunto anche a questa seconda unità organizzativa, con il nuovo ruolo specificato.
6. L'organigramma viene aggiornato per riflettere tutte le aggiunte e i ruoli assegnati al dipendente (inseriti automaticamente anche nella tabella).

B.4 Excluded Requirements

Nessun servizio è stato escluso

B.5 Assunzioni

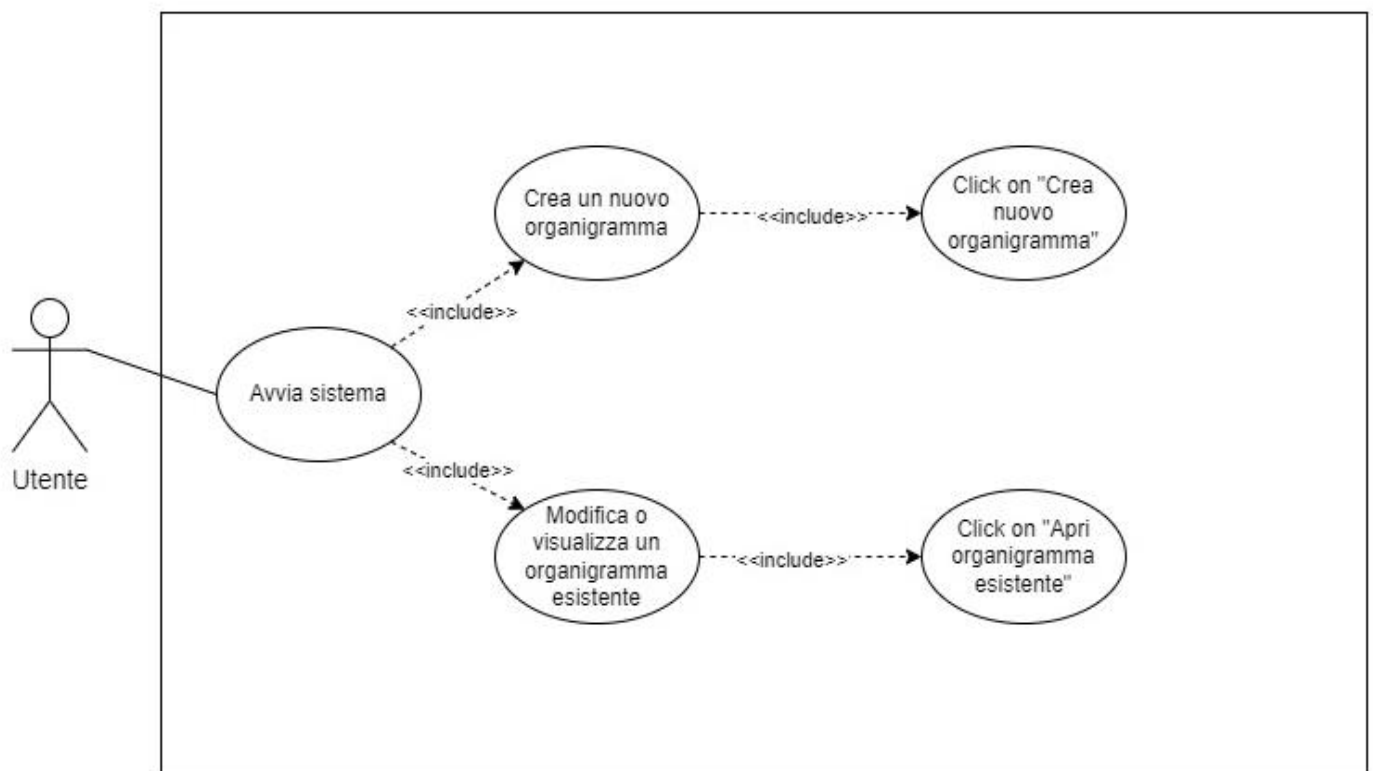
Si assume che l'utente possa aggiungere e rimuovere dipendenti solo se ha i privilegi di amministratore, per garantire che solo gli utenti autorizzati possano modificare l'organigramma.

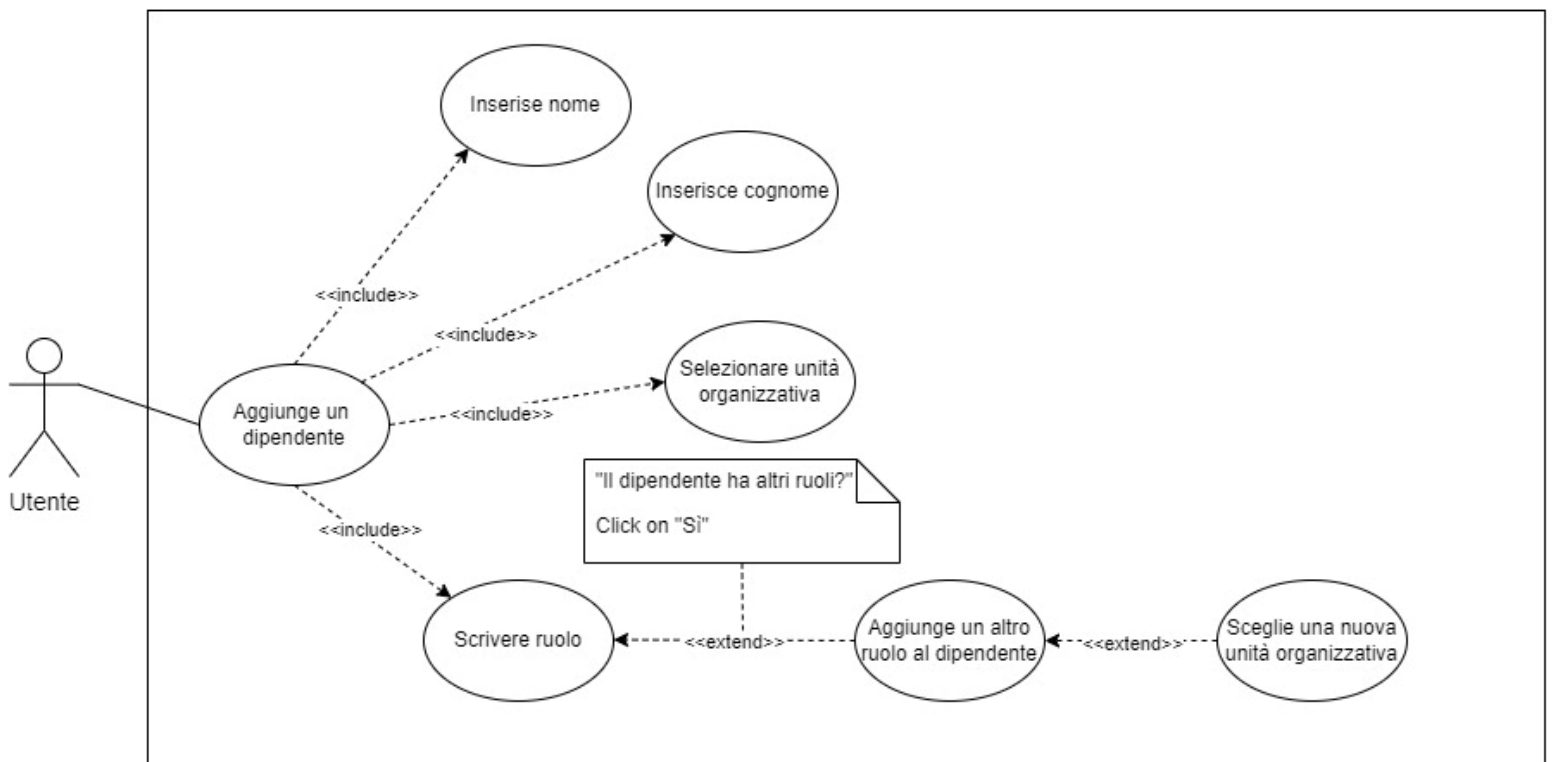
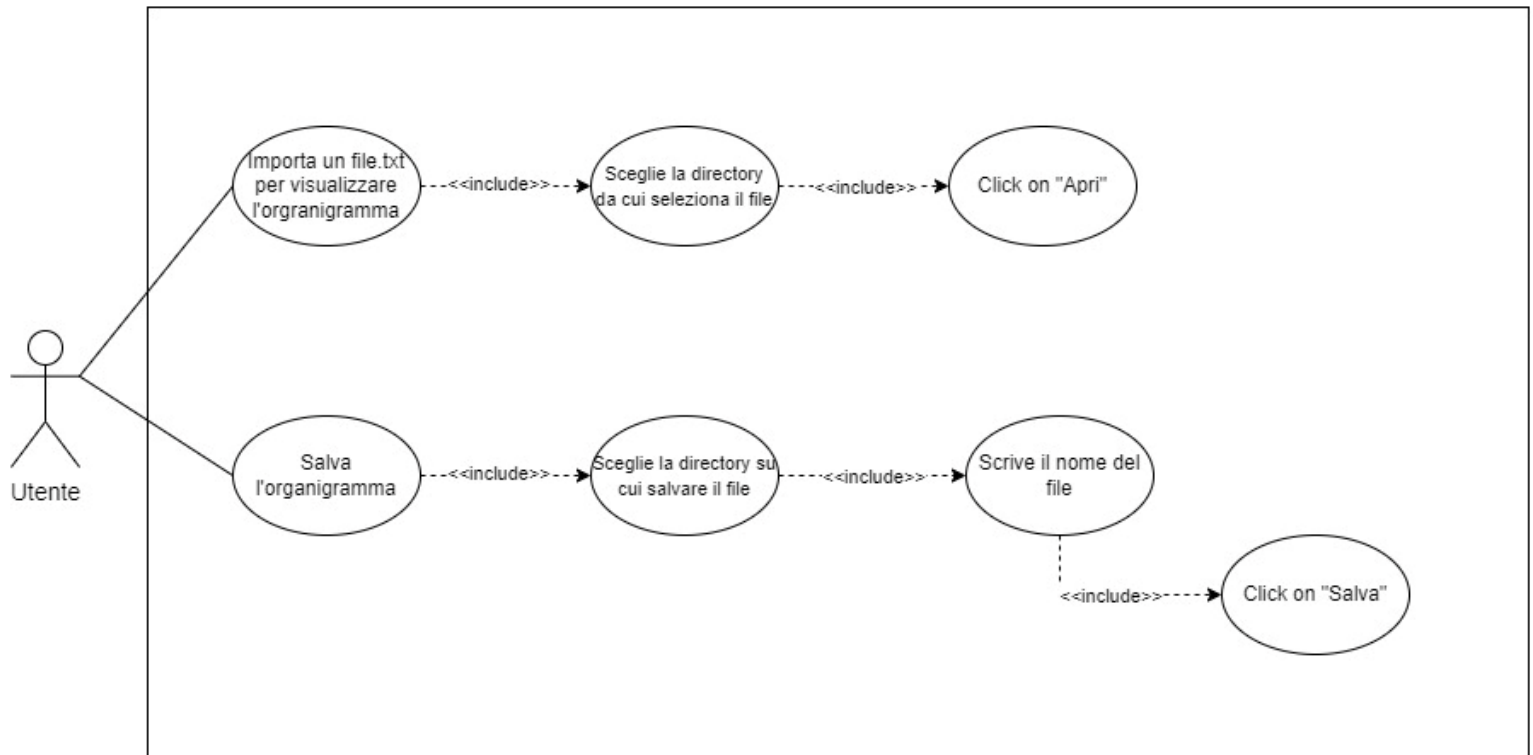
Si assume che l'utente non possa inserire ruoli duplicati per lo stesso dipendente all'interno della stessa unità organizzativa, per mantenere una chiara visualizzazione.

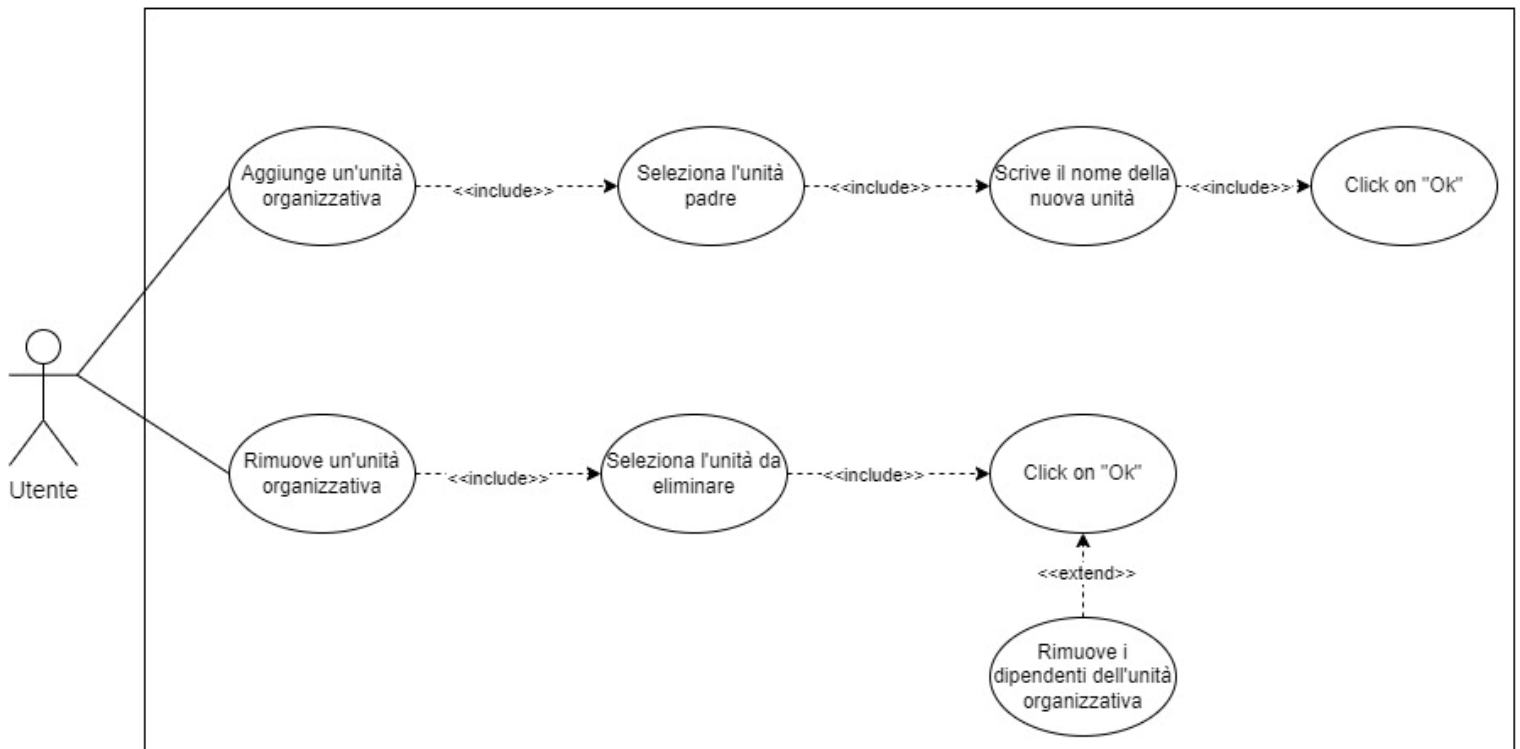
Si assume che i dati relativi ai dipendenti e alle unità organizzative vengano salvati in un formato compatibile con il sistema di gestione dei file dell'applicazione, per garantire una corretta importazione ed esportazione delle informazioni.

B.6 Use Case Diagrams

I seguenti diagrammi dei casi d'uso (Use Case Diagram) offrono una rappresentazione visiva delle principali funzionalità del sistema, mostrando in modo chiaro come gli utenti interagiscono con il sistema per svolgere compiti specifici. Ogni caso d'uso illustra un flusso di lavoro definito, fornendo una guida dettagliata sulle operazioni eseguibili.

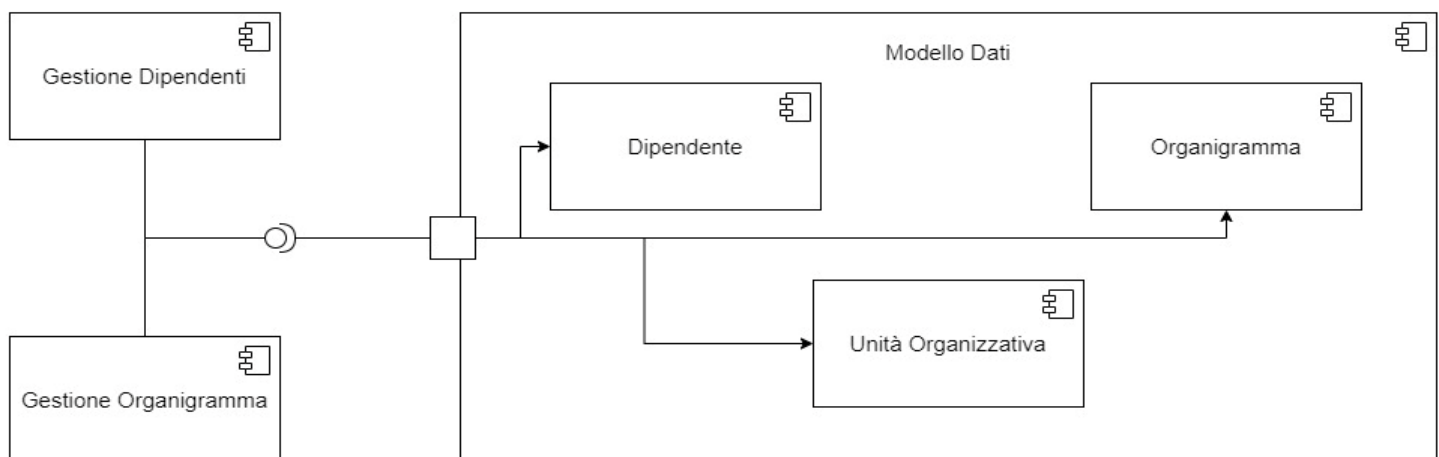






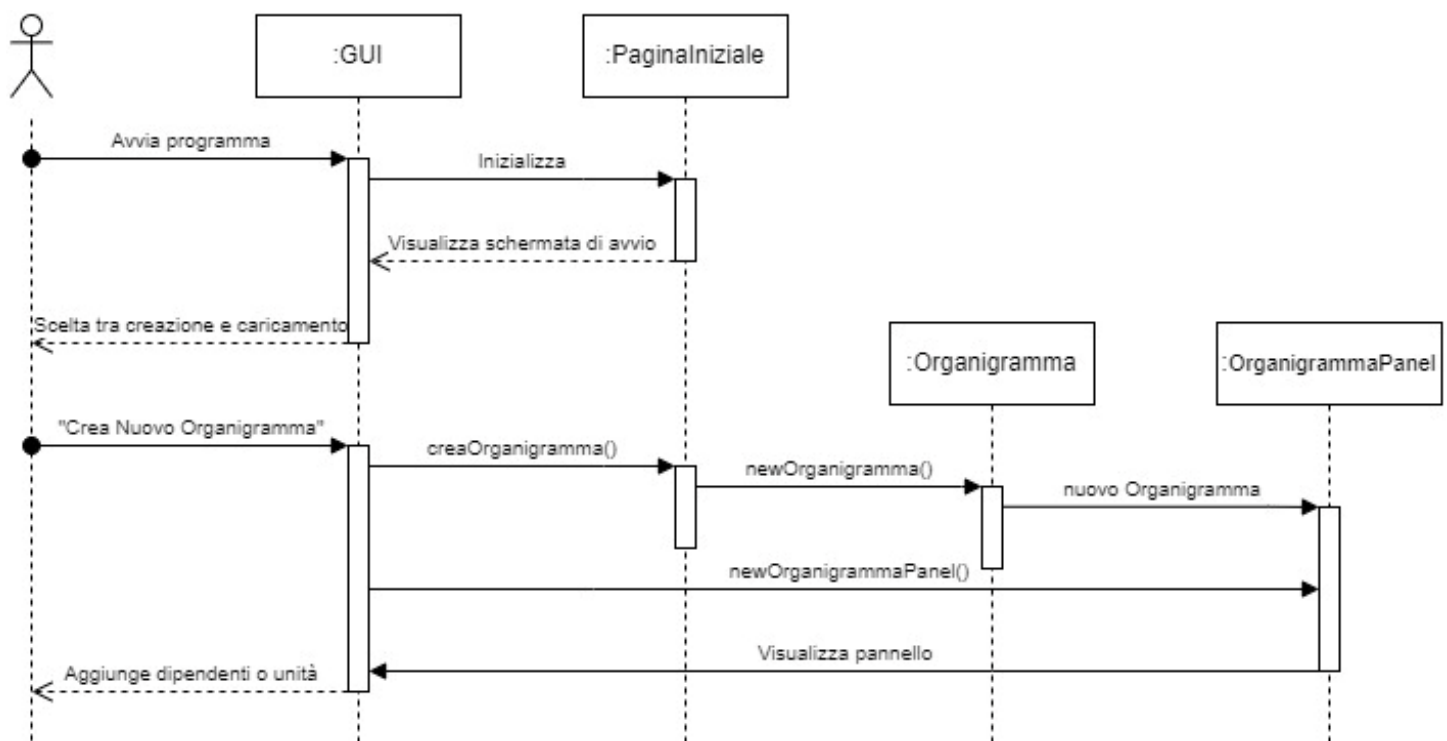
C. Architettura Software

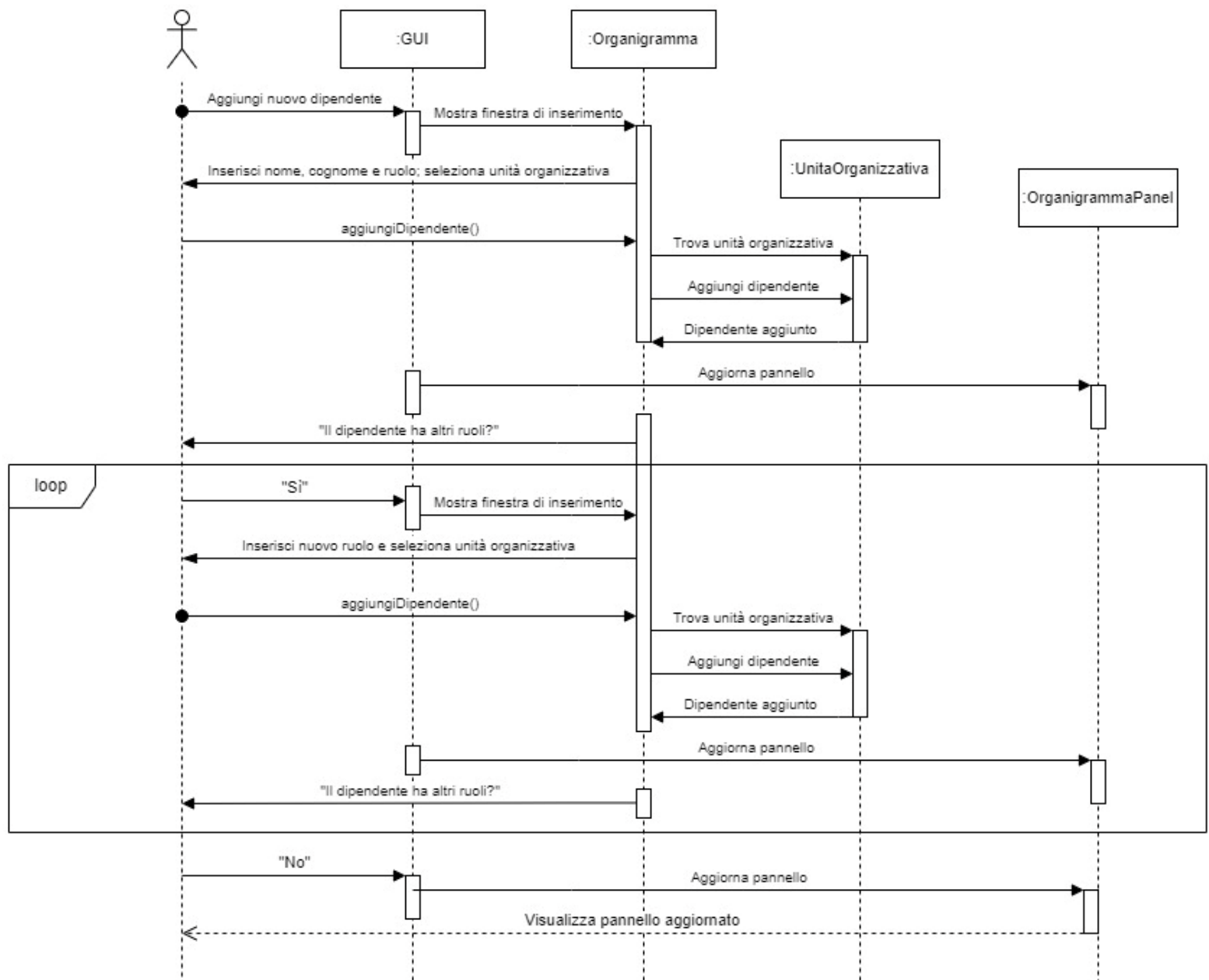
C.1 The static view of the system: Component Diagram



In questo esempio di Diagramma dei Componenti (Component Diagram), i componenti *Gestione Dipendenti* e *Gestione Organigramma* utilizzano le classi presenti nel componente *Modello Dati* per offrire una visione completa e aggiornata dell'organigramma aziendale. Grazie a queste classi, è possibile rappresentare in modo dettagliato ogni dipendente e la sua posizione all'interno dell'azienda.

C.2 The dynamic view of the software architecture: Sequence Diagram





Questi diagrammi rappresentano graficamente il flusso di interazioni tra gli attori e gli oggetti del sistema; in particolare l'avvio del programma, la creazione di un nuovo organigramma e l'aggiunta di nuovi dipendenti alle unità organizzative dell'organigramma.

D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)

Il sistema utilizza un file di testo come sorgente di dati principale, sia per il salvataggio che per il caricamento dell'organigramma aziendale. Le informazioni gestite includono:

- Unità Organizzativa: Nome e struttura delle unità organizzative aziendali.
- Dipendenti: Dettagli sui dipendenti, inclusi nome, cognome, ruolo e unità organizzativa associata.
- Ruoli: Ruoli dei dipendenti all'interno delle unità organizzative, con la possibilità di gestire più ruoli per dipendente (seppur attributi della classe Dipendenti, è importante rappresentarli correttamente nel modello dei dati).

I file possono essere aperti tramite editor di testo (quindi possono essere facilmente corrotti) e la loro forma è pressappoco la seguente:

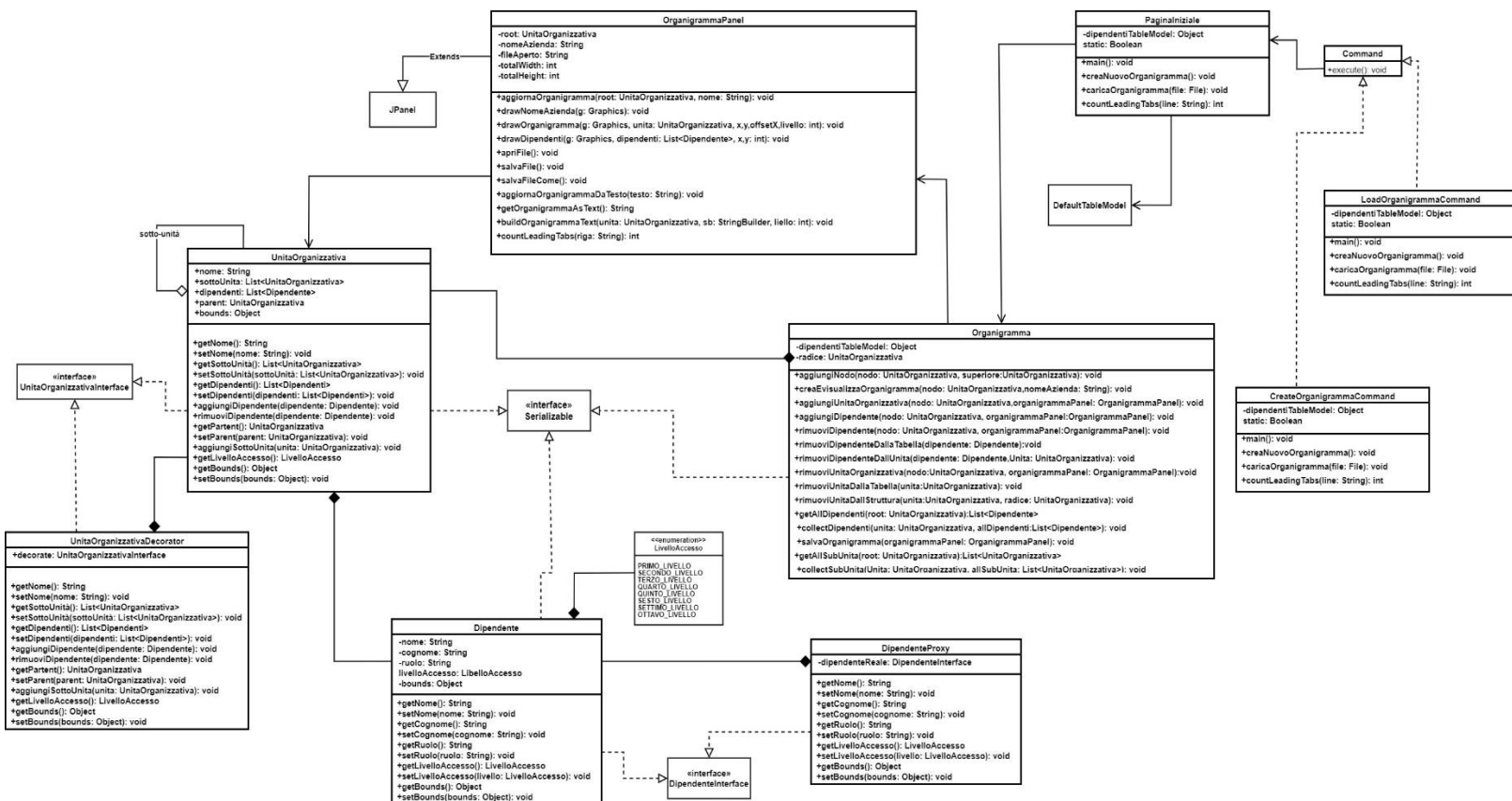
```
Organigramma Azienda: Azienda AAA

- Direzione Generale
* CEO: Mario Rossi
* CTO: Laura Bianchi
  - Dipartimento IT
  * Developer: Andrea Verdi
  * DevOps: Marco Gialli
    - Dipartimento HR
    * HR Manager: Elisa Neri
  - Dipartimento Finanza
  * CFO: Giovanni Blu
```

Ogni unità organizzativa è indicata e riconosciuta dal sistema grazie al trattino (“-“); ogni indentazione (“Tab” o “\t”) rappresenta la gerarchia tra unità organizzative e l’asterisco (“*”) sotto ogni unità organizzativa indica il ruolo, con rispettivo nome e cognome, del dipendente presente all’interno di quest’ultima.

E. Scelte Progettuali (Design Decisions)

Di seguito il Class Diagram, utile per mostrare i tipi degli oggetti che fanno parte del sistema e le varie tipologie di relazioni statiche tra essi:



Inoltre, l'applicazione è supportata da alcuni Design Pattern tra cui: Composite, Decorator, Proxy, Factory Method e Command.

La descrizione, l'implementazione e i benefici che apportano al sistema informativo verranno illustrati nel prossimo paragrafo.

F. Progettazione di Basso Livello

1. Design Pattern *Decorator* per estendere le capacità delle unità organizzative.

Il pattern Decorator consente di aggiungere nuove responsabilità ad un oggetto esistente senza alternare il comportamento degli altri oggetti della stessa classe. Per tale motivo, è stato applicato alla classe *UnitaOrganizzativa* permettendo all'utente di aggiungere nuove funzionalità (come la gestione di un'unità organizzativa con decorazioni aggiuntive) senza modificare direttamente l'intera classe.

Benefici relativi all'utilizzo del pattern Decorator:

- Permette di estendere le funzionalità in modo flessibile.
- Mantiene il codice più modulare e riutilizzabile.
- Evita l'ereditarietà e la complessità di classi sovraccariche.

2. Design Pattern *Proxy* per gestire l'accesso ai dettagli dei dipendenti.

Il pattern Proxy favorisce una rappresentazione sostitutiva di un oggetto per controllarne gli accessi. Perciò, nel sistema, è stata creata una versione *DipendenteProxy* per controllare l'accesso ai dettagli dei dipendenti, permettendo la gestione delle operazioni effettuate sui dipendenti in modo più sicuro e controllato.

Benefici relativi all'utilizzo del pattern Proxy:

- Controlla e gestisce l'accesso a un oggetto.
- Può essere usato per implementare funzionalità come il caching, la protezione e la lazy initialization.
- Mantiene la separazione delle responsabilità.

3. Design Pattern *Command* per separare le operazioni eseguite da bottoni dalla logica concreta delle operazioni.

Il pattern Command trasforma richieste o operazioni in oggetti, consentendo di parametrizzare le operazioni, ritentare le operazioni, e supportare la cancellazione. Ogni comando è rappresentato come un oggetto che incapsula la richiesta e i dettagli per eseguirla. Grazie a queste sue caratteristiche, si è deciso di implementare il pattern Command nella classe *PaginaIniziale*: i comandi concreti *CreateOrganigrammaCommand* e *LoadOrganigrammaCommand* incapsulano le operazioni di creazione e caricamento dell'organigramma, consentendo di eseguire tali operazioni tramite il pattern Command.

Benefici relativi all'utilizzo del pattern Command:

- Separa il mittente di una richiesta dal destinatario.
- Fornisce supporto per le operazioni di annullamento/ripetizione.
- Facilita la registrazione e la gestione delle transazioni.

4. Design Pattern *Composite* per rappresentare e gestire le unità organizzative e le loro sotto-unità in modo coerente

Il pattern Composite consente di trattare oggetti singoli e composizioni di oggetti in modo uniforme, utile per rappresentare strutture ad albero (come le gerarchie di directory o gli organigrammi aziendali), in cui i nodi possono essere sia oggetti foglia (singoli) che composizioni di oggetti. Perciò, nella classe *UnitaOrganizzativa*, il pattern Composite è evidente nella struttura in cui ogni unità organizzativa può contenere altre unità organizzative (sotto-unità): questo consente di trattare un'unità e le sue sotto-unità come un'unica struttura gerarchica, semplificando la manipolazione e la visualizzazione dell'organigramma.

Benefici relativi all'utilizzo del pattern Composite:

- Permette di trattare gli oggetti singoli e le composizioni di oggetti in modo uniforme, semplificando il codice che gestisce queste strutture.
- Facilita l'aggiunta di nuovi tipi di nodi (sia foglie che composizioni) senza modificare il codice esistente.
- Riduce la complessità del codice, poiché il codice client può utilizzare un'interfaccia comune per interagire con gli oggetti composti e i loro componenti.

5. Design Pattern *Factory Method* per gestire la creazione e la configurazione degli oggetti Organigramma.

Il pattern Factory Method consente di delegare la creazione di oggetti a sottoclassi. Nella classe *PaginaIniziale*, il pattern Factory Method viene utilizzato per creare e visualizzare l'organigramma: quando viene scelta l'opzione per creare un nuovo organigramma o caricare uno esistente, il metodo Factory è responsabile per istanziare e configurare l'oggetto Organigramma appropriato. Ad esempio, i metodi *creaNuovoOrganigramma()* e *caricaOrganigramma(File file)* utilizzano la factory method per creare un'istanza di Organigramma e configurarla con i dati necessari.

Benefici relativi all'utilizzo del pattern Factory Method:

- Permette di separare la creazione degli oggetti dal loro utilizzo, semplificando il codice e migliorando la manutenibilità.
- Facilita l'aggiunta di nuovi tipi di oggetti senza modificare il codice esistente, poiché le sottoclassi possono definire le loro specifiche implementazioni.
- Consente un controllo centralizzato sul processo di creazione degli oggetti, garantendo che gli oggetti vengano creati e configurati in modo coerente.

G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs)

Requisiti funzionali (FRs):

1. Creazione e Aggiornamento dell'Organigramma:

La classe ***Organigramma*** gestisce la struttura complessiva dell'organigramma, permettendo la creazione e l'aggiornamento delle unità organizzative. Le modifiche alle unità e le operazioni di aggiunta e rimozione sono effettuate tramite metodi specifici in questa classe. La classe ***UnitaOrganizzativa*** rappresenta le singole unità all'interno dell'organigramma e viene utilizzata per gestire le relazioni tra le unità. Le operazioni di modifica sono realizzate tramite metodi che aggiornano le strutture dati interne della classe ***Organigramma***.

2. Gestione dei Dipendenti:

La classe ***Dipendente*** gestisce i dati dei dipendenti, inclusi nome, cognome e ruolo. La gestione dei dipendenti all'interno delle unità organizzative è effettuata tramite metodi nella classe ***Organigramma***, che associa i dipendenti alle rispettive unità, che permettono all'utente di aggiungere e rimuovere dipendenti dalle unità organizzative.

3. Gestione delle Relazioni Gerarchiche:

Le relazioni gerarchiche tra le unità organizzative sono gestite tramite la classe ***UnitaOrganizzativa***. Ogni unità può avere una o più sotto-unità, e le relazioni vengono aggiornate tramite metodi nella classe ***Organigramma***. Inoltre, la classe ***UnitaOrganizzativa*** contiene metodi per aggiungere e rimuovere sotto-unità.

4. Salvataggio e Caricamento dell'Organigramma:

La classe ***OrganigrammaPanel*** è responsabile del salvataggio e del caricamento dei dati dell'organigramma. I metodi di salvataggio e caricamento utilizzano file.txt per memorizzare e recuperare le informazioni.

5. Visualizzazione dell'Organigramma

La classe ***OrganigrammaPanel*** è utilizzata per la visualizzazione grafica dell'organigramma. Questa classe rappresenta su schermo l'organigramma, permettendo agli utenti di vedere la struttura completa.

Requisiti Non Funzionali (NFRs):1. Usabilità Interfaccia Utente:

Le classi ***PaginaIniziale*** fornisce un'interfaccia utente iniziale che guida l'utente attraverso le operazioni principali dell'applicazione. L'interfaccia è progettata per essere intuitiva e facile da usare, migliorando l'esperienza utente complessiva.

2. Scalabilità:

L'architettura del progetto utilizza classi che possono gestire dinamicamente le unità organizzative e i dipendenti. Le strutture dati e i metodi sono progettati per scalare con l'aumento del numero di unità e dipendenti senza compromettere le prestazioni.

3. Accessibilità:

L'interfaccia grafica, inclusa la classe ***OrganigrammaPanel***, è progettata per essere accessibile, con attenzione alle risoluzioni diverse e alle modalità di visualizzazione.

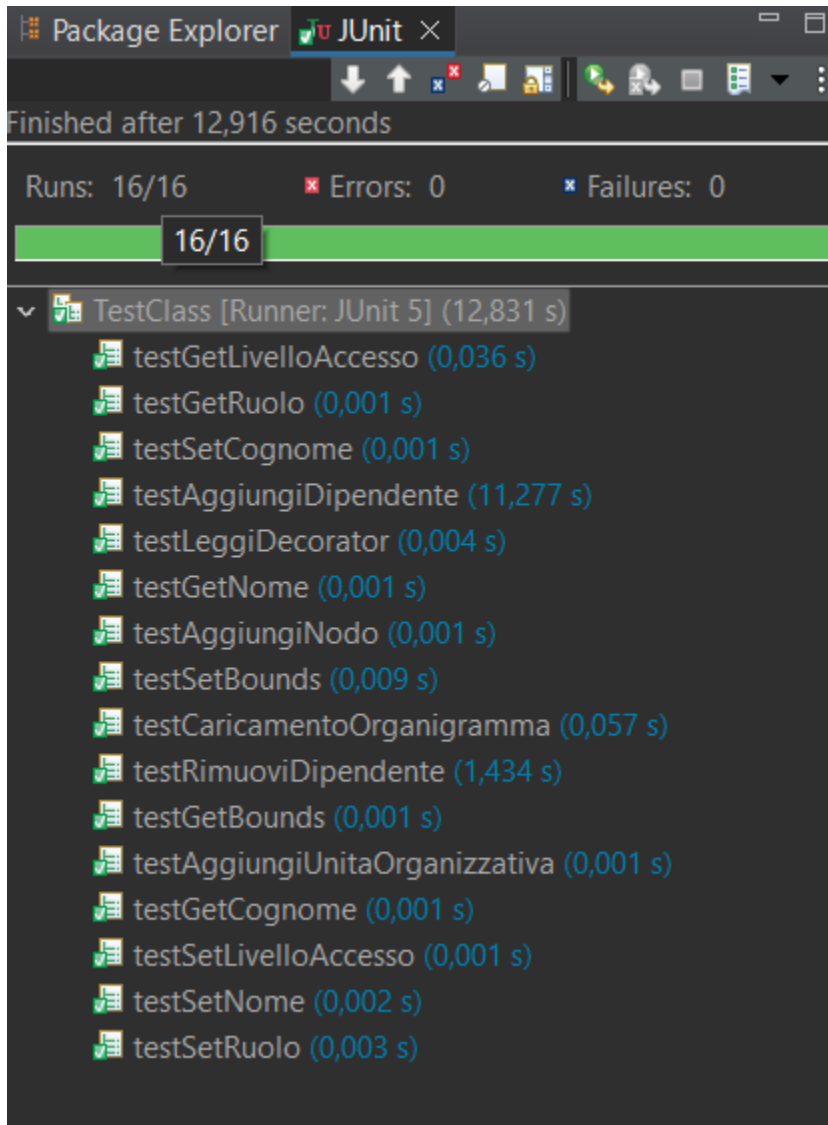
4. Performante:

La progettazione delle classi e dei metodi è ottimizzata per garantire buone prestazioni anche con un gran numero di unità organizzative e dipendenti. Le operazioni critiche sono ottimizzate per minimizzare i tempi di risposta e garantire un'esperienza utente fluida.

Appendix. Prototype

Per verificare la robustezza dell'applicazione, è necessario testarla sottoponendola ad alcuni test strutturati appositamente utilizzando il framework JUnit.

Per far ciò si crea una classe test (**TestClass**) e, come si può vedere nell'immagine qui sotto, tutti i test sono andati a buon fine.



Nello specifico:

```
@Before
public void setUp() {
    radice = new UnitaOrganizzativa("Radice");
    dipendentiTableModel = new DefaultTableModel(new String[]{"Nome", "Ruolo"}, 0);
    organigramma = new Organigramma(radice, dipendentiTableModel);
}

@Test
public void testAggiungiNodo() {
    UnitaOrganizzativa unital = new UnitaOrganizzativa("Unità 1");
    organigramma.aggiungiNodo(unital, radice);

    assertTrue(radice.getSottoUnità().contains(unital));
}

@Test
public void testAggiungiUnitaOrganizzativa() {
    UnitaOrganizzativa unital = new UnitaOrganizzativa("Unità 1");
    organigramma.aggiungiNodo(unital, radice);

    assertEquals(1, radice.getSottoUnità().size());
    assertEquals("Unità 1", radice.getSottoUnità().get(0).getNome());
}
```

Il metodo *setUp()* serve per creare un ambiente per l'esecuzione dei test.

```
@Test
public void testLeggiDecorator() {
    UnitaOrganizzativa baseUnità = new UnitaOrganizzativa("Unità Base");
    UnitaOrganizzativaInterface decorata = new LeggiUnitaOrganizzativaDecorator(baseUnità);

    Dipendente dipendente = new Dipendente("Mario", "Rossi", "Manager", LivelloAccesso.PRIMO_LIVELLO);
    decorata.aggiungiDipendente(dipendente);

    // Verifica che il dipendente sia stato aggiunto
    assertTrue(decorata.getDipendenti().contains(dipendente));
}
```

Un *@Test* per valutare il comportamento del pattern Decorator.

```
@Test
public void testAggiungiDipendente() {
    livello=LivelloAccesso.PRIMO_LIVELLO;
    UnitaOrganizzativa unital = new UnitaOrganizzativa("Unità 1");
    organigramma.aggiungiNodo(unital, radice);

    Dipendente dipendente = new Dipendente("Mario", "Rossi", "Manager", livello);
    unital.aggiungiDipendente(dipendente);
    organigramma.aggiungiDipendente(radice, new OrganigrammaPanel(radice, "Azienda Test"));

    List<Dipendente> dipendenti = organigramma.getAllDipendenti(radice);
    assertTrue(dipendenti.contains(dipendente));
    assertEquals(1, dipendentiTableModel.getRowCount());
    assertEquals("Mario Rossi", dipendentiTableModel.getValueAt(0, 0));
    assertEquals("Manager", dipendentiTableModel.getValueAt(0, 1));
}

@Test
public void testRimuoviDipendente() {
    UnitaOrganizzativa unital = new UnitaOrganizzativa("Unità 1");
    organigramma.aggiungiNodo(unital, radice);

    Dipendente dipendente = new Dipendente("Mario", "Rossi", "Manager", livello);
    unital.aggiungiDipendente(dipendente);
    organigramma.rimuoviDipendente(radice, new OrganigrammaPanel(radice, "Azienda Test"));

    List<Dipendente> dipendenti = organigramma.getAllDipendenti(radice);
    assertFalse(dipendenti.contains(dipendente));
    assertEquals(0, dipendentiTableModel.getRowCount());
}
```

Questi sono i metodi relativi all’inserimento e alla rimozione di un dipendente (aiutano a capire se il dipendente viene aggiunto e/o rimosso anche dalla tabella).

```
private static final long TIME_LIMIT_MS = 3000; //3 secondi
private static File tempFile;

@Before
public void setUp() throws IOException {
    StringBuilder content = new StringBuilder();
    content.append("Organigramma Azienda: Test Company\n");
    for (int i = 0; i < 10; i++) {
        content.append("- Unit ").append(i).append("\n");
        for (int j = 0; j < 10; j++) {
            content.append("\t* Ruolo ").append(j).append(": Dipendente ")
                .append(i).append(" ").append(j).append("\n");
        }
    }
    tempFile = Files.createTempFile("organigramma", ".txt").ToFile();
    Files.write(tempFile.toPath(), content.toString().getBytes());
}

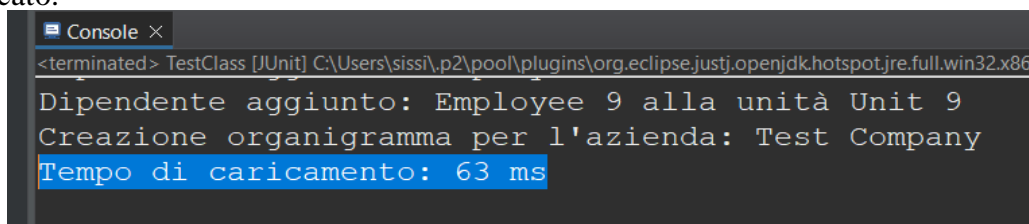
@After
public void tearDown() {
    if (tempFile != null && tempFile.exists()) {
        tempFile.delete();
    }
}

@Test
public void testCaricamentoOrganigramma() {
    long startTime = System.currentTimeMillis();
    PaginaIniziale.caricaOrganigramma(tempFile);

    long endTime = System.currentTimeMillis();
    long duration = endTime - startTime;

    System.out.println("Tempo di caricamento: " + duration + " ms");
    assertTrue(duration <= TIME_LIMIT_MS, "Il caricamento dell'organigramma "
        + "ha superato il tempo limite di " + TIME_LIMIT_MS + " ms");
}
```

Mentre, per valutare la performance del sistema, si è utilizzato un nuovo `setUp()` per poter creare un file di testo di grandi dimensioni (subito dopo il test verrà eliminato (`@After`)). Si implementa, dunque, il metodo `testCaricamentoOrganigramma()` che serve a valutare sia se l'organigramma viene avviato entro un certo tempo accettabile, sia ci permette di visualizzare effettivamente quanto tempo impiega il file.txt ad essere caricato.



The screenshot shows a console window titled "Console X" with the following output:

```
<terminated> TestClass [JUnit] C:\Users\sissi\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\java.exe
Dipendente aggiunto: Employee 9 alla unità Unit 9
Creazione organigramma per l'azienda: Test Company
Tempo di caricamento: 63 ms
```

```
@Before
public void setUp3() {
    dipendenteReale = new Dipendente("Mario", "Rossi", "Manager",
        LivelloAccesso.PRIMO_LIVELLO);
    dipendenteProxy = new DipendenteProxy(dipendenteReale);
}

@Test
public void testGetNome() {assertEquals("Mario", dipendenteProxy.getNome());}

@Test
public void testSetNome() {
    dipendenteProxy.setNome("Luigi");
    assertEquals("Luigi", dipendenteReale.getNome());
}

@Test
public void testGetCognome() {assertEquals("Rossi", dipendenteProxy.getCognome());}

@Test
public void testSetCognome() {
    dipendenteProxy.setCognome("Verdi");
    assertEquals("Verdi", dipendenteReale.getCognome());
}

@Test
public void testGetRuolo() {assertEquals("Manager", dipendenteProxy.getRuolo());}

@Test
public void testSetRuolo() {
    dipendenteProxy.setRuolo("Team Leader");
    assertEquals("Team Leader", dipendenteReale.getRuolo());
}

@Test
public void testGetLivelloAccesso() {assertEquals(LivelloAccesso.PRIMO_LIVELLO,
    dipendenteProxy.getLivelloAccesso());}

@Test
public void testSetLivelloAccesso() {
    dipendenteProxy.setLivelloAccesso(LivelloAccesso.SECONDO_LIVELLO);
    assertEquals(LivelloAccesso.SECONDO_LIVELLO, dipendenteReale.getLivelloAccesso());
}

@Test
public void testGetBounds() {assertNotNull(dipendenteProxy.getBounds());}

@Test
public void testSetBounds() {
    Rectangle newBounds = new Rectangle(10, 10, 100, 100);
    dipendenteProxy.setBounds(newBounds);
    assertEquals(newBounds, dipendenteReale.getBounds());
}
```

Il metodo *setUp3()* crea un oggetto Dipendente reale e il suo proxy