UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e implementazione di un sistema per la gestione di un orario per una Scuola Media

Design and implementation of a timetable management system for a Middle School

Relatore

Prof. Domenico Ursino

Candidato

Samuele Di Summa

ANNO ACCADEMICO 2024-2025

Dall'informatica abbiamo ereditato un approccio, l'approccio che le cose possono essere scomposte in maniera logica, ma allo stesso tempo l'approccio che le cose non possono essere ridotte alla sola logica.

Salvatore Sanfilippo

Sommario

L'organizzazione degli istituti scolastici di qualsiasi grado di istruzione è un aspetto importante che coinvolge la vita lavorativa delle diverse figure di sistema. Una buona gestione e organizzazione del tempo scolastico, resa possibile in buona parte da un orario scolastico ben strutturato, permette al personale docente di vivere l'esperienza scolastica nel modo più professionale. Questa tesi si pone l'obiettivo di descrivere la realizzazione di un software per la creazione di un orario scolastico di una scuola media, illustrandone le varie fasi, dall'analisi alla progettazione, fino alla sua implementazione. I vincoli e le necessità personali dei docenti mettono alla prova la realizzazione di un orario ottimale, rendendolo uno dei problemi più classici nel mondo della gestione delle risorse umane (Timetabling Problem).

Keyword: Python, PyQt, QtDesigner, pattern MVC, Linear Programming, Timetabling Problem, software gestionale, algoritmo di ordinamento.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Da tempo viene utilizzata la tecnologia per migliorare molti aspetti del lavoro aziendale. In particolare, una scuola ha la necessità di strutturare un buon orario scolastico in modo da favorire un corretto apprendimento dei ragazzi. Prima degli anni '90 i dirigenti costruivano gli orari manualmente su fogli cartacei; con la diffusione dei computer nelle scuole avviene una digitalizzazione progressiva, portando i docenti a utilizzare fogli di calcolo come Excel. Dagli anni 2000 in poi, gli strumenti digitali si fanno sempre più completi; cominciano ad apparire nel mercato i primi software dedicati al problema dell'orario scolastico.
Questa tesi ha l'obiettivo di descrivere un percorso fatto per lo sviluppo di un software che rispondesse a tali esigenze. I motivi per cui, per una scuola, è necessario un software di questa tipologia sono diversi; i docenti lamentano spesso ore di lezione distanti temporalmente tra loro, portandoli a non ottimizzare la loro giornata lavorativa. Al tempo stesso, gli studenti trovano difficoltà nell'apprendimento se le ore più impegnative sono concentrate in un unico giorno. Oltre a queste problematiche, le scuole impiegano generalmente settimane per strutturare un orario scolastico che soddisfi le esigenze dei docenti; inoltre, molte scelte che vengono prese sono del tutto arbitrarie e non seguono una logica di ottimizzazione, portando non pochi disappunti tra i docenti. Ad esempio, i docenti che insegnano materie non di laboratorio preferiscono ottenere le prime ore della giornata, approfittando della buona attenzione dei ragazzi non condizionata dalla lunga mattinata.
Il software sviluppato e discusso in questa tesi si inserisce nel panorama dei software per la gestione scolastica, rispondendo alle esigenze di una scuola secondaria di primo grado; tuttavia, rappresenta una soluzione modesta, mirata a un caso specifico. Negli ultimi anni, il progresso informatico nell'ambito dell'Intelligenza Artificiale ha portato alla comparsa di diversi software AI che generano automaticamente l'orario scolastico in pochi minuti. Il software discusso in questo elaborato si colloca come una soluzione mirata e semplificata, pensata per rispondere a esigenze specifiche senza ricorrere a sistemi complessi di ottimizzazione o Intelligenza Artificiale.
Questa tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 viene introdotto il problema dell'orario scolastico, approfondendo la famiglia di problemi di cui fa parte; inoltre, vengono discusse le possibili soluzioni che si adottano in tale scenario.
- Nel Capitolo 2 si analizzano le esigenze della scuola, raccogliendo, così, i requisiti; inoltre, vengono illustrati i casi d'uso e un glossario che consente una migliore comprensione di determinate parole.

- Nel Capitolo 3 viene trattato il flusso di lavoro dell'analisi, identificando le entità principali del software e descrivendole grazie ai diagrammi UML.
- Nel Capitolo 4 si descrive la fase di progettazione, mostrando, nel dettaglio, i diagrammi delle classi; inoltre, viene illustrato uno schema di funzionamento per l'algoritmo dell'orario scolastico.
- Nel Capitolo 5 viene trattata la fase di implementazione, esponendo i concetti chiave che hanno portato allo sviluppo del software. Successivamente, viene riportato il manuale utente, ossia una sezione che suggerisce al lettore il corretto utilizzo del programma.
- Il Capitolo 6 è dedicato alle conclusioni tratte dal lavoro svolto; vengono descritte le difficoltà riscontrate nel percorso e vengono riportati alcuni miglioramenti futuri che possono essere apportati al software.

CAPITOLO 1

Descrizione del contesto

In questo capitolo introdurremo il problema dell'orario scolastico nel caso di una scuola media. Introdurremo inoltre, dei concetti di programmazione lineare intera individuando il problema nel concetto di "timetabling problem"; nell'ultimo paragrafo sarà, invece, discussa la soluzione adottata in questo elaborato.

1.1 Complessità dell'organizzazione di una scuola secondaria di primo grado

L'organizzazione dell'istituto è abbastanza complessa e deve rendere conto di diversi aspetti. La scuola è composta da due edifici distanti 5 km tra loro ed è fornita attualmente di 35 docenti che insegnano in 21 classi da 3 sezioni l'una, un numero che può variare in base al numero di iscrizioni di nuovi studenti. I docenti hanno esigenze diverse, possono lavorare in più scuole e potrebbe essere impossibile per loro lavorare in determinate ore di uno o più giorni. Inoltre, la presenza di due edifici potrebbe complicare gli spostamenti dei docenti, poiché sarebbe improponibile per loro spostarsi ripetutamente da un edificio all'altro. I professori, ovviamente, possono insegnare non solo in un'altra scuola, ma anche in più classi; inoltre, alcuni hanno delle lezioni combinate, ovvero una classe può sostenere due lezioni da docenti diversi contemporaneamente, come le lezioni di francese che vanno di pari passo con quelle di spagnolo. Il numero di ore che i docenti devono sostenere è composto da un massimo di 18 ore di lezione con eventualmente 4 ore di potenziamento. Esistono, poi, diversi criteri da rispettare secondo l'art. 28 del CCNL, che stabiliscono il numero di ore di impiego dei docenti, la gestione dei giorni liberi e delle ore buca, l'aggiunta di eventuali ore di supplenza e ulteriori direttive che garantiscono una maggiore flessibilità nel piano settimanale lavorativo.

1.2 L'importanza di un buon orario scolastico

Un buon orario scolastico si formula rispettando i vincoli descritti nel paragrafo precedente, soddisfacendo egualmente i docenti. La buona realizzazione dell'orario permette ai docenti di evitare troppi spostamenti tra i dipartimenti, nonché di poter lavorare in orari più comodi incastrando, eventualmente, le ore di lavoro svolte in altri istituti. Gli studenti possono avere giornate equilibrate con pochi giorni di carico di lavoro elevato; si noti, infatti,

che non tutte le materie sono uguali; alcune possono essere più impegnative di altre, specialmente se si parla di lezioni da due ore. Un buon orario tiene conto di vincoli strutturali, come l'esistenza di professori part time o la necessità di usare spazi scolastici, come la palestra per educazione fisica o i laboratori. Per una soluzione ottimale ci sono da considerare anche vincoli educativi; esempi sono la ricerca di un equilibrio tra materie pratiche e teoriche, una buona alternanza di lezioni da un'ora con lezioni da due ore e una distribuzione delle lezioni tale da permettere agli studenti di consolidare le nozioni apprese durante la lezione.

1.3 Un problema di PLI

Il termine "ottimizzare" viene comunemente utilizzato per esprimere la volontà di scegliere l'opzione migliore tra diverse alternative possibili. L'ottimizzazione è la scienza che si occupa di formulare modelli utili nelle applicazioni e di inventare metodi efficienti per identificare la soluzione ottimale. Nel tentativo di ottimizzare un problema complesso con diverse variabili, sono state definite quattro fasi di sviluppo principali:

- *Identificazione del problema:* per risolvere il problema in modo adeguato, è necessario individuare le variabili in gioco, ciò che le lega tra loro, le criticità e l'obiettivo da raggiungere.
- *Formulazione del modello:* Una volta fatte tali considerazioni, si decide il modello matematico più idoneo tenendo conto di diversi criteri: relazioni matematiche, parametri numerici, l'orizzonte temporale, variabili di decisione e il loro dominio.
- *Sviluppo degli algoritmi:* Dopo aver deciso il tipo di modello da adottare, si procede allo sviluppo di un algoritmo che soddisfi le richieste, possibilmente in tempi adeguati.
- *Realizzazione e collaudo:* Si realizza e collauda l'algoritmo, analizzando la consistenza dei risultati e la loro stabilità.

1.3.1 Timetabling problem

Con il termine "Timetabling" si fa riferimento a una tipologia di problemi, modellabili tramite Programmazione Lineare Intera (PLI), che riguardano la gestione delle risorse umane di una azienda e l'assegnazione di un certo lavoro a un certo dipendente. Nel nostro caso possiamo individuare nella creazione dell'orario scolastico la necessità di assegnare una determinata lezione di una classe a un docente in una data ora della settimana.

1.3.2 Vincoli forti e vincoli deboli

Lo sviluppo di un algoritmo efficiente per la creazione di un orario è spesso complicato, questo perché la natura del problema viene classificata come NP (non polinomiale). In altre parole, verificare che un orario sia ottimo potrebbe richiedere dei tempi estremamente lunghi; ciò è dovuto in parte alla presenza di vincoli forti e vincoli deboli. Per vincoli forti intendiamo quelle richieste che devono essere necessariamente esaudite, come il fatto che una classe non può avere due lezioni diverse alla stessa ora, oppure i professori, per motivi legali, non possono lavorare tutto il giorno senza ore di buca. I vincoli deboli sono richieste preferibili e negoziabili, ma ignorare il rispetto di tali vincoli non incide sul raggiungimento di un orario accettabile; un esempio sono le preferenze personali dei docenti non di tipo logistico.

1.3.3 Funzione obiettivo

Quando si prendono decisioni, ci viene naturale individuare un parametro su cui basare la nostra strategia; ad esempio, ci può capitare di preparare uno zaino e potremmo ottimizzare lo spazio scegliendo gli oggetti in base al loro possibile utilizzo. Allo stesso modo, un'azienda che produce giocattoli potrebbe decidere di costruire una certa tipologia per minimizzare lo spreco di risorse e massimizzare i guadagni. La funzione obiettivo esprime uno o più valori che, massimizzandoli o minimizzandoli, ci porta al risultato ottimo. Nel nostro caso è difficile valutare quale possa essere il parametro da portare all'estremo, questo perché siamo soddisfatti se l'algoritmo riesce a darci un orario che rispetti tutti i vincoli. In ogni caso bisogna sottolineare il fatto che esistono i vincoli deboli che possono determinare dei risultati più preferibili rispetto ad altri, come il numero di spostamenti che un docente deve sostenere durante la settimana.

1.4 Una soluzione euristica

Nei paragrafi precedenti è stato sottolineato come il timetabling sia un problema risolvibile sfruttando la teoria dell'ottimizzazione, identificabile in una categoria di problemi e adottando concetti di Programmazione Lineare Intera. Il problema è noto per essere di tipo NP-hard e le soluzioni per lo sviluppo di un algoritmo trovate nel tempo hanno avuto approcci diversi, come l'utilizzo di Intelligenza Artificiale e metodi come il Branch and Bound. L'algoritmo adottato in questo elaborato può essere considerato euristico, poiché l'orario viene mediamente generato in tempi accettabili (sotto al minuto), fallisce in rari casi e la sua semplicità, rapportandola alle soluzioni che offre, soddisfa le nostre esigenze.

CAPITOLO 2

Specifica dei requisiti

In questo capitolo viene trattata l'analisi dei requisiti, un passo fondamentale per il successo di qualsiasi progetto di sviluppo software. Durante le fasi iniziali dell'analisi di sistemi complessi, come aziende e istituti, è richiesta una visione di insieme sulle principali funzionalità che l'utente deve poter eseguire tramite il software; un'imprecisa descrizione di queste specifiche può compromettere lo sviluppo del progetto nelle fasi successive, aumentando costi e dilungando i tempi di sviluppo. Il capitolo comprende, inoltre, un paragrafo dedicato al glossario e uno ai casi d'uso; il primo riporta una lista di definizioni che aiuta a evidenziare le parole chiave del mondo della scuola e dello sviluppo software, mentre il secondo riguarda uno strumento adottato per descrivere meglio le interazioni tra utente e sistema.

2.1 Glossario

La Tabella 2.1 riporta un elenco di termini specifici utilizzati in questo elaborato, consentendo una comprensione di concetti fondamentali che riguardano lo sviluppo software e l'ambiente scolastico.

2.2 Requisiti di sistema

I requisiti di sistema sono specifiche di alto livello che il sistema dovrebbe essere in grado di svolgere; esempi sono la gestione dei dati delle entità del programma, oppure caratteristiche desiderate dal cliente, come efficienza e prestazioni elevate. Per individuare le specifiche, sono state svolte diverse interviste con alcuni collaboratori scolastici, grazie alle quali sono emerse le principali esigenze che il software deve soddisfare. I temi principalmente discussi hanno riguardato le regole da rispettare per la creazione dell'orario, la gestione delle informazioni dei professori e la capacità del programma di essere comprensibile nel suo utilizzo. La comunicazione tra sviluppatore e cliente è cruciale, poiché specifiche inesatte o irrealizzabili rischiano di complicare le fasi successive della progettazione. I requisiti che sono trattati in questo elaborato possono essere classificati in due tipologie: requisiti funzionali e requisiti non funzionali.

Termine	Descrizione
Affidabilità	La capacità di un prodotto o sistema di eseguire le funzionalità specificate sotto prefissate condizioni, per uno specifico periodo di tempo o per un numero specificato di transazioni.
Collaboratore scolastico	Membro del personale amministrativo,tecnico e ausiliario, è il personale non docente che lavora nella scuola italiana.
Graphical User Interface	Interfaccia che permette una interazione visiva con un componente o sistema informatico.
Lezioni gemellate	Lezioni condivise da due classi alla stessa ora.
Orario scolastico	Tabella che rappresenta l'assegnazione delle lezioni dei docenti nella settimana.
Ore di buca	Ore scolastiche in cui il docente non insegna.
Software gestionale	Un insieme di programmi che automatizzano i processi di gestione all'interno di organizzazioni.
Python	Linguaggio di programmazione di alto livello.
Unifield Modeling Language	Linguaggio di modellazione basato sul paradigma orientato agli oggetti.
Vincolo	Condizione da rispettare che restringe i risultati accettabili.

Tabella 2.1: Glossario dei termini relativi al contesto di interesse

2.2.1 Requisiti funzionali

I requisiti funzionali evidenziano in modo diretto le principali funzionalità del software, fornendo informazioni su come l'utente tratterà le entità nel programma. Nella Tabella 2.2 sono elencati tutti i requisiti funzionali; una parte significativa di essi è stata individuata intervistando un collaboratore del dirigente scolastico.

Requisito	Descrizione
Cerca professore	Il sistema dovrà gestire la ricerca di un professore
Elimina professore	Il sistema dovrà gestire l'eliminazione di un professore
Modifica professore	Il sistema dovrà gestire le modifiche ai dati di un professore esistente
Visualizza professore	Il sistema dovrà gestire la visualizzazione dei professori con i loro dati
Inserisci professore	Il sistema dovrà gestire la creazione del profilo di nuovi professori
Crea scuola	Il sistema dovrà gestire la creazione di una nuova scuola
Modifica scuola	Il sistema dovrà gestire le modifiche ai dati di una scuola
Visualizza scuola	Il sistema dovrà gestire la visualizzazione dei dati di una scuola
Elimina scuola	Il sistema dovrà gestire l'eliminazione di una scuola
Salva dati scuola	Il sistema dovrà gestire il salvataggio dei dati di una scuola
Genera orario	Il sistema dovrà gestire la generazione di un orario nuovo o parzialmente completo
Elimina orario	Il sistema dovrà gestire l'eliminazione di un orario esistente
Modifica orario	Il sistema dovrà gestire le modifiche alle ore che compongono un orario
Visualizza orario	Il sistema dovrà gestire la visualizzazione di un orario
Pdf orario	Il sistema dovrà gestire la creazione e il salvataggio di un orario in formato PDF

Tabella 2.2: Requisiti funzionali

2.2.2 Requisiti non funzionali

I requisiti non funzionali riguardano le specifiche del programma in modo più indiretto, concentrandosi su come il software dovrebbe comportarsi, piuttosto che su cosa dovrebbe fare, ed esprimono i vincoli progettuali che gli sviluppatori devono tenere in considerazione. Nella Tabella 2.3 è possibile visualizzare questi particolari requisiti, individuati puntando l'attenzione sui seguenti aspetti: affidabilità, prestazioni e semplicità di implementazione del programma.

Requisito	Descrizione
Python	Il sistema dovrà essere implementato in Python
G.U.I.	Il sistema dovrà permettere all'utente di interagire tramite una interfaccia grafica
Login	Il sistema dovrà gestire la creazione di credenziali per l'utente e il loro utilizzo per l'accesso
Codici univoci	Il sistema dovrà etichettare ogni entità in maniera univoca
Database	Il sistema dovrà gestire il salvataggio e il recupero dei dati tramite un database

Tabella 2.3: Requisiti non funzionali

2.3 Diagramma dei casi d'uso

L'analisi dei casi d'uso è uno strumento generalmente impiegato durante le fasi iniziali di sviluppo ed è utile a fornire una descrizione più accurata dei requisiti, mettendo in luce i modi in cui il sistema potrà essere utilizzato. Un caso d'uso può essere descritto come la specifica di una sequenza di azioni, incluse eventuali sequenze alternative e sequenze di errore, che un sistema, un sottosistema o una classe può eseguire interagendo con attori esterni. La struttura di questo strumento è ben definita da un modulo che, grazie ad informazioni chiave, permette una descrizione accurata di come il sistema si comporta senza entrare eccessivamente nei dettagli; tale modulo ha le seguenti informazioni: codice univoco, una breve descrizione, attori coinvolti, condizioni dello stato del sistema prima e dopo l'esecuzione del caso d'uso, passaggi chiave dello scenario e, infine, una sequenza degli eventi alternativi. Una delle informazioni che potrebbero risultare meno intuitive è l'attore, ovvero un ruolo coperto da un certo insieme di entità del sistema; spesso tale ruolo viene ricoperto dall'utente, ma è possibile anche l'utilizzo di entità astratte, come il tempo o entità del sistema stesso. Il cuore centrale del modulo è rappresentato dalla sequenza degli eventi principali, ovvero un elenco ordinato di passi che costituiscono lo scenario; generalmente la prima azione è eseguita dall'attore principale che permette l'inizio del caso d'uso. Il seguente elenco mostra, in sintesi, le informazioni del modulo:

- *ID:* identificatore del caso d'uso;
- *Breve descrizione:* consente una rapida comprensione dell'obiettivo del caso d'uso;
- *Attori primari e secondari:* elenco delle entità che interagiscono con il sistema;
- *Precondizioni:* lo stato del sistema prima che il caso d'uso possa iniziare;
- *Sequenza degli eventi principali:* elenco dei passi che compongono lo scenario;
- *Postcondizioni:* descrive lo stato del sistema al termine degli eventi;

- *Sequenza degli eventi alternative:* elenco degli di scenari secondari che possono eventualmente innescarsi.

Inserimento di un professore

Nella Tabella 2.4 viene riportato il caso d'uso relativo all'inserimento di un nuovo professore.

Caso d'uso: CreaProfessore
ID:1
Breve descrizione: Il sistema crea un nuovo professore.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola.
Sequenza degli eventi principali: 1. Il caso d'uso inizia quando l'utente seleziona "nuovo professore". 2. L'utente inserisce i dati del professore. 3. Il sistema salva il professore con i relativi dati. 4. Il sistema aggiorna la lista dei professori.
Postcondizioni: Un nuovo professore è stato inserito.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.4: Caso d'uso relativo all'inserimento di un professore

Modifica di un professore

Nella Tabella 2.5 viene riportato il caso d'uso relativo alla modifica di un professore.

Caso d'uso: ModificaProfessore
ID:2
Breve descrizione: Il sistema modifica i dati di un professore.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola
Sequenza degli eventi principali: 1. `Include` (VisualizzaProfessore). 2. Il sistema mostra i dati del professore. 3. L'utente modifica i dati desiderati. 4. Il sistema salva i dati aggiornati relativi al professore.
Postcondizioni: I dati del professore sono stati aggiornati.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.5: Caso d'uso relativo alla modifica di un professore

Visualizzazione di un professore

Nella Tabella 2.6 viene riportato il caso d'uso relativo alla visualizzazione di un professore.

Caso d'uso: VisualizzaProfessore
ID:3
Breve descrizione: Il sistema mostra all'utente i dati di un professore.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola.
Sequenza degli eventi principali: 1. Il caso d'uso inizia quando l'utente seleziona un professore. 2. Il sistema mostra i dati del professore. 3. L'utente smette di visualizzare i dati.
Postcondizioni: Nessuna.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.6: Caso d'uso relativo alla visualizzazione di un professore

Rimozione di un professore

Nella Tabella 2.7 viene riportato il caso d'uso relativo alla rimozione di un professore.

Caso d'uso: EliminaProfessore
ID:4
Breve descrizione: Il sistema elimina un professore e i relativi dati.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola. 3. Il professore è visualizzabile.
Sequenza degli eventi principali: 1. Il caso d'uso inizia quando l'utente seleziona un professore. 2. L'utente seleziona "elimina professore". 3. L'utente conferma che desidera eliminare il professore. 4. Il sistema elimina il professore con i relativi dati. 5. Il sistema aggiorna la lista dei professori.
Postcondizioni: Il professore è stato eliminato.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.7: Caso d'uso relativo alla rimozione di un professore

Inserimento di un orario

Nella Tabella 2.8 viene riportato il caso d'uso relativo all'inserimento di un orario.

Caso d'uso: CreaOrario
ID:5
Breve descrizione: Il sistema crea un nuovo orario.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola. 3. Esiste almeno un professore.
Sequenza degli eventi principali: 1. Il caso d'uso inizia quando l'utente seleziona "nuovo orario". 2. L'utente inserisce i dati del nuovo orario. 3. Il sistema salva l'orario con i relativi dati. 4. Il sistema aggiorna la lista degli orari.
Postcondizioni: Un nuovo orario è stato inserito.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.8: Caso d'uso relativo all'inserimento di un orario

Modifica di un orario

Nella Tabella 2.9 viene riportato il caso d'uso relativo alla modifica di un orario.

Caso d'uso: ModificaOrario
ID:6
Breve descrizione: Il sistema modifica i dati di un orario.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola.
Sequenza degli eventi principali: 1. `Include` (VisualizzaOrario). 2. Il sistema mostra i dati dell'orario. 3. L'utente modifica i dati desiderati. 4. Il sistema salva i dati aggiornati relativi all'orario.
Postcondizioni: I dati dell'orario sono stati aggiornati.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.9: Caso d'uso relativo alla modifica di un orario

Visualizzazione di un orario

Nella Tabella 2.10 viene riportato il caso d'uso relativo alla visualizzazione di un orario.

Caso d'uso: VisualizzaOrario
ID:7
Breve descrizione: Il sistema mostra all'utente i dati di un orario.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola.
Sequenza degli eventi principali: 1. Il caso d'uso inizia quando l'utente seleziona un orario. 2. Il sistema mostra i dati dell'orario. 3. L'utente smette di visualizzare i dati.
Postcondizioni: Nessuna.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.10: Caso d'uso relativo alla visualizzazione di un orario

Rimozione di un orario

Nella Tabella 2.11 viene riportato il caso d'uso relativo alla rimozione di un orario.

Caso d'uso: EliminaOrario
ID:8
Breve descrizione: Il sistema elimina un orario e i relativi dati.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola. 3. L'orario è visualizzabile.
Sequenza degli eventi principali: 1. Il caso d'uso inizia quando l'utente seleziona un orario. 2. l'utente seleziona "elimina orario". 3. l'utente conferma che vuole eliminare l'orario. 4. Il sistema elimina l'orario con i relativi dati. 5. Il sistema aggiorna la lista degli orari.
Postcondizioni: L'orario è stato eliminato.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.11: Caso d'uso relativo alla rimozione di un orario

Salvataggio di un orario in formato PDF

Nella Tabella 2.12 viene riportato il caso d'uso relativo al salvataggio in formato PDF di un orario.

Caso d'uso: ScaricaOrario
ID:9
Breve descrizione: Il sistema salva un orario in formato PDF.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola. 3. L'orario è visualizzabile.
Sequenza degli eventi principali: 1. `Include`(VisualizzaOrario) 2. L'utente seleziona "salva in pdf". 3. Il sistema salva l'orario con i relativi dati in formato PDF. 4. Il sistema aggiorna la lista degli orari in formato PDF.
Postcondizioni: L'orario è stato salvato.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.12: Caso d'uso relativo al salvataggio di un orario in formato PDF

Inserimento di una scuola

Nella Tabella 2.13 viene riportato il caso d'uso relativo all'inserimento di una nuova scuola.

Caso d'uso: CreaScuola
ID:10
Breve descrizione: Il sistema crea una nuova scuola.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: Nessuna.
Sequenza degli eventi principali: 1. Il caso d'uso inizia quando l'utente decide di inserire nuove credenziali. 2. l'utente seleziona "iscriviti". 3. Il sistema crea la scuola con i relativi dati.
Postcondizioni: La scuola è stata generata.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.13: Caso d'uso relativo all'inserimento di una scuola

Modifica di una scuola

Nella Tabella 2.14 viene riportato il caso d'uso relativo alla modifica di una scuola.

Caso d'uso: ModificaScuola
ID:11
Breve descrizione: Il sistema modifica i dati di una scuola.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. Esiste una scuola.
Sequenza degli eventi principali: 1. `Include` (VisualizzaScuola). 2. Il sistema mostra i dati della scuola. 3. L'utente modifica i dati desiderati. 4. Il sistema salva i dati aggiornati relativi alla scuola.
Postcondizioni: La scuola è stata modificata.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.14: Caso d'uso relativo alla modifica di una scuola

Visualizzazione di una scuola

Nella Tabella 2.15 viene riportato il caso d'uso relativo alla visualizzazione di una scuola.

Caso d'uso: VisualizzaScuola
ID:12
Breve descrizione: Il sistema mostra i dati di una scuola.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola.
Sequenza degli eventi principali: 1. Il caso d'uso inizia quando l'utente seleziona "info scuola". 2. Il sistema mostra i dati della scuola. 3. l'utente smette di visualizzare i dati.
Postcondizioni: Nessuna.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.15: Caso d'uso relativo alla visualizzazione di una scuola

Rimozione di una scuola

Nella Tabella 2.16 viene riportato il caso d'uso relativo alla rimozione di una scuola.

Caso d'uso: EliminaScuola
ID:13
Breve descrizione: Il sistema elimina una scuola e i relativi dati.
Attori primari: Utente.
Attori secondari: Nessuno.
Precondizioni: 1. L'utente è stato autenticato dal sistema. 2. Esiste una scuola.
Sequenza degli eventi principali: 1. l'utente seleziona "elimina scuola". 2. l'utente conferma di voler eliminare la scuola. 3. Il sistema elimina la scuola con i relativi dati.
Postcondizioni: La scuola è stata eliminata.
Sequenza degli eventi alternative: Nessuna.

Tabella 2.16: Caso d'uso relativo alla rimozione di una scuola

Diagramma dei casi d'uso

Per evidenziare in modo grafico e diretto la relazione tra attori e casi d'uso, si costruiscono dei diagrammi UML (Unifield Modeling Language). Quest'ultimo è un linguaggio di modellazione che permette, attraverso semplici figure, la rappresentazione di funzioni o entità del sistema. Le Figure 2.1, 2.2 e 2.3 mostrano i diagrammi dei casi d'uso relativi al sistema oggetto della presente tui. I diagrammi dei casi d'uso sono un particolare costrutto UML che illustra il modo in cui gli attori interagiscono con il sistema.

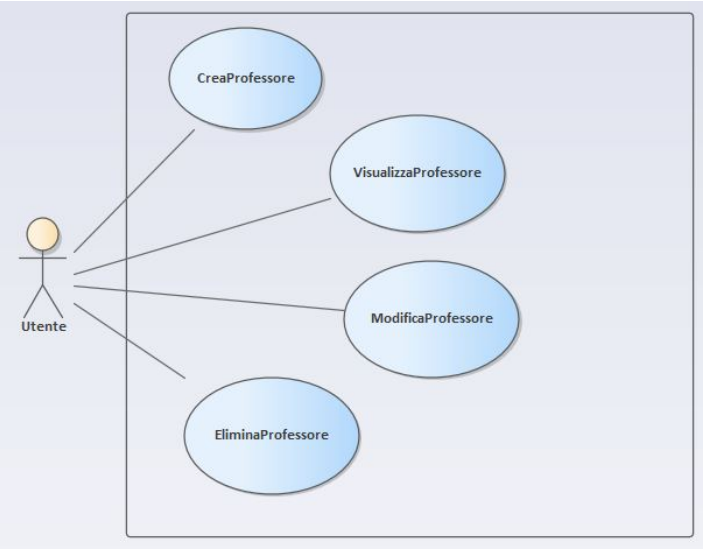

Figura 2.1: Diagramma casi d'uso relativo ai professori

Attori

La Figura 2.4 mostra gli attori del sistema.

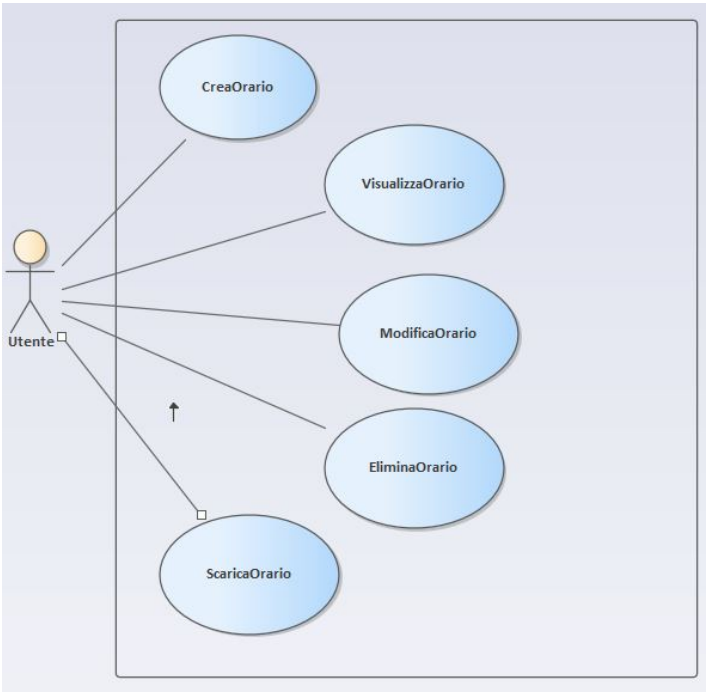

Figura 2.2: Diagramma casi d'uso relativo agli orari

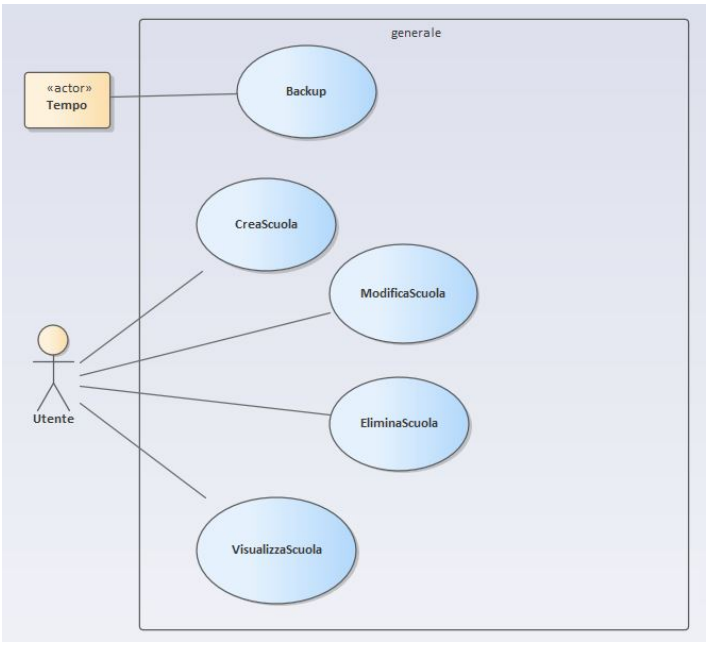

Figura 2.3: Diagramma casi d'uso relativo alla scuola

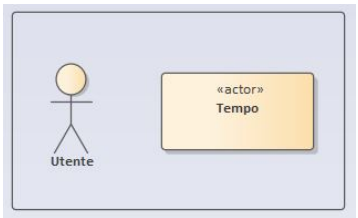

Figura 2.4: Attori

CAPITOLO 3

Analisi

In questo capitolo viene trattato il flusso di lavoro dell'analisi, ovvero la fase di progettazione che mira a produrre un modello di analisi. Tale modello si concentra su cosa il software debba fare, fornendo una descrizione del comportamento desiderato del sistema stesso. Nei vari paragrafi viene descritta l'elaborazione di diagrammi di classe, sequenza e attività, ottenuti sfruttando il paradigma orientato agli oggetti e i diagrammi UML.

3.1 La fase di analisi nello sviluppo software

Il flusso di lavoro di analisi, in genere, viene identificato nella fase di elaborazione di un progetto software e ha lo scopo di costruire un modello che descrive il comportamento desiderato del sistema. Nel ciclo di vita del software, la fase di analisi si colloca tra la raccolta dei requisiti e la progettazione; possiamo quindi dire che tale fase sia un ponte tra le esigenze del cliente e la costruzione tecnica di una soluzione per soddisfarle. La principale difficoltà, spesso, risiede nel modellare adeguatamente i requisiti, evitando di generare complicazioni per le fasi successive; la fase di analisi, infatti, traduce le richieste del cliente in un modello coerente, chiaro e completo, ponendo le basi per una buona fase di progettazione. Il seguente listato richiama alcune regole importanti che sono state seguite per una buona modellazione dell'analisi:

- *Mantenere una distinzione tra il dominio del problema e il dominio delle soluzioni;*
- *Esprimere il modello dell'analisi nel linguaggio del problema;*
- *Mantenere semplice il modello;*
- *Concentrarsi sulla descrizione della vista d'insieme;*
- *Assicurare coerenza interna ed esterna del modello;*

3.2 Classi di analisi

La classe è un costrutto che definisce una categoria di oggetti che condividono le stesse proprietà (attributi) e comportamenti (metodi); tale concetto è l'astrazione fondamentale del paradigma orientato agli oggetti, ossia l'architettura utilizzata per descrivere al meglio le entità presenti nel software. L'idea di questo approccio risale intorno agli anni'60, quando gli informatici norvegesi Kristen Nygaard e Ole-Johan Dahl gettarono le basi del concetto di classe attraverso lo sviluppo di Simula (1962), un linguaggio di programmazione concepito per le simulazioni numeriche. In Simula, le entità erano rappresentate da oggetti che, avendo le stesse proprietà e comportamenti, venivano raggruppati in classi che fungevano da sotto-programmi per la simulazione. Negli anni '70, invece, Alan key e il gruppo di ricerca PARC Company svilupparono Smalltalk, un linguaggio che ridefinì l'idea di oggetto ereditando le caratteristiche principali delle classi da Simula, rendendo la classe un oggetto dinamico. Dopo Simula e Smalltalk, il concetto di classe divenne interessante e si diffuse nei principali linguaggi di programmazione presenti già negli anni '80; i linguaggi C++ e Java si ispirarono a tale concetto ereditando caratteristiche importanti rispettivamente da Simula e Smalltalk, come l'ereditarietà e la raccolta automatica della memoria. Le classi di analisi riprendono il concetto di classe appena discusso, rappresentando un'astrazione ben definita nel dominio del problema.

3.2.1 Diagramma delle classi

Nella Figura 3.1 sono rappresentate le classi che modellano le entità principali del programma: gli orari, la scuola e i professori. Ogni riquadro raffigura una classe con i principali attributi.

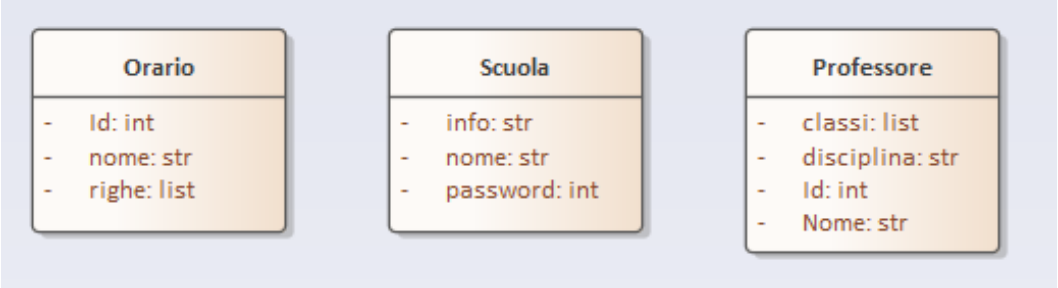

Figura 3.1: Classi delle principali entità del programma

Scuola

La scuola è l'entità più importante, poiché senza di essa non potrebbero esistere i professori, tanto meno gli orari. Gli oggetti di questa classe sono caratterizzati dai seguenti attributi:

- *Info:* informazioni generiche della scuola;
- *Password:* credenziale privata;
- *Nome:* nome della scuola;

Professore

La classe `Professore` modella tutti i docenti appartenenti alla scuola, l'esistenza degli oggetti di questa classe è necessaria per la creazione di un orario scolastico. Gli oggetti di questa classe sono caratterizzati dai seguenti attributi:

- *Id:* identificatore dell'oggetto;
- *Nome:* nome del professore;
- *Classi:* lista delle classi in cui insegna il professore;
- *Disciplina:* materia di insegnamento del professore;

Orario

La classe `Orario` è concepita per modellare l'orario scolastico ed è caratterizzata dai seguenti attributi:

- *Id:* identificatore dell'oggetto;
- *Nome:* nome dell'orario;
- *Righe:* lista di righe che compongono l'orario;

3.2.2 Relazioni tra classi

Nella Figura 3.2 sono evidenziate le associazioni tra le classi. In un diagramma UML, un'associazione rappresenta una relazione strutturale tra due classificatori, in questo caso si tratta di una relazione tra classi. L'utilizzo di tale strumento aiuta a comprendere le scelte di progettazione e il modo in cui le entità interagiscono tra loro.

Associazione tra Scuola e Gestori

La classe `Scuola` utilizza `GestoreOrari` e `GestoreProfessori`, ossia le classi che, come vedremo nella fase di progettazione, saranno costituite da una sola istanza per gestire gli oggetti delle classi a cui sono associate. Per indicare lo scopo della relazione, nel diagramma è stata utilizzata la dipendenza `usa`.

Associazione tra GestoreProfessori e Professore

`GestoreProfessori` può gestire zero o più professori, mentre tutti i professori sono gestiti da un solo oggetto della classe `GestoreProfessori`.

Associazione tra GestoreOrari e Orario

`GestoreOrari` può gestire zero o più orari, mentre tutti gli orari sono gestiti da un solo oggetto della classe `GestoreOrari`.

Associazione tra Professore e Orario

Un docente può comparire in più orari generati dal programma, mentre un orario deve includere almeno un professore per esistere (anche se incompleto).

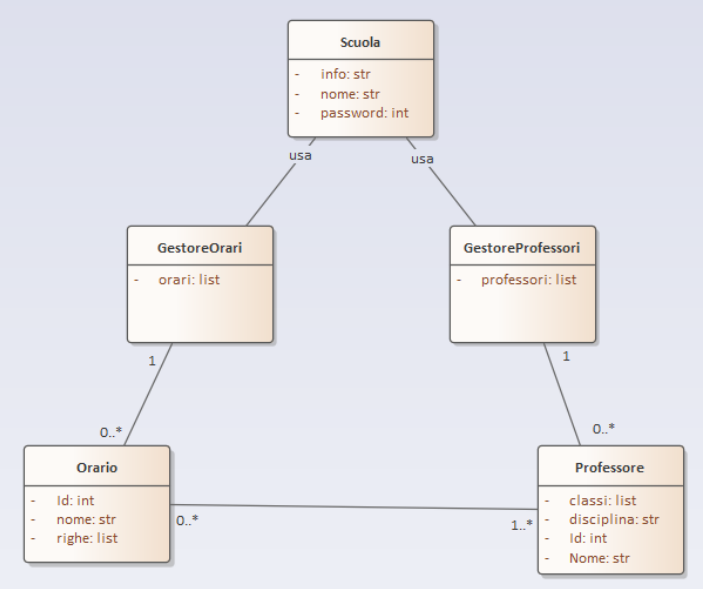

Figura 3.2: Diagramma delle classi principali del programma

3.3 Diagramma di sequenza

I diagrammi di sequenza sono diagrammi di interazione UML che rappresentano graficamente le modalità di interazione tra le istanze di un programma. Tali diagrammi esprimono una comunicazione tra istanze come una sequenza di eventi ordinati temporalmente. Le linee di vita di un'entità (o di un oggetto) si attivano nel momento in cui avviene uno scambio di messaggi con un'altra entità, mostrando il proprio stato attivo. Al fine di una più semplice comprensione del funzionamento del programma, nell'elaborato sono stati inseriti i diagrammi delle principali figure: i professori e gli orari.

Gestione dei professori

Nella Figura 3.3 viene mostrato il diagramma di sequenza relativo alla gestione dei professori. Le interazioni principali riguardano l'amministratore che, grazie a `GestoreProfessori`, è in grado di gestire i professori. Il riquadro `Alt` consente la visualizzazione di 4 diversi scenari: visualizzazione, aggiunta, modifica e rimozione.

Gestione degli orari

Nella Figura 3.4 viene mostrato il diagramma di sequenza relativo alla gestione degli orari. Le interazioni principali riguardano l'amministratore che, grazie a `GestoreOrari`, è in grado di gestire gli orari. Il riquadro `Alt` consente la visualizzazione di 5 diversi scenari: visualizzazione, aggiunta, modifica, rimozione e salvataggio in PDF.

3.4 Diagramma di attività

I diagrammi di attività sono diagrammi di interazione UML che consentono la visualizzazione del comportamento del software nei diversi casi d'uso. La possibilità di modellare un processo senza dover specificare la sua struttura statica costituisce un vantaggio nell'esposizione chiara e semplice di un flusso di lavoro. Tale caratteristica rappresenta la motivazione principale per cui questo tipo di diagramma è stato inserito nell'elaborato.

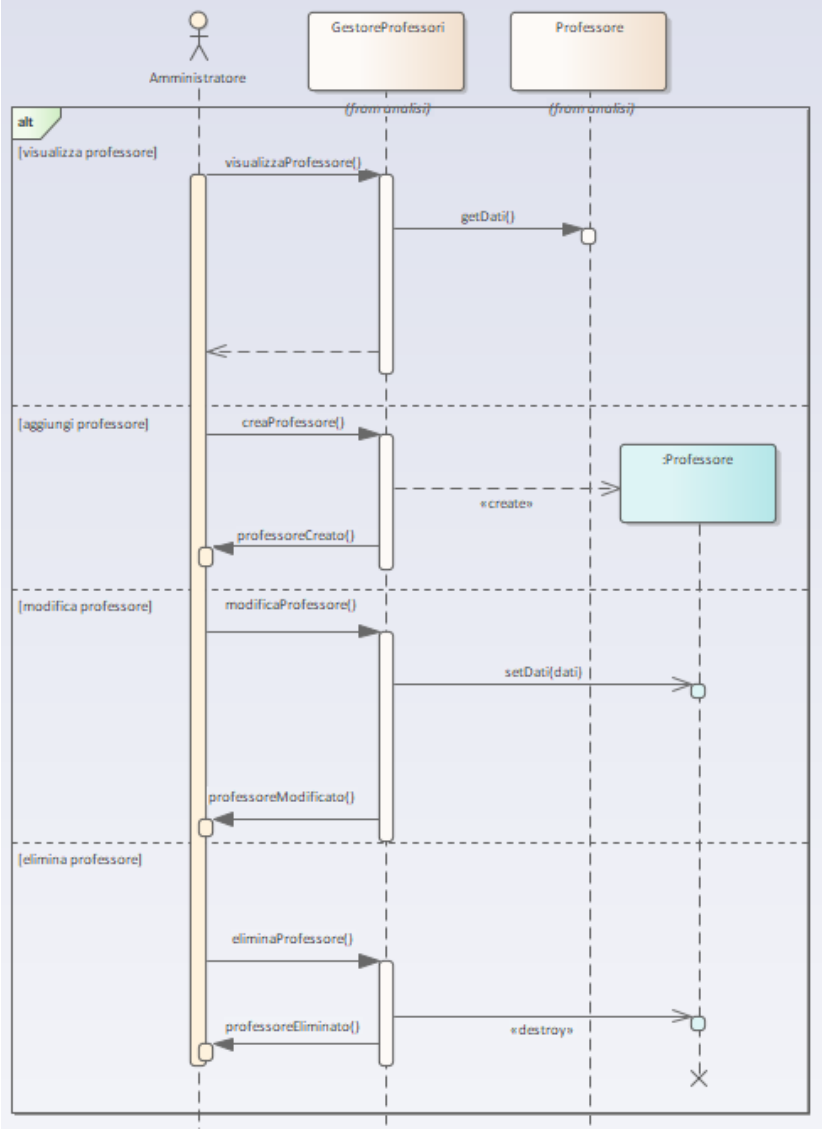

Figura 3.3: Diagramma di sequenza relativo alla gestione dei professori

Attività dei professori

Nella figura 3.5 vengono mostrate le attività riguardanti la gestione dei professori. Un nodo iniziale è collegato a diversi nodi azione, creando una diramazione di percorsi diversi in base ai casi d'uso; infine, un nodo finale chiude il diagramma.

Attività degli orari

Nella figura 3.6 vengono mostrate le attività riguardanti la gestione degli orari. Un nodo iniziale è collegato a diversi nodi azione, creando una diramazione di diversi percorsi in base ai casi d'uso; infine, un nodo finale chiude il diagramma.

Attività riguardante la modifica degli orari

La Figura 3.7 approfondisce la sequenza di azioni che costituiscono l'operazione di modifica di un orario. Nel diagramma, oltre ai nodi azione, sono raffigurati dei rombi, utili per la rappresentazione di diversi scenari.

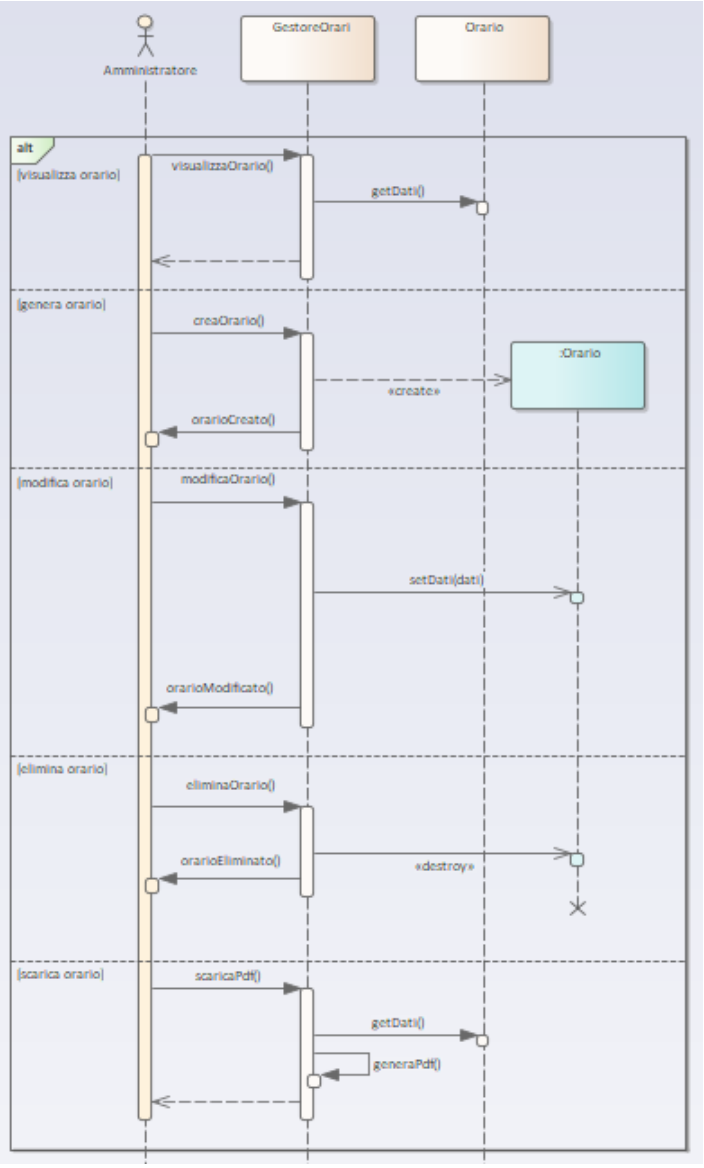

Figura 3.4: Diagramma di sequenza relativo alla gestione degli orari

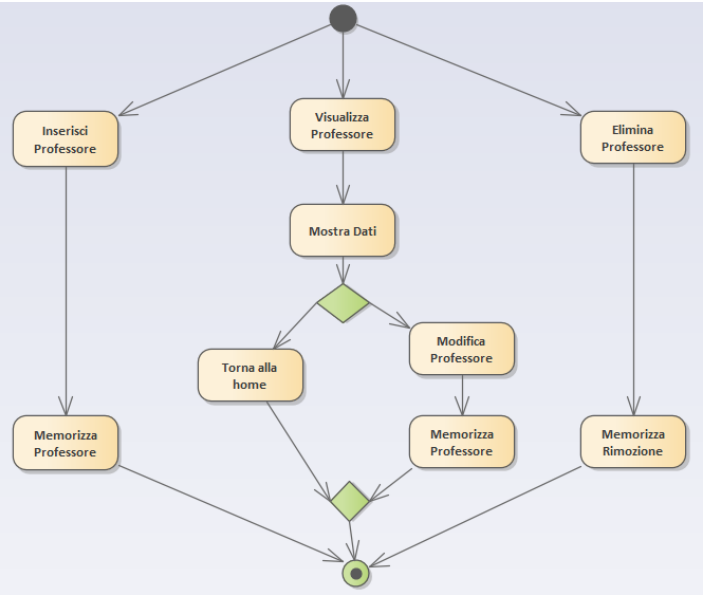

Figura 3.5: Diagramma di attività relativi alla gestione dei professori

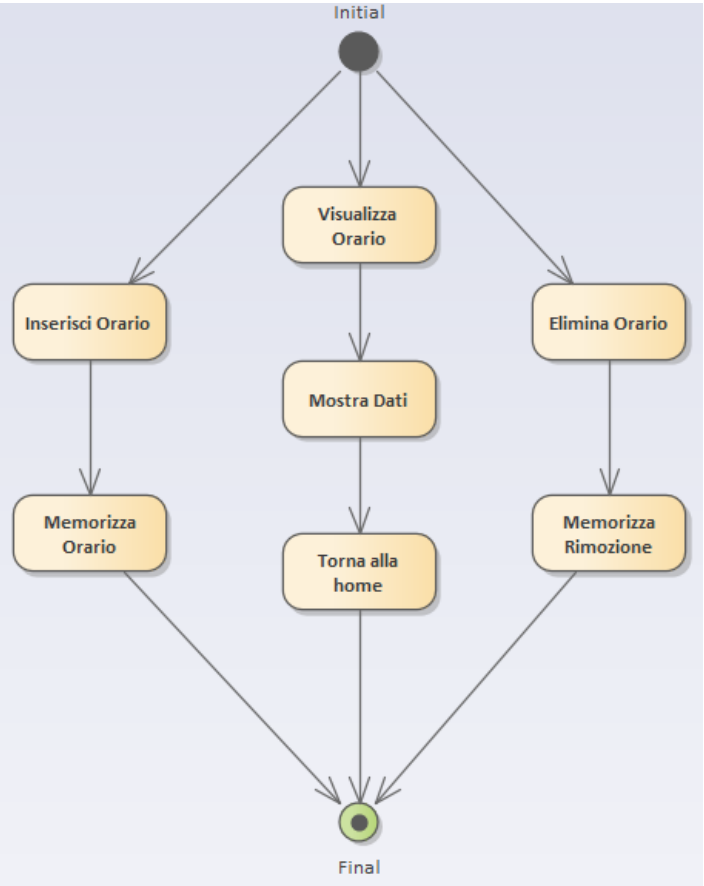

Figura 3.6: Diagramma di attività relativi alla gestione degli orari

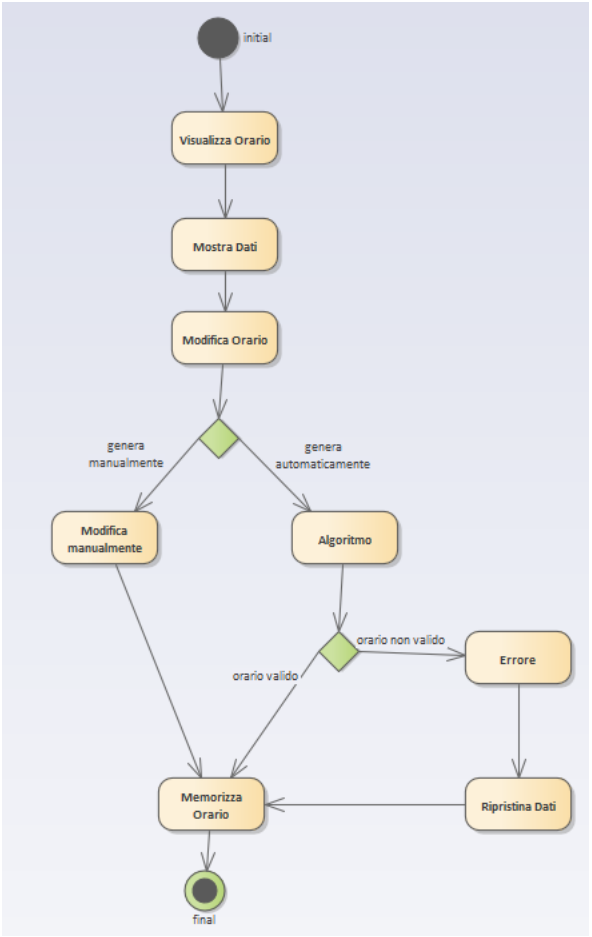

Figura 3.7: Diagramma di attività relativi alle modifiche degli orari

Attività riguardanti il salvataggio degli orari

La figura 3.8 mostra le attività riguardanti la gestione del salvataggio in PDF di un orario.

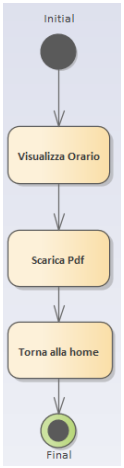

Figura 3.8: Diagramma di attività relativi al salvataggio degli orari

CAPITOLO 4

Progettazione

In questo capitolo viene trattata la fase di progettazione, un passaggio cruciale nello sviluppo software, poiché i prodotti che ne derivano, come la classe di progettazione, determinano la corretta implementazione del sistema. L'obiettivo del capitolo è mostrare una panoramica della fase di progettazione nel mondo dello sviluppo software, per poi illustrare lo schema delle classi e dell'algoritmo del completamento degli orari, che sono i prodotti sviluppati nella fase di implementazione.

4.1 La fase di progettazione nello sviluppo software

La fase di progettazione è la parte del processo in cui si cerca di costruire un modello del sistema che possa essere implementato. A seconda della complessità del progetto, questa fase può esistere in maniera indipendente oppure essere inclusa nella fase di analisi. In linea teorica, la fase di progettazione si compone di diversi passaggi; il seguente elenco ne mostra i principali:

- *Sottosistema di progettazione:* suddivisione del sistema in sottosistemi semplici e funzionali.
- *Classi di progettazione:* si cerca di definire in modo completo le classi.
- *Interfacce:* si progetta ciò che verrà mostrato all'utente.
- *Progettazione delle realizzazioni dei casi d'uso:* si chiarisce come il sistema andrà a soddisfare i requisiti.
- *Diagramma di sviluppo:* si modellano le componenti software sui nodi hardware.

Poiché la complessità del progetto non è elevata, questo elaborato mostra: le classi di progettazione, le interfacce e l'algoritmo per il completamento dell'orario scolastico. Molte problematiche che si riscontrano in un software, come i costi elevati di mantenimento, si verificano perché l'implementazione del software si basa su una progettazione inefficiente che non ottimizza molti aspetti del programma. Seguire validi principi di progettazione aiuta ad evitare questi inconvenienti; infatti, prima di implementare un software, si valuta la modularità del progetto, si definiscono le relazioni complesse tra classi e si valuta il grado di coesione e accoppiamento; la Tabella 4.1 approfondisce il significato dei principi appena citati.

Principio	Descrizione
Modularità	Capacità di suddividere il sistema in moduli indipendenti mediante la decomposizione del dominio del problema. Questa caratteristica facilita lo sviluppo, permettendo la rappresentazione del funzionamento del software anche in assenza di moduli completamente implementati.
Relazioni complesse	Rappresentazione di relazioni avanzate tra gli elementi del sistema, come l'ereditarietà, le dipendenze funzionali e le aggregazioni tra classi.
Coesione	Grado di correlazione funzionale tra le responsabilità di un modulo. Una bassa coesione comporta difficoltà nella manutenzione, nel riutilizzo del codice e nella comprensione del funzionamento complessivo del programma.
Accoppiamento	Misura del livello di dipendenza tra i moduli del software. Un elevato accoppiamento indica una progettazione inefficiente, poiché introduce numerose dipendenze esterne che riducono la manutenibilità del sistema. La Figura 4.1 illustra un esempio per comprendere meglio questo principio

Tabella 4.1: Definizione dei principi della progettazione software

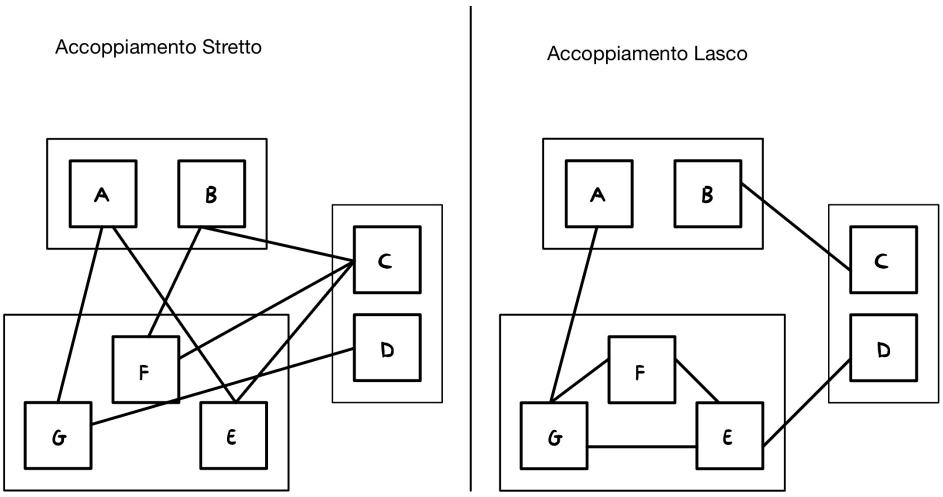

Figura 4.1: Esempio di accoppiamento lasco e stretto

4.2 Classi di progettazione

Nella fase di analisi è stato illustrato un diagramma UML che rappresenta le classi con le dovute relazioni; in questa fase avviene una rifinitura del diagramma. Le classi di progettazione risultano, infatti, più specifiche rispetto a quelle sviluppate nella fase di analisi. Al fine di definire come le classi assolveranno le loro responsabilità, vengono elencati gli attributi e i metodi di ciascuna classe, delineando, così, un quadro più chiaro del sistema. Le classi sono state sviluppate in modo da rispettare le seguenti caratteristiche:

- *Completezza:* ogni classe contiene tutte le informazioni necessarie.
- *Essenzialità:* ogni classe include esclusivamente attributi e metodi rilevanti.
- *Massima coesione:* gli elementi interni a una classe sono fortemente correlati.
- *Minima interdipendenza:* le classi dipendono il meno possibile l'una dall'altra.

Di seguito vengono approfondite le proprietà delle classi, definendo i principali metodi e attributi. La Figura 4.2 illustra il diagramma UML completo delle classi.

Scuola

La classe `Scuola` si relaziona con `GestoreOrari` e `GestoreProfessori`, ovvero le due classi responsabili delle operazioni su orari e professori. Si sottolinea, tra gli attributi, `disciplineVerifiche` ed `infoScuola`: il primo attributo è una lista delle materie che necessitano di 2 ore accoppiate per le verifiche, il secondo è un testo composto dall'utente che riguarda le informazioni della scuola. Tra i metodi, invece, `caricaScuola()` e `salvaScuola()` gestiscono i dati durante la comunicazione con il database; i metodi restanti sono finalizzati alla gestione degli attributi.

GestoreProfessori

Questa classe si occupa di gestire gli oggetti di tipo `Professore` e, nel programma, esisterà solo un'istanza di tipo `GestoreProfessori`. L'attributo `listaProfessori` è una lista di oggetti di tipo professore e viene richiamata ogni volta che si devono eseguire operazioni su uno o più professori. L'attributo `nextId` è necessario per assegnare in modo univoco un identificatore ai professori.

GestoreOrari

Questa classe si occupa di gestire gli oggetti di tipo `Orario` e, nel programma, esisterà solo un'istanza di tipo `GestoreOrari`. L'attributo `listaOrari` è una lista di oggetti di tipo `Orario` e viene richiamata ogni volta che si devono eseguire operazioni su uno o più orari. L'attributo `nextId` è necessario per assegnare in modo univoco un identificatore agli orari.

Professore

Questa classe rappresenta un docente dell'istituto con nome, ore di lavoro, materia di insegnamento (disciplina) e un identificatore assegnato grazie a `GestoreProfessori`. L'attributo `classi` è una lista di classi in cui il professore insegna, mentre `eccezioni` rappresenta le ore della settimana in cui il docente non può lavorare. Tutti i metodi visibili nel diagramma sono utili alla gestione degli attributi.

Orario

Questa classe rappresenta un orario generato dall'utente ed è provvista di un nome e di un identificatore univoco. L'attributo `listaRighe` è una matrice, ossia il corpo dell'orario scolastico; nella prima posizione di ogni riga si trova il nome di un professore, seguito da 30 elementi, i quali rappresentano le 6 ore di ogni giorno lavorativo della settimana. Tutti i metodi visibili nel diagramma sono utili alla gestione degli attributi. La classe si relaziona con `Professore`, evidenziando che uno o più professori possono essere visualizzati in un orario.

Algoritmo

Questa classe rappresenta l'algoritmo per la generazione dell'orario scolastico. La relazione con la dipendenza «`use`» specifica che l'oggetto `Orario` utilizza l'istanza di questa classe per generare automaticamente le righe mancanti.

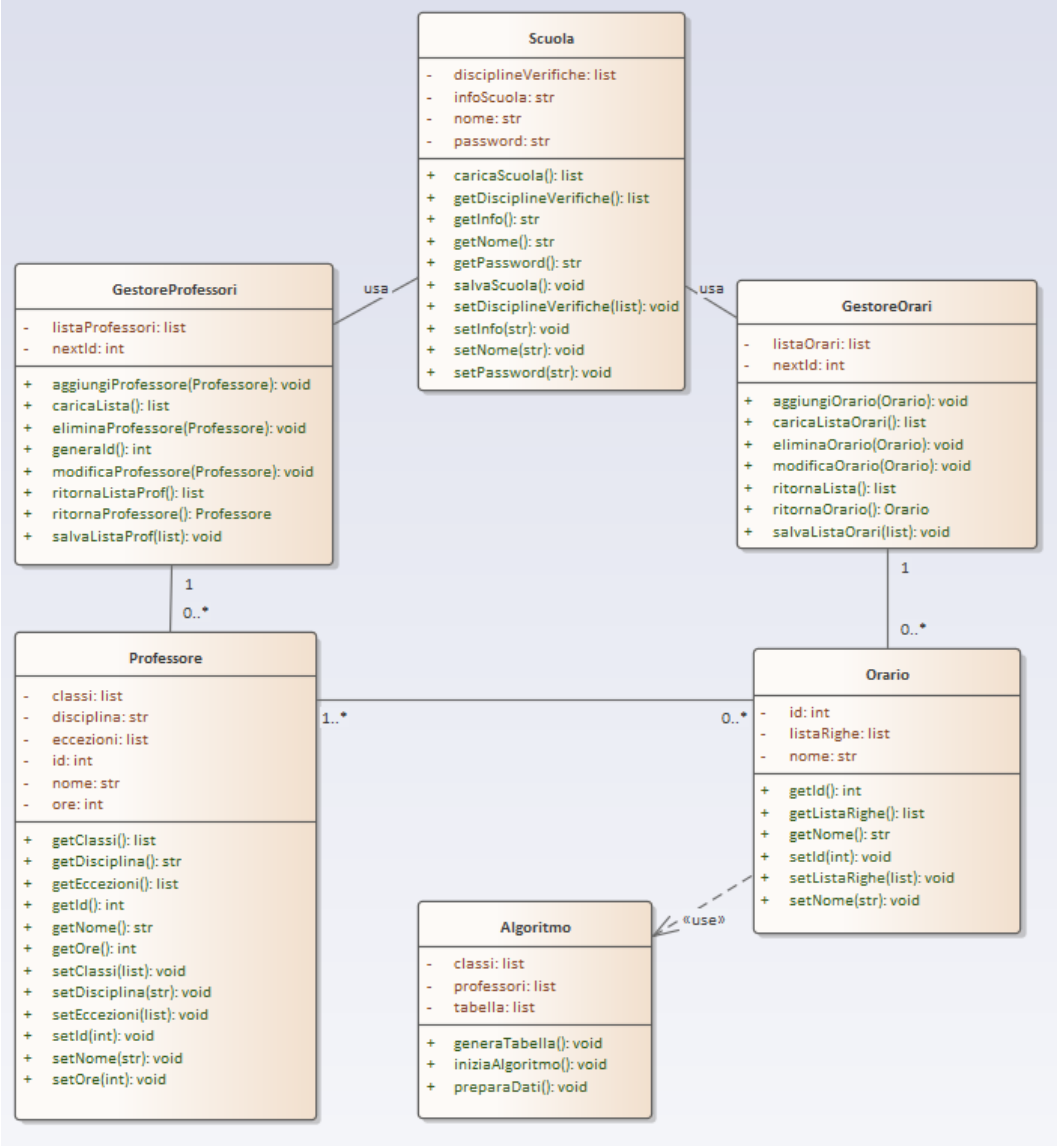

Figura 4.2: Diagramma delle classi del programma

4.3 Algoritmo per l'orario scolastico

Il primo passo per sviluppare un algoritmo è sicuramente quello di comprendere la natura del problema. I problemi di ottimizzazione non sono tutti uguali; una fabbrica che produce oggetti, come plastica o vernici, necessita di prodotti modellabili tramite variabili continue, mentre in una gestione del personale aziendale, le turnazioni sono caratterizzate dal numero di ore, le quali possono essere modellate come variabili discrete. Esistono, quindi, problemi di Programmazione Lineare e problemi di Programmazione Lineare Intera (PLI); quest'ultima categoria include anche la realizzazione di un orario scolastico.

4.3.1 Alternative nei problemi di PLI

Un problema di Programmazione Lineare Intera è un problema di ottimizzazione matematica nel quale si ricerca la massimizzazione o la minimizzazione di una funzione obiettivo. Nel caso specifico dell'orario scolastico, tuttavia, l'obiettivo principale non è l'ottimizzazione di una grandezza numerica, bensì la costruzione di una tabella oraria che soddisfi sia i vincoli

forti sia quelli deboli. Questo tipo di problema rientra nella categoria della Programmazione Lineare Intera e, più precisamente, è noto come problema di scheduling. Per la risoluzione di questa tipologia esistono diverse alternative, dalla più professionale alla più pratica; il seguente elenco ne mostra le principali:

- *Backtracking:* si prova a sistemare una classe alla volta, tornando indietro ad ogni violazione di uno o più vincoli.
- *Programmazione Lineare Intera di fattibilità:* si cerca una soluzione che soddisfi tutti i vincoli, ignorando la ricerca della soluzione ottimale.
- *Approccio euristico:* si completa l'orario usando regole pratiche ed intuitive, come sistemare per primi i professori con turni più complessi.
- *Approccio ibrido:* si ottiene un risultato che soddisfi i vincoli, per poi migliorarlo ottimizzando dei parametri, come la riduzione delle ore di buco.

4.3.2 Progettare l'algoritmo

In questa sezione è descritto l'algoritmo per la creazione dell'orario scolastico. Il prodotto finale dovrà essere una tabella che rappresenti i professori con le ore di insegnamento per ogni giorno di lavoro. La Figura 4.3 mostra lo schema generale dell'intero processo; i blocchi sono descritti di seguito e saranno approfonditi durante la fase di implementazione.

Setup

Nel blocco `Setup` avviene una preparazione dei dati per procedere con l'ordinamento; si identificano i professori che hanno già delle ore assegnate manualmente dall'utente.

Ordina professori

Nel blocco `Ordina professori` avviene un ordinamento dei docenti in base al grado di difficoltà per l'assegnazione delle ore lavorative. Questo passaggio è importante perché un ordine intelligente dei professori migliora le prestazioni dell'algoritmo.

Il ciclo While

Questo blocco iterativo specifica che la fine della fase di completamento dell'orario si verifica in due condizioni: i tentativi per ottenere un orario valido superano un certo valore, oppure abbiamo ottenuto un orario prima di superare il limite di tentativi; quest'ultimo evento viene riconosciuto dal programma tramite una variabile.

Inizializza dati

In questo blocco vengono ripristinati i dati dei professori e delle classi che sono stati modificati nel tentativo fallimentare dell'iterazione precedente, consentendo una nuova iterazione con i dati corretti.

Associa ore

In questo blocco vengono assegnati ai docenti i blocchi orari, in quanto, a seconda della disciplina insegnata, un professore può necessitare di lezioni della durata di due ore consecutive.

Cicli iterativi classe-professore

Dopo il blocco `Associa ore`, sono presenti due `For` annidati. L'algoritmo completa una classe alla volta, sistemando le ore dei professori per ogni classe.

Output orario

Questo blocco viene eseguito se l'algoritmo è terminato con successo; il prodotto finale è un orario scolastico valido.

Errore

Questo blocco viene eseguito se l'algoritmo ha avuto esito negativo. L'utente riceverà un messaggio di errore che avvisa in merito all'incapacità del programma di produrre un orario.

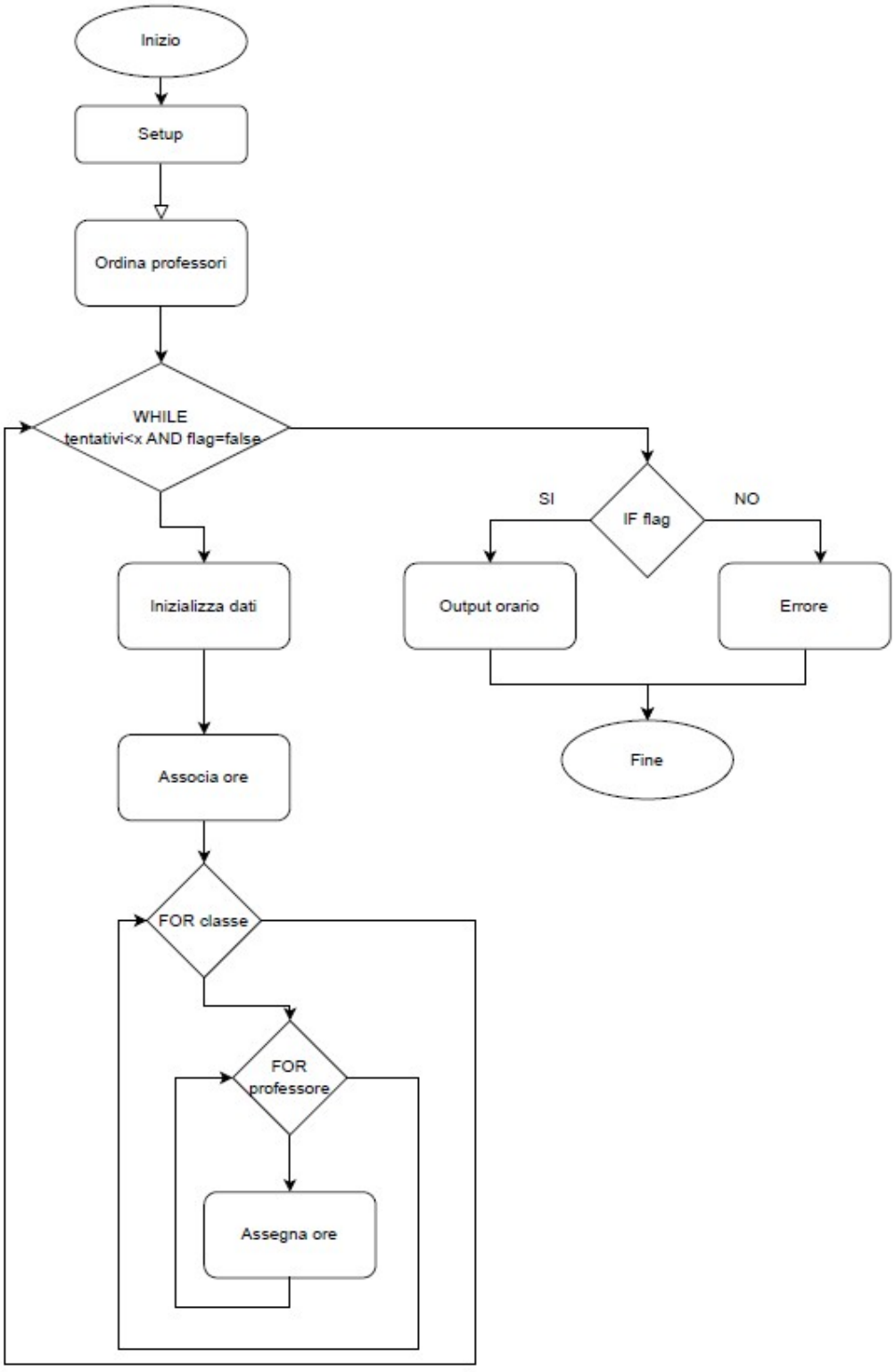

Figura 4.3: Algoritmo per la creazione dell'orario scolastico

CAPITOLO 5

Implementazione

In questo capitolo viene trattata la fase di implementazione, che è l'ultimo passo per lo sviluppo di un software. Il prodotto finale è il software eseguibile, completo di ogni funzione discussa e progettata nelle fasi precedenti. Il capitolo comprende delle sezioni riguardanti il codice delle principali funzioni del programma; la sezione finale mostra il manuale, ovvero le istruzioni per un utilizzo corretto del software.

5.1 Model-View-Controller

Nella fase di implementazione si concretizza il lavoro svolto nei passi precedenti, costruendo il software secondo le linee guida sviluppate nella fase di progettazione e rispettando i requisiti approfonditi nell'analisi. Per l'implementazione di questo progetto è stata scelta l'architettura Model-View-Controller (MVC). Il pattern MVC è un'architettura ben nota nel mondo dello sviluppo software in quanto il suo impiego semplifica la costruzione di sistemi complessi e facilita il lavoro di gruppo. Separare il codice seguendo la logica MVC comporta vantaggi e svantaggi; la Tabella 5.1 ne cita i più evidenti. La Figura 5.1 illustra come i moduli del pattern architetturale interagiscono fra loro. Il Controller è il principale attore dell'architettura, poiché modifica i due moduli con cui comunica; al tempo stesso, però, riceve aggiornamenti su determinati eventi dalla View tramite input che quest'ultimo modulo invia. Il Model memorizza i dati e fornisce i metodi specifici per recuperarli in un secondo momento.

Vantaggi	**Svantaggi**
Promuove il riuso del codice.	Complessità inutilmente elevata per piccoli progetti.
Facilita la manutenzione delle singole componenti.	Spesso al controller è affidata troppa logica da gestire.
Gli sviluppatori che costruiscono la grafica non interferiscono con chi sviluppa la logica.	In alcuni contesti la separazione della logica dall'interfaccia può risultare macchinosa.
Rende il software abbastanza scalabile.	

Tabella 5.1: Vantaggi e svantaggi del pattern MVC

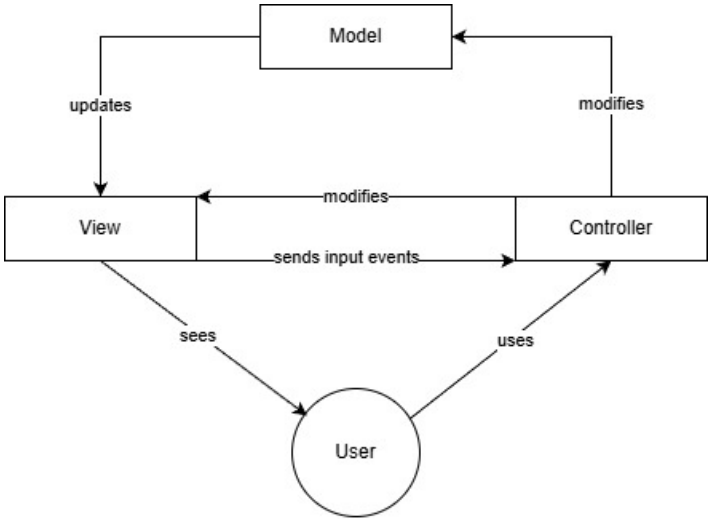

Figura 5.1: Architettura MVC

5.1.1 Model

Il Model è il modulo che si occupa dello stato e della logica dell'applicazione; tutte le classi delle entità del programma sono definite e racchiuse nel Model. In esso avvengono le interazioni con il database, il salvataggio, l'aggiornamento o la cancellazione delle entità; il Model è anche responsabile della logica dei dati. Esso si compone di 5 classi: `GestoreProfessori`, `GestoreOrari`, `Scuola`, `Professore`, `Orario`. Le classi `GestoreProfessori` e `GestoreOrari` sono state implementate seguendo la logica del pattern Singleton, il che significa che esse consentono l'esistenza di un solo oggetto all'interno del programma; il loro scopo consiste nel direzionare le operazioni richieste dall'utente. I gestori sono le classi del Model che interagiscono direttamente con il database.

GestoreProfessori

Nel Listato 5.1 viene riportato il codice che definisce la classe `GestoreProfessori`. Nel Listato 5.2 viene riportato il metodo `carica da file`, che consente il caricamento del file contenente le informazioni dei professori esistenti. Il Listato 5.3 mostra come vengono salvati i professori.

```
class GestoreProfessore:
    def __init__(self):
        self.professori = []
        self.next_id = 0
        self.id_liberi = set()
```

Listing 5.1: Costruttore di `GestoreProfessore`

```
@staticmethod
def carica_da_file(percorso="database/gestore_professori.pkl"):
    if os.path.exists(percorso):
        try:
            with open(percorso, "rb") as f:
                return pickle.load(f)
        except Exception as e:
            print(f"Errore durante il caricamento: {e}")
```

```
        return GestoreProfessore()
```

Listing 5.2: Metodo per il caricamento dei professori

```
    def salva_su_file(self, percorso="database/gestore_professori.pkl"):
        with open(percorso, "wb") as f:
            pickle.dump(self, f)
```

Listing 5.3: Metodo per il salvataggio dei professori

GestoreOrari

Nel Listato 5.4 viene riportato il codice che definisce la classe `GestoreOrari`. Nel Listato 5.5 viene riportato il metodo `carica orari`, che consente il caricamento del file contenente le informazioni degli orari esistenti. Il Listato 5.6 mostra come vengono salvati gli orari.

```
    class GestoreOrari:
        def __init__(self):
            self.orari=[]
            self.next_id = 0
            self.id_liberi = set()
```

Listing 5.4: Costruttore di `GestoreOrari`

```
    @staticmethod
    def carica_orari(percorso="database/orari.pkl"):
        if os.path.exists(percorso):
            try:
                with open(percorso, "rb") as f:
                    print("ha␣trovato")
                    return pickle.load(f)
            except Exception as e:
                print(f"Errore␣durante␣il␣caricamento:␣{e}")
        return GestoreOrari()
```

Listing 5.5: Metodo per il caricamento degli orari

```
        def salva_orari(self):
        with open("database/orari.pkl", "wb") as f:
            print(self.orari)
            pickle.dump(self, f)
```

Listing 5.6: Metodo per il salvataggio degli orari

5.1.2 View

La View si occupa della gestione dell'interfaccia utente, permettendo la visualizzazione di dati e consentendo l'inserimento di informazioni. Ciascuna vista passa gli eventi di input al Controller, il quale comunicherà con il Model per la gestione dei dati.

Pyqt

Per l'implementazione delle viste è stata scelta la libreria `Qt`, nota nel panorama dello sviluppo software per la sua versatilità e semplicità. L'utilizzo del software Qt Designer permette uno sviluppo della GUI in modo grafico e intuitivo, risparmiando allo sviluppatore la costruzione delle viste tramite codice. Alcune delle funzioni presenti in Pyqt sono le seguenti:

- *QtCore:* modulo contenente una serie di risorse per l'organizzazione dei file dell'applicazione.
- *QtWidgets:* fornisce le capacità di base per il rendering sullo schermo, consentendo la gestione degli eventi di input dell'utente.
- *QtGui:* fornisce le classi per l'implementazione della GUI nel software.

QtWidgets

La classe `QtWidgets` racchiude al suo interno diversi elementi di input che sono stati utilizzati nel programma, la lista che segue mostra i più importanti:

- *QPushButton:* pulsante che, se premuto, può scatenare eventi o richiamare funzioni nel programma.
- *QCheckBox:* casella che, se spuntata, ritorna un certo valore.
- *QScrollBar:* sezione che racchiude un contenuto visualizzabile interamente grazie alla barra di scorrimento posta lateralmente.
- *QLabel:* elemento testuale o grafico.
- *QTextEdit:* casella di testo di una o più righe modificabili.
- *QTabWidget:* area di una finestra dove è possibile disporre una o più pagine; per l'impiego di questo elemento è necessario `QStackWidget`.
- *QSpinBox:* numero intero incrementabile.
- *QLineEdit:* casella che contiene un testo modificabile.

5.1.3 Controller

Il Controller funge da intermediario tra il Model e la View, traducendo l'input dell'utente in aggiornamenti che vengono passati al Model. La corretta implementazione di questo modulo è cruciale per trarre vantaggio dai benefici dell'architettura adottata. Il Listato 5.7 mostra le righe di codice chiave del costruttore, fornendo informazioni su come questa classe interagisce con il Model e la View. Le prime righe mostrano la costruzione della vista principale, ossia quella nella quale è possibile accedere a tutte le funzioni del programma; successivamente vengono configurati i gestori e la scuola con i rispettivi dati. Le ultime righe di codice mostrano l'implementazione degli osservatori, dimostrando come il Controller sia il vero intermediario nella logica MVC. Il Listato 5.10 mostra il metodo che il Controller utilizza per eseguire un salvataggio totale ogni 5 minuti, dal momento dell'accensione del programma fino al suo spegnimento.

Observer

Il pattern Observer è un pattern comportamentale che consente di avvisare più oggetti degli eventi che accadono all'oggetto osservato; questa soluzione evita che ci sia un interrogazione continua dello stato di un oggetto, con uno spreco inutile di risorse CPU. L'impiego di questa logica si può notare nelle ultime righe del Listato 5.7; il Controller, infatti, si è aggiunto a una lista di osservatori che vengono notificati riguardo agli eventi che accadono nella vista principale. Il Listato 5.8 , invece, mostra le funzioni di callback presenti nel costruttore della vista principale; questa implementazione consente di mettere in lista le funzioni da richiamare quando accade un determinato evento.

```
class Controller:
   def __init__(self,scuola):

       self.main_window = QtWidgets.QMainWindow()
       self.ui = Ui_MainWindow()
       self.ui.setupUi(self.main_window)

       #crea gestori e carica database
       self.gestoreProf=GestoreProfessore.carica_da_file()
       self.gestoreOrari=GestoreOrari.carica_orari()
       self.scuola=scuola

       # Collega i segnali
       self.ui.pushButton_3.clicked.connect(self.nuovo_orario)
       self.ui.pushButton_5.clicked.connect(self.nuovo_professore)

       for bottone in self.ui.listaProfBtn:
           bottone.clicked.connect(self.show_prof)
       for bottone in self.ui.listaOrariBtn:
           bottone.clicked.connect(self.show_orario)

       #osservatori dei prof e orari
       self.ui.aggiungi_observer(self.show_prof)
       self.ui.aggiungi_observer_elimina(self.elimina_prof)
       self.ui.aggiungi_observer_orario(self.show_orario)
       self.ui.aggiungi_observer_elimina_orario(self.elimina_orario)
```

Listing 5.7: Righe principali del costruttore del Controller

```
   #logica observer per i prof e orari
   def aggiungi_observer(self, callback):
       self.observers.append(callback)
   def aggiungi_observer_elimina(self,callback):
       self.observers_elimina.append(callback)
   def aggiungi_observer_orario(self,callback):
       self.observers_orario.append(callback)
   def aggiungi_observer_elimina_orario(self,callback):
       self.observers_elimina_orario.append(callback)
```

Listing 5.8: Funzioni di callback della vista principale

```
   def back_up(self):
       self.scuola.infoScuola=self.ui.textEdit.toPlainText()
```

```
        self.gestoreProf.salva_su_file()
        self.gestoreOrari.salva_orari()
        self.scuola.salva_su_file()
        t=threading.Timer(300, self.back_up)
        t.daemon = True
        t.start()
```

Listing 5.9: Funzioni di salvataggio di tutti i dati

```
    def back_up(self):
        self.scuola.infoScuola=self.ui.textEdit.toPlainText()
```

Listing 5.10: Funzioni di salvataggio di tutti i dati

5.2 Come funziona l'algoritmo

L'algoritmo per la creazione dell'orario scolastico è il cuore del progetto e la sua implementazione si basa sul diagramma presentato nel capitolo precedente. Quando l'utente decide di compilare automaticamente l'orario, il programma genera un oggetto della classe `DevOrario`; tale oggetto riceve i professori come input dal Controller e restituirà, come output, l'orario completato. La Figura 5.2 mostra la classe `DevOrario` con attributi e metodi, mentre, di seguito, sono elencati i metodi con una breve descrizione:

- `inizializza()`: raccoglie i dati dell'orario di partenza.
- `ordinaProf()`: ordina i professori per complessità decrescente di assegnazione delle ore.
- `genera()`: è la funzione principale che crea e completa l'orario.
- `output()`: restituisce al Controller l'orario completo.
- `ripristina()`: ripristina i dati dei professori.
- `associaOre()`: associa ai professori le lezioni in blocchi da una o due ore.
- `valutaGiorno()`: determina il giorno in cui assegnare una certa lezione a un dato professore.
- `assegnaOre()`: assegna una lezione a un professore.

5.2.1 Il metodo principale

L'algoritmo sfrutta il metodo `genera()` per eseguire tutti i passaggi che portano al completamento dell'orario. Il primo passo è raccogliere e strutturare i dati iniziali tramite i metodi `inizializza()` e `ordinaProf()`; il primo serve a preservare le assegnazioni manuali fatte dall'utente, il secondo ordina i professori in modo che, nel corso dell'assegnazione delle ore, i professori più difficili da sistemare vengano eseguiti subito. Il secondo passo consiste nell'associare i blocchi di lezione ai professori, poiché una lezione, a seconda della materia di insegnamento, può essere di due ore oppure di un blocco di un'ora; inoltre, un professore può insegnare fino a 18 ore a settimana. Il risultato del metodo `associaOre()` è un dizionario in cui i professori, per ogni classe, hanno un numero di lezioni da un'ora e un numero di lezioni da due ore. Una volta che i dati sono stati sistemati, si procede con l'assegnamento delle lezioni; un ciclo `while` itera questo passaggio per 200 volte e, se non si trova una soluzione per

Figura 5.2: Classe DevOrario

l'orario, tale ciclo termina senza fornire un orario al Controller. In queste iterazioni vengono assegnate le lezioni grazie ai metodi `assegnaOre()` e `valutaGiorno()`; l'ordinamento consiste nell'assegnare tutte le lezioni di una classe alla volta. Per ottimizzare le iterazioni, l'ordinamento rispetta i blocchi illustrati nella Figura 5.3.

5.3 Manuale

In questa sezione vengono definite le linee guida per la corretta navigazione del programma.

5.3.1 Login

Per entrare nel programma occorre generare il nome e la password per la prima volta; la Figura 5.4 mostra la vista che ci appare per tale scopo. Tra i campi di inserimento notiamo, oltre al nome e alla password, il campo chiave, necessario per salvare le credenziali. La Figura 5.5 mostra la vista utile all'inserimento dei dati dell'utente. Nel caso fosse necessario ricreare tali dati, cliccando il pulsante `ricrea credenziali` apparirà una finestra che richiede la chiave di autorizzazione, come illustrato nella Figura 5.6.

5.3.2 Finestra principale

La Figura 5.7 mostra la vista principale visualizzabile dopo il corretto inserimento delle credenziali. Sul lato sinistro dello schermo sono visualizzabili i pulsanti di navigazione, mentre al centro è visibile la lista scorrevole dei professori esistenti nel programma.

Informazioni della scuola

La Figura 5.8 mostra la sezione in cui è possibile visualizzare e modificare le informazioni della scuola; tale finestra è accessibile premendo il pulsante `Info Scuola`.

5.3.3 Gestione dei docenti

Per inserire un docente occorre cliccare il pulsante `Nuovo professore`; in questo modo si aprirà la finestra mostrata dalla Figura 5.9. La finestra permette di inserire il nome, le ore

Figura 5.3: Algoritmo di assegnazione delle ore

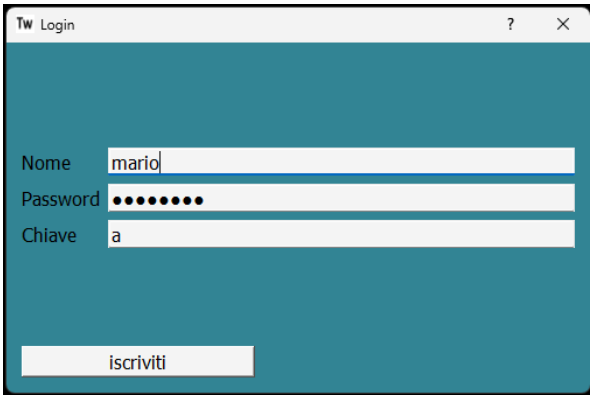

Figura 5.4: Inserimento di nuove credenziali

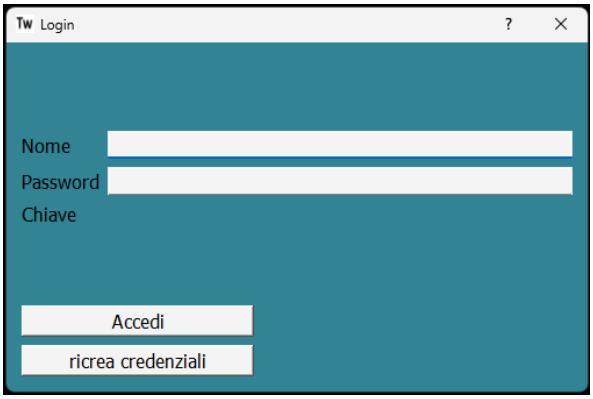

Figura 5.5: Login

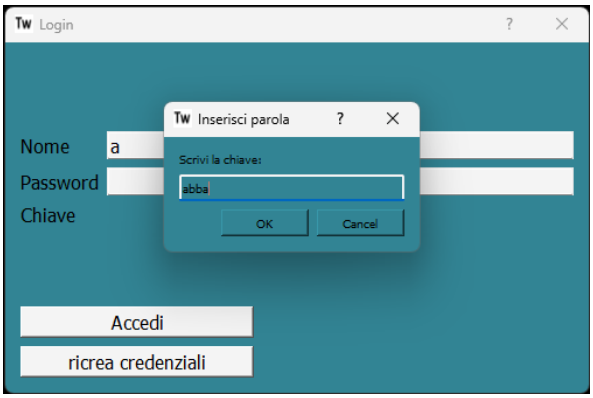

Figura 5.6: Chiave per cambiare le credenziali

Figura 5.7: Vista principale del programma

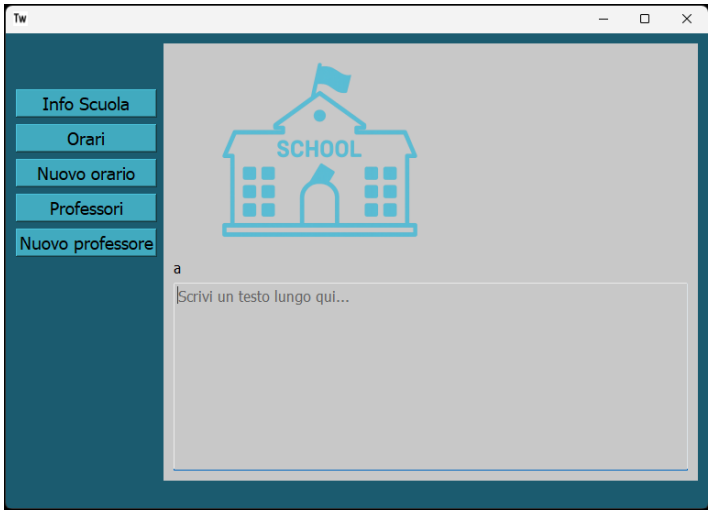

Figura 5.8: Vista relativa alle informazioni della scuola

totali di insegnamento, la materia di insegnamento, le classi, e le ore in cui il docente non può insegnare. Una volta inseriti tutti i dati è possibile, cliccando il pulsante di conferma, uscire dalla finestra e visualizzare il docente nel menù principale. In esso sono visualizzabili i professori che possono essere eliminati tramite il pulsante rosso posto in alto per ogni icona. Quando si clicca un professore verrà mostrata la finestra con le informazioni relative al docente; le Figure 5.10 e 5.11 evidenziano che le modifiche sono possibili se si clicca il pulsante `Abilita modifiche`.

5.3.4 Gestione degli orari

Inserimento di un orario

Per inserire un nuovo orario è necessario cliccare il pulsante `Nuovo orario` situato nel menù principale. La Figura 5.12 mostra la finestra che consente l'inserimento manuale delle ore, oppure la compilazione automatica tramite il pulsante `compila`. Per assegnare manualmente l'ora a un dato professore, basta cliccare una casella e comparirà un menù a tendina con le classi disponibili, come mostrato nella Figura 5.13. Al termine della compilazione sarà

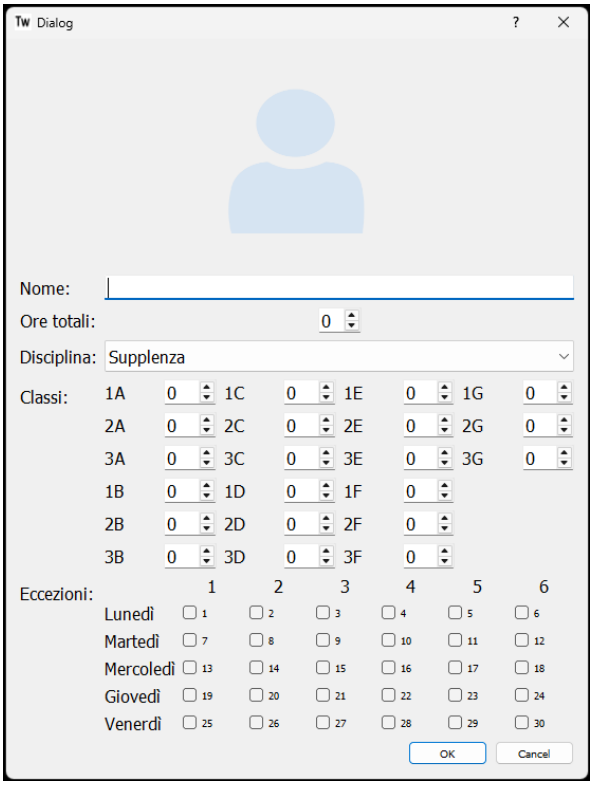

Figura 5.9: Vista relativa all'inserimento di un nuovo docente

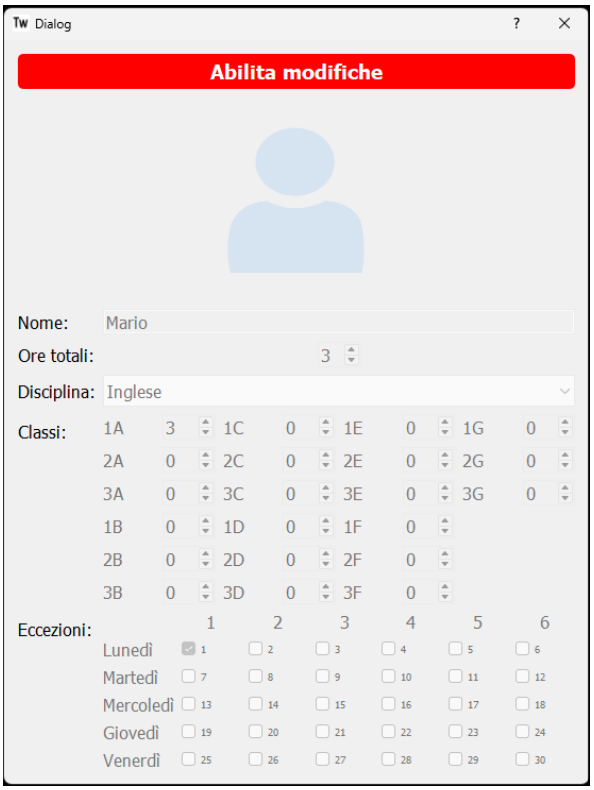

Figura 5.10: Vista relativa alle modifiche di un docente

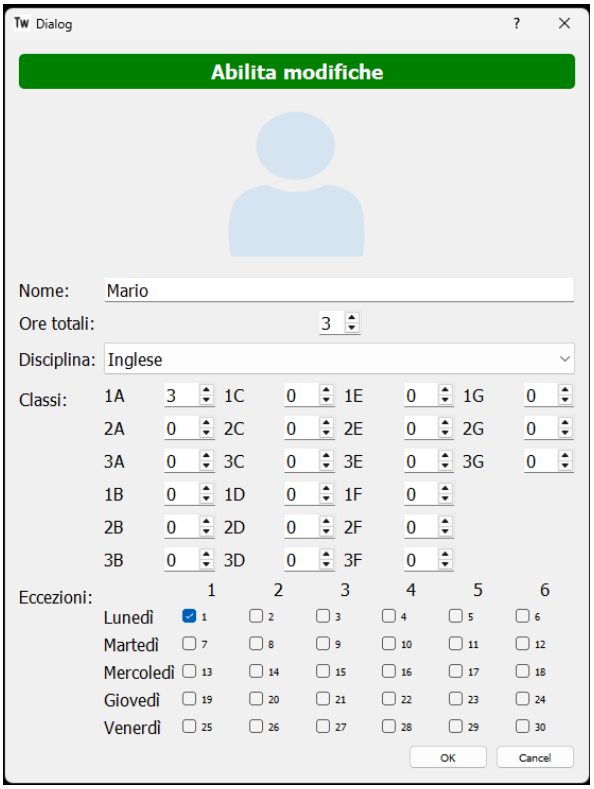

Figura 5.11: Vista relativa alle modifiche di un docente

possibile salvare con un nome l'orario scolastico; tale operazione viene svolta con la finestra mostrata in Figura 5.14.

Visualizzazione e modifica di un orario

Per visualizzare gli orari bisogna premere il pulsante `Orari`, il quale mostrerà una schermata scorrevole con tutti gli orari creati, come si può notare nella Figura 5.15. Premendo su un orario sarà possibile visualizzarlo come in Figura 5.16; la modifica può essere effettuata spuntando la casella posta in alto a destra. Se si vuole eliminare un orario, bisogna premere il pulsante rosso posto in alto a destra dell'icona del dato orario.

Salvataggio dell'orario

Per salvare l'orario in PDF basta cliccare il pulsante `salva in pdf` posto in alto a destra, a fianco alla casella per le modifiche; tale operazione salverà l'orario nell'apposita cartella del programma.

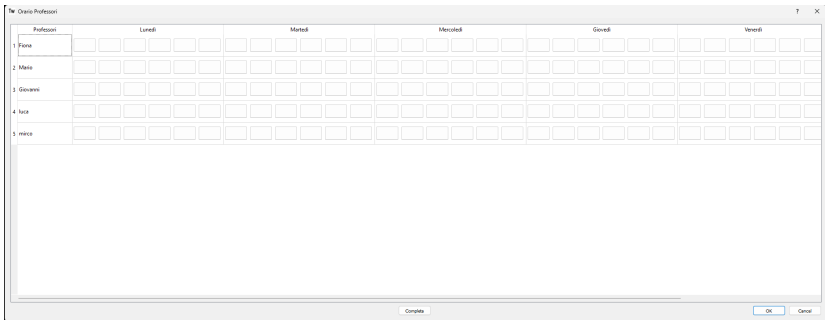

Figura 5.12: Vista relativa alla creazione di un nuovo orario

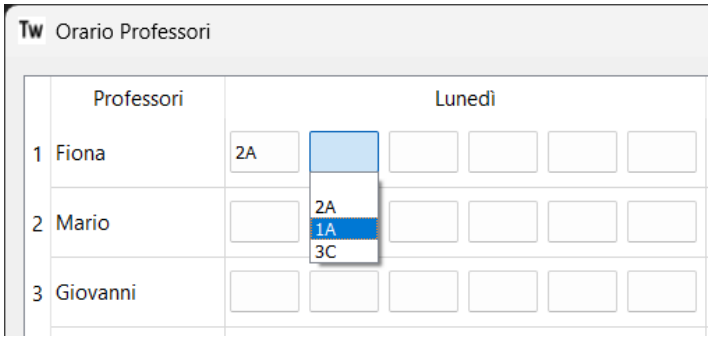

Figura 5.13: Vista relativa alla selezione manuale di assegnazione

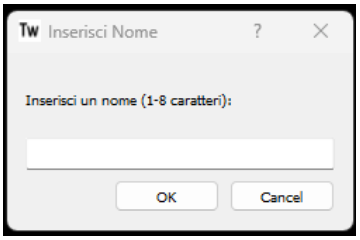

Figura 5.14: Vista relativa all'inserimento di un nome per l'orario

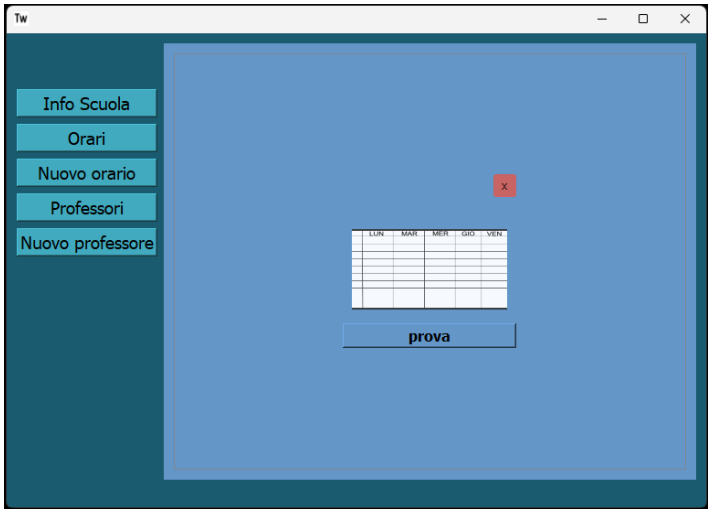

Figura 5.15: Vista relativa alla visualizzazione degli orari

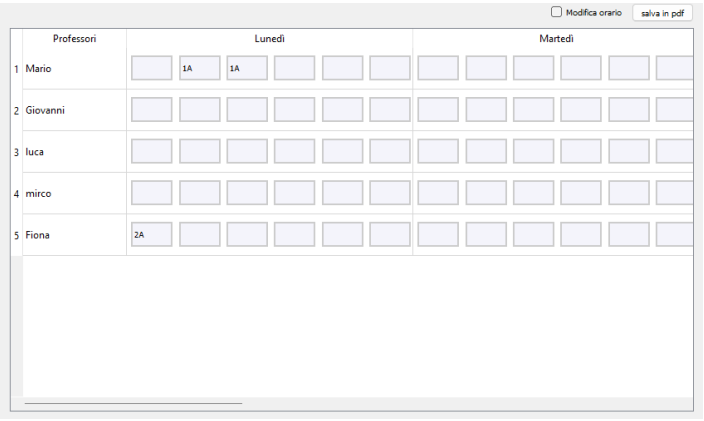

Figura 5.16: Vista relativa alla visualizzazione di un orario

CAPITOLO 6

Conclusioni

L'obiettivo della tesi è stato quello di mostrare il percorso svolto per la progettazione e l'implementazione di un sistema per la gestione di un orario per una scuola media. Il primo passo è stato quello di analizzare un problema che accomuna non solo gli istituti scolastici, ma anche qualsiasi struttura aziendale con molti dipendenti. Abbiamo inquadrato la problematica dell'orario scolastico nell'ambito della Programmazione Lineare, definendo le possibili soluzioni. Nel secondo capitolo abbiamo incanalato tutte le esigenze della scuola racchiudendole in una lista di requisiti, ponendoci l'obiettivo di soddisfarli nelle fasi successive. Nella fase di analisi, abbiamo descritto e illustrato, attraverso diagrammi, le entità principali e il loro funzionamento, concentrandoci maggiormente sull'aspetto gestionale, anziché sull'algoritmo. La fase di progettazione è stata fondamentale per dare una forma più completa a quanto emerso nelle fasi precedenti; inoltre, abbiamo esposto una soluzione per l'algoritmo che ha l'obiettivo di completare l'orario scolastico. Il software, ottenuto al termine della fase di implementazione, è frutto di un lavoro strutturato in fasi propedeutiche, seguendo, come linea guida, le classiche fasi dello sviluppo software. Uno degli aspetti che ha reso più semplice l'implementazione è sicuramente il software QtDesigner, che ha consentito la creazione della GUI in modo più intuitivo, anche per chi non ha grandi conoscenze di tale branca dell'informatica. Il software è funzionale per una scuola media con determinate esigenze, ma la sua struttura può essere migliorata per costruire una soluzione generale che soddisfi le richieste di più istituti di diversa organizzazione e grado. L'algoritmo può essere migliorato con tecniche di ottimizzazione, seguendo la teoria matematica che si cela dietro ai problemi di Programmazione Lineare; inoltre, i diagrammi UML, sviluppati nella fase di analisi e in quella di progettazione, possono sembrare eccessivamente dettagliati per un software di queste dimensioni, ma saranno fondamentali in futuro, specialmente in un'ottica di scalabilità del software.

Bibliografia

ANGERILLI, M. A., GIUSEPPONI, K. e RICCI, G. (2011), *Dirigere una scuola. Gestione, organizzazione, controllo, comunicazione. Manuale per il dirigente della scuola autonoma e per i concorsi*, vol. 41, Maggioli Editore.

FITZPATRICK, M. (2020), *Create GUI Applications with Python & Qt5 (PyQt5 Edition): The hands-on guide to making apps with Python*, Martin Fitzpatrick.

GAMMA, E., HELM, R., JOHNSON, R. e VLISSIDES, J. (1995), *Design patterns: elements of reusable object-oriented software*, Pearson Deutschland GmbH.

GHEZZI, C., JAZAYERI, M. e MANDRIOLI, D. (2004), *Ingegneria del software: fondamenti e principi*, Pearson Italia Spa.

HORSTMANN, C. e NECAISE, R. D. (2014), *Concetti di informatica e fondamenti di Python*, Maggioli Editore.

JOHNSON, R. (2025), *Principles of Model-View-Controller Architecture: Definitive Reference for Developers and Engineers*, HiTeX Press.

LUTZ, M. (2008), *Imparare Python*, Tecniche Nuove.

MOORE, A. D. (2019), *Mastering GUI programming with Python: Develop impressive cross-platform GUI applications with Pyqt*, Packt Publishing Ltd.

SERAFINI, P. (2009), *Ricerca operativa*, Springer Science & Business Media.

VERCELLIS, C. e OTHERS (2008), *Ottimizzazione. Teoria, metodi, applicazioni.*, McGraw-Hill.

ZHU, K., LI, L. D. e LI, M. (2021), «School Timetabling Optimisation using artificial bee colony algorithm based on a virtual searching space method», *Mathematics*, vol. 10 (1), p. 73.

Sitografia

- Github - https://github.com
- HTML.it - https://www.html.it
- Python Documentation - https://docs.python.org
- Qt for Python - https://doc.qt.io/archives/qtforpython-5
- QT Widgets Manual - https://doc.qt.io/qt-6/qtdesigner-manual.html
- Stack Overlflow - https://stackoverflow.com
- Wikipedia - https://it.wikipedia.org
- W3Schools - https://www.w3schools.com

Ringraziamenti

Ringrazio i miei genitori, mio fratello Gioele e i miei parenti per avermi sostenuto in questo lungo percorso. Ringrazio gli amici che mi supportano fin dalla mia infanzia e coloro che si sono inseriti nella mia vita durante questo viaggio. Ringrazio i miei coinquilini che mi hanno accompagnato in questo cammino, lontano da casa. Un ringraziamento va anche a tutte le persone che ho conosciuto all'università, con le quali ho studiato, collaborato e legato molto. Desidero ringraziare anche il mio relatore Prof. Domenico Ursino, per la pazienza e la professionalità che ha mostrato, per i consigli senza i quali non sarei riuscito nella scrittura di questa tesi.