ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

Corso di Laurea in Ingegneria e Scienze Informatiche

Progettazione e realizzazione di un'applicazione web per la gestione dell'orario delle lezioni

Elaborato in
Basi di Dati

Relatore:
Chiar.ma Prof.ssa
Annalisa Franco

Correlatore:
Chiar.mo Prof.
Matteo Ferrara

Presentata da:
Gianluca Bianchi

Sessione I
Anno Accademico 2024-2025

Indice

Capitolo 1

1 Introduzione

La digitalizzazione della pubblica amministrazione rappresenta una sfida strategica e un'opportunità cruciale per l'innovazione dei servizi pubblici, in particolare nel contesto universitario, dove l'efficienza, l'accessibilità e la trasparenza dei processi amministrativi sono fondamentali per sostenere la qualità dell'offerta formativa e della ricerca.

Il suddetto processo di transizione non è solo una questione tecnologica, ma coinvolge aspetti politici e organizzativi: le esperienze più avanzate si riscontrano in contesti guidati da figure competenti, capaci di promuovere un approccio digitale che ripensa i servizi, invece di replicare processi esistenti in formato digitale.

In molti casi, la trasformazione viene ridotta alla pubblicazione di contenuti online o allo sviluppo di portali e App poco efficaci: questa interpretazione parziale ha spesso portato a risultati insoddisfacenti, anche a livello internazionale.

L'obiettivo di questa tesi è quello di migliorare e promuovere il processo di digitalizzazione anche per l'università di Bologna: in particolare si vogliono migliorare e formalizzare tutti i processi relativi alla gestione dell'orario delle lezioni, occupazione aule e disponibilità dei docenti.

Le attività svolgono un ruolo chiave all'interno di una università, tutti questi processi vengono attualmente gestiti tramite un foglio di calcolo: la gestione di questi processi allo stato attuale crea diverse difficoltà nella condivisione e nella modifica dei dati, senza considerare la possibilità di introdurre errori umani ad ogni operazione svolta.

Gestire l'orario e la programmazione delle lezioni di università è un compito estremamente complicato, per questo il corso di laurea ha deciso di intraprendere questo progetto e di standardizzare e automatizzare i processi di gestione dell'orario e della programmazione delle lezioni tramite lo sviluppo di un applicazione web dedicata.

1.1 Glossario

Tabella 1: Glossario

Termine	Definizione
Cdl	Corso di Laurea
Wep App	Applicazione Web
Framework	Insieme di strumenti e regole che facilitano e velocizzano lo sviluppo di applicazioni web.
Admin	Utente con pieni privilegi di gestione e controllo su un sistema o applicazione
SQL	Linguaggio usato per gestire e interrogare database relazionali.
DBMS	Software che crea, gestisce e controlla l'accesso a un database
Home Page	Pagina principale di un sito web che offre all’utente una panoramica iniziale e consente di accedere alle funzionalità e sezioni principali.
Login	Processo di accesso a un sistema informatico mediante inserimento di credenziali (nome utente e password)
Logout	Processo di uscita da un sistema informatico, che termina la sessione utente attiva
Modale	Finestra sovrapposta che richiede interazione prima di tornare al contenuto principale
Form	Insieme di campi per inserire e inviare dati
Backend	Componente server di un'applicazione che gestisce logica, dati e comunicazioni con il frontend
Frontend	Interfaccia utente di un'applicazione
API	Interfaccia che permette a sistemi diversi di comunicare e scambiarsi dati

1.2 Piattaforma per la gestione e programmazione delle lezioni

Attraverso un'apposita interfaccia web, gli operatori della segreteria didattica incaricati della gestione dell'orario e della programmazione delle lezioni potranno consultare e modificare l'orario in modo intuitivo, rapido e strutturato.

La condivisione dei dati tra operatori non rappresenterà più un ostacolo: ciascun utente sarà dotato di credenziali personali per accedere alla piattaforma e visualizzare esclusivamente le informazioni di propria competenza, garantendo così sicurezza e tracciabilità delle operazioni.

Uno degli aspetti centrali dell'applicazione è la gestione avanzata della ricerca dell'orario, resa possibile grazie a filtri specifici che permettono una visualizzazione dettagliata delle lezioni. Questo sistema consente di individuare rapidamente eventuali sovrapposizioni tra insegnamenti, aule disponibili o conflitti legati ai docenti. Una funzionalità di questo tipo rappresenta un supporto fondamentale per la segreteria, semplificando la gestione logistica e operativa dell'orario e riducendo significativamente errori e tempi di pianificazione.

È importante sottolineare come la gestione delle lezioni implichi una serie di attività correlate che ruotano attorno al concetto stesso di "lezione". Tra queste, rientrano:

- Gestione dei docenti, con controllo del monte ore e verifica dell'assenza di sovrapposizioni orarie;
- Gestione delle aule, inclusa la verifica della capienza e la disponibilità oraria;
- Gestione degli insegnamenti;
- Gestione dei Corsi di Laurea;
- Gestione dei curricula;
- Gestione delle classi.

Questi rappresentano solo alcuni dei processi integrati nella gestione delle lezioni. L'obiettivo di questa prima versione della piattaforma è porre le basi per una struttura solida e scalabile, su cui poter sviluppare e formalizzare progressivamente tutti i processi sopra elencati. La versione iniziale dell'applicazione web includerà le funzionalità presenti nella *Tabella 2*.

Tabella 2: Rappresentazione tabellare delle funzionalità del software incluse nella prima versione

Funzionalità	**Area Funzionale**
Visualizzazione dell'orario completo	lezioni
Ricerche con filtri per la visualizzazione dell'orario in base ad appositi criteri	lezioni
Accounting per il personale adibito all'utilizzo della piattaforma	utenti
Creazione di nuovi utenti	utenti
Gestione permessi utente	utenti
Clonazione dell'orario da un anno accademico a quello corrente	lezioni
Vincoli per garantire l'integrità dei dati	lezioni,aule,docenti,insegnamenti

I vincoli per la gestione dell'integrità dei dati sono un aspetto fondamentale di questa piattaforma: quando si realizza una programmazione delle lezioni bisogna tener conto di diversi vincoli per garantire che l'orario prodotto rispetti tutti i criteri impliciti ed espliciti.

I criteri impliciti sono quelli che derivano dalla natura "fisica" dei concetti presenti all'interno dell'orario:

- una stessa aula non può ospitare più lezioni nello stesso giorno e nello stesso orario;
- un'aula non può ospitare lezioni che hanno un numero di partecipanti che supera la capienza dell'aula stessa;
- uno stesso docente non può svolgere contemporaneamente due o più lezioni nello stesso orario.

I criteri espliciti fanno riferimento a quelli forniti dall'Università di Bologna:

- insegnamenti obbligatori di uno stesso anno di un corso di studi non si possono sovrapporre con altri
- controllare che le ore di un Insegnamento siano state raggiunte con la programmazione delle Lezione

- in caso di sdoppiamento (classi), verificare che la somma dei membri di ciascuna classe corrisponda al numero effettivo di iscritti totali

Questi criteri costituiscono un elemento essenziale del sistema, contribuendo in modo significativo al miglioramento del processo di creazione e modifica dell'orario rispetto alla situazione attuale.

Attualmente, infatti, gli operatori sono costretti a svolgere manualmente una serie di verifiche per assicurarsi che ogni operazione sull'orario rispetti tutti i criteri previsti. Questo approccio non solo comporta un notevole dispendio di tempo, ma aumenta anche il rischio di errori, legati alla componente soggettiva dell'intervento umano.

In questo capitolo ci limiteremo a enunciare tali criteri, mentre nel Capitolo 2 essi verranno approfonditi e formalizzati sotto forma di veri e propri vincoli.

1.3 Applicazione Web

Una Web App è un'applicazione accessibile tramite un browser web, come Chrome, Safari o Firefox, senza bisogno di essere scaricata o installata, questo significa che può essere utilizzata da qualsiasi dispositivo connesso a Internet, come computer, tablet o smartphone.

Tra i suoi principali vantaggi ci sono l'accessibilità da remoto, la compatibilità con diversi sistemi operativi e la semplicità di aggiornamento: ogni modifica fatta dagli sviluppatori è subito disponibile per tutti gli utenti.

Inoltre, le Web App non occupano spazio sul dispositivo e offrono un'esperienza d'uso fluida e immediata, rendendole ideali sia per l'utente finale sia per chi le gestisce.

Le applicazioni web si basano sul paradigma “client-server” ed utilizzano il protocollo HTTP a livello Application dello stack ISO.

Quando un utente apre una Web App dal browser, il primo passo è la richiesta al server dove l'app è ospitata: il server risponde inviando i file necessari, come HTML, CSS e JavaScript, che vengono caricati nel browser. A questo punto, l'utente può interagire con l'interfaccia: ogni azione, come cliccare un bottone può generare nuove richieste al server, ad esempio per salvare dati o ricevere informazioni.

Questo scambio continuo tra client (il browser) e server è ciò che permette alla Web App di funzionare in modo dinamico ed efficace.

All’interno di un’applicazione Web sono presenti linguaggi lato client (Javascript), linguaggi lato server (PHP, Java, Perl…), linguaggi di stile (CSS, SCSS) per definire l’aspetto visivo delle pagine web ed un linguaggio di markup standard (HTML5) che viene interpretato dal browser per renderizzare graficamente i contenuti.

I linguaggi lato client vengono interpretati dal browser e permettono di rendere dinamica la pagina lato client: consentono di manipolare la struttura del documento HTML (detta Document Object Model) tramite apposite funzioni.

I linguaggi lato server vengono utilizzati per modificarne il contenuto delle pagine durante l’interazione con il client (ad esempio andato ad effettuare delle ricerche su un database).

Ecco di seguito illustrato, nella *Figura 1*, un esempio del flusso di comunicazione che vi è tra client e server per accedere alla “Home Page” di un tipico sito che utilizza PHP come linguaggio lato server ed un database in cui memorizza i dati:

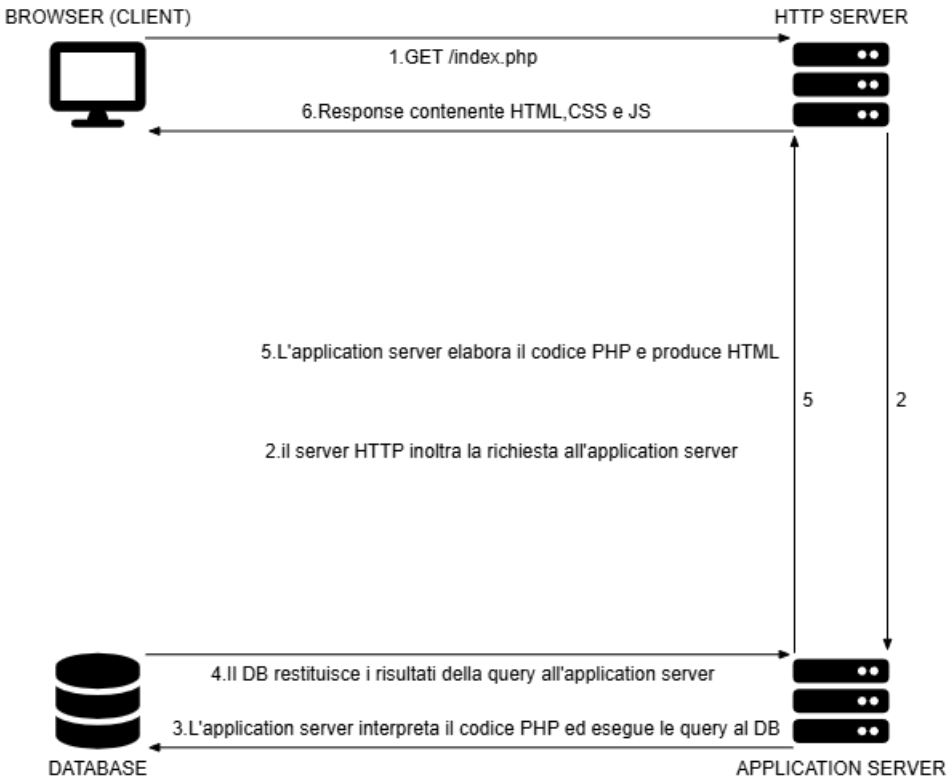

Figura 1: Diagramma per la rappresentazione del flusso di messaggi HTTP per l'utilizzo base di un applicazione Web

Negli ultimi anni, le applicazioni Web hanno conosciuto una notevole evoluzione, influenzando profondamente l’intero settore IT. In particolare, la diffusione di numerosi framework per lo sviluppo Web ha semplificato in modo significativo la creazione e la manutenzione delle applicazioni, rendendole più rapide da sviluppare e più semplici da gestire.

Parallelamente, l'introduzione di nuove tecnologie per il deployment, come Docker, ha reso possibile raggiungere alti livelli di scalabilità, indipendentemente dalla tipologia dell'applicazione.

Nel contesto specifico dell'Università di Bologna, e alla luce dell'esigenza di realizzare una piattaforma per la gestione e la programmazione delle lezioni, lo sviluppo di una Web App rappresenta la soluzione più adatta per rispondere in modo efficace e flessibile a tutte le necessità operative.

Capitolo 2

2 Analisi

La fase di analisi è un pilastro fondamentale nello sviluppo di un software, poiché determina il successo o il fallimento dell'intero progetto. Durante questa fase, gli analisti raccolgono e interpretano i requisiti del cliente, identificando obiettivi, vincoli e requisiti del sistema da realizzare.

Un'analisi accurata evita manutenzioni correttive costose, riducendo il rischio di dover apportare modifiche sostanziali nelle fasi avanzate dello sviluppo, quando i costi e i tempi di correzione sarebbero maggiori. Inoltre, questa fase permette di definire chiaramente l'architettura del software, le funzionalità richieste e i casi d'uso, creando solide basi per la progettazione.

Senza un'analisi approfondita, il team di sviluppo potrebbe lavorare su presupposti errati, portando a un prodotto finale che non soddisfa le esigenze reali degli utenti. Documenti come il Software Requirements Specification (SRS) nascono proprio in questa fase, garantendo tracciabilità e chiarezza.

Investire tempo e risorse nell'analisi significa ottimizzare l'intero ciclo di vita del software, migliorarne la qualità e ridurre i rischi di ritardi o superamenti del budget.

2.1 Contesto Applicativo

Obiettivo di questo lavoro di tesi è lo sviluppo una piattaforma web per la gestione e la programmazione delle lezioni. Ogni operatore incaricato della gestione delle lezioni è dotato di apposite credenziali per accedere all’applicazione.

All'interno della piattaforma sono previsti diversi ruoli utente, ciascuno con specifici livelli di accesso e funzionalità. L'utente base ha la possibilità di visualizzare esclusivamente le lezioni e i relativi dettagli. L'utente admin, oltre a disporre degli stessi privilegi dell'utente base, può anche modificare e creare nuove lezioni. Infine, l'utente super-admin possiede tutti i privilegi dell'admin e

ha, in aggiunta, la facoltà di cancellare le lezioni. Nella *Tabella 3* è possibile visualizzare un riassunto dei principali ruoli con i rispettivi privilegi.

Tabella 3: Rappresentazione tabellare di ruoli e privilegi all'interno dell'applicazione

Ruolo	**Visualizza lezioni**	**Crea/modifica lezioni**	**Crea/modifica utenti e permessi**
base	Sì	No	No
admin	Sì	Sì	No
super-admin	Sì	Sì	Sì

La principale funzionalità richiesta per il sistema è la visualizzazione dell'orario relativo a un determinato semestre e a un preciso anno accademico. A tal fine, si intende storicizzare tutti i dati associati a ciascun anno accademico, consentendo la possibilità di clonarli per l'anno accademico corrente, così da facilitare il riutilizzo e l'aggiornamento delle informazioni nel tempo.

Ogni corso di laurea (CdL) ha una durata annuale, che può essere triennale (tre anni) o magistrale (due anni), e in ogni anno accademico fa riferimento all'anno accademico corrente. Ciascun CdL possiede almeno un curriculum tradizionale, erogato in lingua italiana; tuttavia, alcuni CdL offrono anche un curriculum aggiuntivo in lingua inglese. I diversi curriculum, pur appartenendo allo stesso corso di studio, possono presentare insegnamenti, iscritti e docenti distinti.

Relativamente a un determinato anno di corso, riferito sempre all'anno accademico, sono previsti diversi insegnamenti. Ogni insegnamento è identificato da un nome, una descrizione, i crediti formativi universitari (CFU), il settore scientifico disciplinare (SSD), il docente responsabile, il numero complessivo di ore e l'eventuale suddivisione in moduli. In presenza di moduli, ciascuno di essi è associato a un docente specifico, e dunque l'insegnamento può essere affidato a più docenti. È previsto inoltre che un insegnamento possa essere sdoppiato, il che implica la creazione di due classi distinte tra le quali verranno suddivisi gli studenti iscritti al corso.

Un insegnamento specifico all'interno di un CdL può essere mutuato da un altro CdL o da un curriculum differente dello stesso CdL. In tal caso, l'insegnamento

mutuato mantiene identiche caratteristiche rispetto all'originale. È anche possibile mutuare un singolo modulo di un insegnamento; in questa circostanza, gli insegnamenti coinvolti avranno codici differenti, ma condivideranno un modulo comune.

Un ulteriore aspetto da gestire riguarda gli insegnamenti integrati, ovvero insegnamenti che si compongono di altri insegnamenti distinti. Il codice dell'insegnamento integrato deve necessariamente essere diverso da quello dei singoli insegnamenti che lo compongono.

La funzionalità di ricerca deve essere progettata per consentire l'applicazione di filtri multipli su diverse componenti del sistema, così da permettere agli operatori di individuare in modo preciso sia eventuali disponibilità sia potenziali criticità, come sovrapposizioni di lezioni, docenti, corsi obbligatori di uno stesso curriculum o aule . La ricerca deve supportare la combinazione dei seguenti filtri:

- Anno accademico;
- Corso di Laurea;
- Curriculum;
- Anno corso di laurea;
- Semestre;
- Aule;
- Docenti.

Infine, un elemento centrale nell'analisi riguarda i criteri che regolano la creazione o la modifica delle lezioni. Tali criteri sono distinti in criteri espliciti, ovvero specificamente indicati dall'Università di Bologna, e criteri impliciti, che derivano dalla struttura logica e dalle caratteristiche delle entità coinvolte. Per facilitare il riferimento a tali criteri nel prosieguo del documento, a ciascuno di essi verrà assegnato un nome simbolico, che sarà utilizzato come riferimento univoco nei capitoli successivi. Nella *Tabella 4*, sono mostrati i criteri necessari per l'applicazione corredati dalla descrizione e dalla tipologia (impliciti o espliciti).

Tabella 4: Criteri che regolano creazione e modifica delle lezioni

Nome	**Descrizione**	**Tipo**
UNIQUE_AULA_LEZIONE	una stessa aula non può ospitare più lezioni nello stesso giorno e nello stesso orario	implicito

CHECK_AULA_CAPIENZA	un’aula non può ospitare lezioni che hanno un numero di partecipanti che supera la capienza dell’aula stessa	implicito
UNIQUE_DOCENTE_LEZIONE	uno stesso docente non può svolgere contemporaneamente due o più lezioni nello stesso orario	implicito
CHECK_INSEGNAMENTI_OVERLAPPING	insegnamenti obbligatori di uno stesso anno di un corso di studi non si possono sovrapporre con altri	esplicito
CHECK_INSEGNAMENTO_HOURS_AMOUNT	le ore totali di un insegnamento devono essere raggiunte raggiunte con la programmazione delle lezioni fornita	esplicito
CHECK_CLASSI_MEMBERS	verificare che la somma dei membri di ciascuna classe corrisponda al numero effettivo di iscritti totali	esplicito

2.2 Analisi dei Requisiti

L'analisi dei requisiti è una fase fondamentale nel processo di sviluppo di un sistema software. Consiste nell'identificazione, raccolta e documentazione e validazione delle esigenze specifiche degli stakeholders in merito al sistema da realizzare. Questo processo serve a garantire che il prodotto finale soddisfi gli obiettivi e le necessità per cui è stato concepito.

I requisiti si suddividono principalmente in due categorie: requisiti funzionali e requisiti non funzionali (NFR, Non-Functional Requirements).

I requisiti funzionali descrivono ciò che il sistema deve fare. Indicano le funzioni, i comportamenti e le interazioni che il software deve essere in grado di eseguire. Ad esempio, un requisito funzionale potrebbe essere: “il sistema deve permettere agli utenti di effettuare il login con username e password”.

I requisiti non funzionali, invece, descrivono come il sistema deve funzionare. Riguardano aspetti come la performance, l’usabilità, l’affidabilità, la sicurezza, la scalabilità e la manutenibilità. Ad esempio, un requisito non funzionale potrebbe essere: “il sistema deve rispondere a ogni richiesta dell’utente entro due secondi”.

Entrambe le categorie di requisiti sono essenziali per progettare un sistema efficace: i requisiti funzionali definiscono le capacità del sistema, mentre quelli non funzionali ne determinano la qualità e le caratteristiche operative.

2.2.1 Definizione Requisiti Non Funzionali (NFR)

I requisiti non funzionali di una web application definiscono le caratteristiche qualitative del sistema, ovvero come deve comportarsi piuttosto che cosa deve fare.

Prestazioni e usabilità.
L’applicazione deve garantire tempi di risposta estremamente rapidi, con risposte HTTP nell’ordine dei millisecondi, per assicurare un’interazione fluida e un’elevata usabilità anche in presenza di un alto numero di utenti simultanei, come docenti, studenti e personale amministrativo.

Sicurezza
È essenziale proteggere i dati sensibili degli utenti attraverso tecniche di cifratura, meccanismi di autenticazione sicuri e sistemi di difesa contro attacchi informatici, come SQL injection e cross-site scripting, al fine di tutelare l’integrità e la riservatezza delle informazioni.

Scalabilità e manutenibilità
La piattaforma deve poter gestire un progressivo aumento di utenti, corsi e funzionalità senza richiedere una rivisitazione importante del codice e applicativo e delle logiche di business. Allo stesso tempo, il codice dovrà essere modulare, chiaro e ben documentato, in modo da facilitare interventi futuri di manutenzione o aggiornamento.

Affidabilità e monitoraggio
Il sistema deve funzionare correttamente anche in presenza di errori o comportamenti imprevisti. A tal fine, è necessario implementare strumenti di logging e monitoraggio per rilevare tempestivamente eventuali anomalie e garantire la continuità del servizio.

Compatibilità e disponibilità
L'applicazione deve essere accessibile da una vasta gamma di browser, dispositivi e sistemi operativi, garantendo così una piena portabilità. Inoltre, deve assicurare un'elevata disponibilità del servizio, anche in caso di guasti, grazie a soluzioni di backup e ripristino.

Integrabilità
Un aspetto chiave della piattaforma è la sua capacità di integrarsi con altri sistemi, come software gestionali o servizi esterni. In particolare, deve permettere ai servizi dell'Università di Bologna di accedere a dati sempre aggiornati e coerenti relativi alle lezioni e a tutte le entità collegate, favorendo uno scambio informativo efficiente e affidabile.

Concorrenza
L'applicazione dovrà supportare l'utilizzo simultaneo da parte di più account, garantendo l'accesso concorrente alle funzionalità del sistema. Inoltre, dovrà assicurare in ogni momento la coerenza e l'integrità dei dati, anche in presenza di operazioni effettuate contemporaneamente da utenti diversi.

Database
È necessario adottare un database per memorizzare in modo strutturato tutte le informazioni relative agli utenti, alle lezioni e alle entità ad esse collegate, garantendo così un accesso efficiente e coerente ai dati.

Sessione
La durata di default della sessione di un utente dedicato è pari a 120 minuti (2 ore); se l'utente non esegue operazioni per un tempo pari o superiore alla durata di default della sessione, verrà automaticamente scollegato dall'applicazione.

2.2.2 Definizione Requisiti Funzionali

I requisiti funzionali descrivono cosa deve fare un sistema: le funzionalità, i comportamenti e le interazioni previste in risposta alle azioni degli utenti o ad altri input.

All’interno del processo di sviluppo di un’applicazione, i requisiti funzionali rappresentano i casi d’uso.

Di seguito vengono elencati tutti i requisiti funzionali, ciascuno accompagnato da un nome identificativo univoco, in modo da poterli facilmente richiamare e collegare al diagramma dei casi d’uso riportato nel Capitolo 3.

Login
Pagina di autenticazione in cui gli utenti inseriscono le proprie credenziali per accedere all’area privata dell’applicazione.

Logout
L’applicazione deve consentire agli utenti autenticati di disconnettersi tramite un apposito bottone.

Visualizza_Utenti
Gli utenti con ruolo di “super-admin” possono accedere all’area “impostazioni” dell’applicazione, all’interno della quale possono visualizzare tutti gli utenti del sistema con i relativi ruoli associati.

Crea_Utente
Gli utenti con ruolo “super-admin” hanno la possibilità di creare un nuovo utente del sistema tramite un apposito form presente all’interno dell’area “impostazioni” contente i seguenti campi:

- email
- password
- name
- ruolo (con possibilità di specificarne uno tra i seguenti: “user”,”admin”,”super-admin”)

Modifica_Utente
Gli utenti con ruolo “super-admin” hanno la possibilità di modificare i dati di un determinato utente del sistema tramite un form presente all’interno dell’area “impostazioni” contenente i seguenti campi:

- password
- name
- ruolo (con possibilità di specificarne uno tra i seguenti: “user”,”admin”,”super-admin”)

Elimina_Utente

Gli utenti con ruolo “super-admin” hanno la possibilità di eliminare un determinato utente del sistema tramite un apposito controllo presente all’interno dell’area “Impostazioni”.

Visualizza_Orario

Questa funzionalità rappresenta uno degli aspetti centrali dell’applicazione: permette agli utenti autenticati di visualizzare l’orario delle lezioni in formato tabellare. La struttura con cui vengono presentate le informazioni seguirà il formato adottato nel file Excel utilizzato dalla Segreteria di Unibo, Sede di Cesena mostrato nella *Figura 2*.

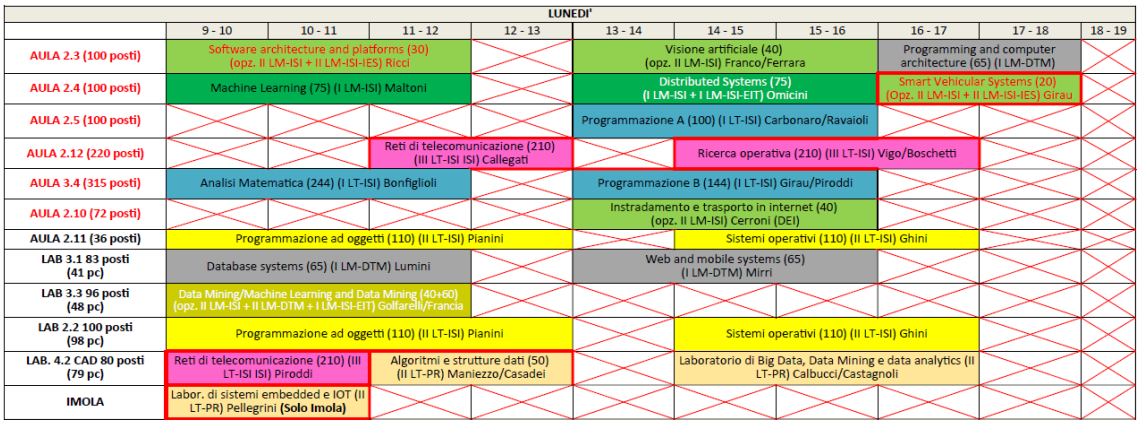

Figura 2: Foglio Excel per la visualizzazione delle lezioni, Segreteria Unibo Sede Cesena

Come si può notare, l’orario è organizzato in una tabella dedicata per ciascun giorno della settimana, riferita a uno specifico semestre. Le colonne indicano le fasce orarie, suddivise in intervalli di un’ora, mentre le righe corrispondono alle aule. In questo schema, ogni cella contiene le informazioni relative all’insegnamento programmato, specificando l’orario coperto (in base al numero di colonne occupate) e l’aula assegnata (in base alla riga). Di default verranno mostratele lezioni del lunedì del primo semestre riferito all’anno accademico corrente. Tramite degli appositi controlli, sarà possibile navigare tra i vari giorni della settimana e modificare semestre ed anno accademico.

Applica_Filtri_Ricerca

Tramite questa funzionalità, integrata all’interno della pagina di visualizzazione dell’orario, ogni utente deve poter applicare filtri di ricerca specifici per personalizzare la consultazione delle lezioni.
I seguenti filtri sono considerati necessari:

- CdL

- Anno di Cdl
- Anno Accademico
- Semestre
- Curriculum
- Docente
- Aula

Visualizza_Lezione
L'applicazione deve consentire di visualizzare i dati dettagliati di ciascuna lezione a partire dalla pagina di visualizzazione dell'orario, comprendendo:

- Orario lezione
- Semestre
- Anno accademico
- Insegnamenti associati
  Docenti
- Aula (o aule nel caso in cui l'Insegnamento si tenga allo stesso orario in 2 aule diverse)

Applica_Vincoli
Questa funzionalità rappresenta un ulteriore aspetto fondamentale dell'applicazione: ad ogni operazione che comporta una modifica dell'orario devono essere verificati tutti i criteri definiti nella *Tabella 4*. Nel caso in cui un' operazione scateni il fallimento della verifica di un determinato criterio, l'applicazione dovrà mostrare a video un messaggio di errore che specifichi i dettagli del criterio non verificato.

Modifica_Orario
Tramite un apposito form presente all'interno dell'area di visualizzazione dell'oraio, gli utenti super-admin saranno in grado di modificare i dati di una determinata lezione. Una volta selezionata una determinata lezione nell'orario, sarà possibile modificare i dati di una lezione tramite un apposito form contenente i seguenti campi:

- Aula/Aule
- Ora di inizio Lezione
- Ora di fine Lezione
- Semestre
- Giorno

Clona_Orario
L’applicazione deve consentire agli utenti con ruolo di “super-admin”, di poter clonare l’orario delle lezioni da un determinato anno accademico a quello corrente. Un altro aspetto fondamentale di questo requisito riguarda il metodo di clonazione dell’orario: il nuovo orario non dovrà contenere riferimenti ad entità appartenenti all’orario clonato, ma dovrà necessariamente eseguire una copia dei dati contenere riferimenti con questi ultimi**.**

Capitolo 3

3 Progettazione

La progettazione rappresenta un momento centrale nello sviluppo di un'applicazione web, in quanto incide direttamente sulla qualità e sull'affidabilità del sistema. All'interno di questa fase, la definizione dell'architettura del database assume un ruolo determinante: da essa dipende non solo la coerenza dei dati gestiti, ma anche le modalità con cui essi vengono archiviati, recuperati e aggiornati.

Una modellazione ben strutturata consente di rendere più efficiente l'interazione tra i vari componenti dell'applicazione, migliorando le performance e facilitando l'integrazione di nuove funzionalità nel tempo. La capacità di supportare carichi di lavoro crescenti e di adattarsi a scenari in evoluzione dipende in larga misura dalla qualità del disegno logico e fisico del database.

Per questo motivo, investire attenzione e rigore nella fase progettuale significa porre le basi per uno sviluppo sostenibile, in grado di assicurare sia la manutenibilità del codice che la stabilità del sistema informativo nel suo complesso.

3.1 Definizione Perimetro Soluzione

L'applicazione web dovrà soddisfare tutti i requisiti funzionali definiti nella sezione *2.1* ponendo particolare attenzione anche ai requisiti non funzionali definiti nella sezione *2.2*.

Nella prima versione dell'applicazione, il focus si pone sulle operazioni di visualizzazione orario e di ricerca: queste 2 operazioni sono fondamentali in quanto permettono agli utenti del sistema di individuare criticità nell'orario e di effettuare ricerche estremamente granulari in modo tale da ricavare tutte le informazioni necessarie in maniera rapida e dettagliata.

Un altro aspetto fondamentale dell'applicazione riguarda la gestione delle utenze: gli utenti "superadmin" devono essere in grado di poter gestire tutti gli utenti del sistema con possibilità di modifica, creazione ed eliminazione. Trattandosi di

un’applicazione che verrà utilizzata internamente da Unibo, non dovranno essere implementati meccanismi di registrazione, in quanto dovranno essere gli utenti “superadmin” del sistema a registrare nuove utenze.

L’obiettivo di questa prima versione dell’applicazione è quello di formalizzare tutti gli elementi che costituiscono l’orario delle lezioni all’interno di un database e fornire delle prime funzionalità di visualizzazione e modifica, in modo tale da realizzare una base solida e scalabile dalla quale partire per realizzare tutte le funzionalità necessarie.

3.2 Tecnologie Adottate

Nel panorama dello sviluppo web contemporaneo, le tecnologie utilizzate per la realizzazione di applicazioni sono in continua evoluzione e riflettono esigenze sempre più complesse in termini di scalabilità, modularità e prestazioni. Tra gli approcci consolidati si colloca lo stack LAMP, basato su Linux, Apache, MySQL e PHP, che continua a essere una scelta solida per progetti che richiedono stabilità e un ambiente maturo. Questo modello privilegia una separazione netta tra il backend e la gestione dei dati, con un’architettura tradizionale server-side.

In contrapposizione, lo stack MEAN, composto da MongoDB, Express.js, Angular e Node.js, rappresenta una soluzione più recente e orientata allo sviluppo full JavaScript, consentendo una maggiore uniformità tra frontend e backend. La differenza principale tra i due approcci risiede non solo nella scelta dei linguaggi e dei database, ma anche nel paradigma adottato: MEAN predilige applicazioni single-page e asincrone, mentre LAMP si adatta meglio a modelli sincroni più lineari.

Inoltre, vi è una grossa differenza tra MySql e MongoDB: quest’ultimo, a differenza dei database relazionali, memorizza i dati in formato JSON flessibile e permette una modellazione più dinamica, adatta a contesti in cui la struttura dell’informazione può variare nel tempo. La differenza tra i due paradigmi di persistenza dei dati emerge chiaramente nel modo in cui gestiscono la scalabilità e la normalizzazione: i sistemi relazionali puntano sulla normalizzazione dei dati ed e il rispetto delle proprietà ACID per le transazioni, mentre quelli documentali privilegiano la velocità di accesso e l’elasticità.

Considerata la natura delle funzionalità previste per l’applicazione, lo stack LAMP risulta essere la scelta più adeguata. La necessità di mantenere una netta separazione tra la logica backend e la gestione dei dati consente di progettare API

scalabili e facilmente manutenibili. Inoltre, l'architettura dell'applicazione non prevede l'utilizzo di una Single Page Application, poiché si rende opportuno disporre di pagine distinte per operazioni specifiche, ciascuna con un proprio contesto funzionale. Un ulteriore elemento che orienta verso questa soluzione è la centralità attribuita all'integrità e alla coerenza dei dati, requisiti che trovano risposta più efficace nei sistemi relazionali come MySQL, propri dello stack LAMP.

3.3 Progettazione Database

La progettazione di un database rappresenta una fase cruciale nello sviluppo di un sistema software, in quanto incide direttamente su elementi fondamentali come la coerenza dei dati, l'efficienza delle operazioni e la scalabilità dell'applicazione. Una corretta progettazione e gestione delle informazioni consente non solo di garantire l'affidabilità del sistema nel tempo, ma anche di agevolarne l'evoluzione e la manutenzione.

Nel contesto dei database relazionali, la progettazione si articola in diverse fasi successive. Si parte dalla progettazione concettuale, che mira a rappresentare le informazioni in modo astratto, indipendente da qualunque vincolo tecnologico. In questa fase viene elaborato un diagramma ER (Entity-Relationship), che descrive entità, attributi e relazioni all'interno della realtà d'interesse definita durante la fase di analisi.

Il passaggio dalla progettazione concettuale a quella logica è un aspetto particolarmente rilevante del processo: il modello ER viene trasformato in uno schema relazionale, in cui le entità diventano tabelle, gli attributi si traducono in colonne con tipi di dato ben definiti, e le relazioni vengono rappresentate tramite chiavi esterne e tabelle associative nei casi di cardinalità molti-a-molti. Questa traduzione non è sempre "meccanica", ma richiede un'attenta riflessione per preservare le semantiche informative originali e garantire la normalizzazione dello schema, così da evitare ridondanze e anomalie nei dati; tenendo in considerazione anche aspetti legati all'efficienza delle operazioni. Durante questa fase, vengono inoltre formalizzati i vincoli di integrità e le chiavi primarie, che svolgono un ruolo centrale nel mantenimento della coerenza logica del sistema.

Con la progettazione logica completata, si passa infine alla fase fisica, che consiste nell'implementazione del modello logico utilizzando un linguaggio di definizione dei dati compatibile con il DBMS scelto. Nel nostro caso, il modello sarà realizzato tramite codice SQL conforme a MySQL. La progettazione fisica

tiene conto anche di aspetti legati all'efficienza, come l'ottimizzazione dell'accesso ai dati, l'uso di indici e la definizione di strategie di memorizzazione. Nelle sezioni successive verranno dettagliate le varie fasi di progettazione per realizzare il database dell'applicazione per la programmazione e gestione delle lezioni.

3.3.1 Progettazione Concettuale

La progettazione concettuale di un database è la prima fase del processo di modellazione dei dati e ha lo scopo di rappresentare in modo chiaro e completo le informazioni rilevanti per l'applicazione. In questa fase si analizza il dominio applicativo per identificare le entità principali, i loro attributi e le relazioni che intercorrono tra di esse, costruendo un modello astratto che descrive la struttura dei dati.

Questo modello viene solitamente rappresentato attraverso un diagramma ER, uno strumento che aiuta a visualizzare la struttura del database e a facilitare la comunicazione tra analisti, sviluppatori e altri stakeholder coinvolti nel progetto. La progettazione concettuale è indipendente dal tipo di database o di tecnologia che verrà utilizzata, e viene utilizzato come punto di partenza per le fasi successive della progettazione logica e fisica.

In questa fase vengono analizzati in dettaglio i requisiti individuati durante la fase di analisi, organizzandoli in diverse "viste" relative al livello dei dati. Per ciascuna vista verranno progettate le entità e le relazioni emerse, al fine di definire in modo strutturato il modello informativo del sistema.

In particolare, abbiamo 6 viste del sistema:

- Lezioni: comprende tutti gli elementi connessi al concetto di "lezione", inclusi orari, docenti, contenuti e modalità di erogazione.
- Insegnamenti: include tutte le componenti che caratterizzano i singoli insegnamenti, come titoli, codici, docenti responsabili e programmi.
- Classi: rappresenta le entità che formalizzano il concetto di "classe", ovvero i gruppi di studenti associati a un Insegnamento eventualmente sdoppiato in più classi.
- Cdl: aggrega gli elementi che descrivono la struttura e l'organizzazione dei diversi corsi di laurea, come piani di studio, anni di corso e insegnamenti previsti.

- Aule: riguarda tutte le informazioni relative agli spazi fisici in cui si svolgono le attività didattiche, comprese le aule e le sedi.
- Sistema: comprende le entità necessarie per la gestione e l'accesso al sistema informativo, come utenti, ruoli e permessi.

3.3.1.1 Vista Cdl

Ciascun CdL è composto da uno o più curriculum: ciascun curriculum rappresenta l'insieme strutturato di insegnamenti e attività formative che uno studente può seguire all'interno di un CdL, spesso corrispondente a un percorso tematico o specialistico erogato in una lingua specifica. Indica gli insegnamenti obbligatori, opzionali e gli obiettivi formativi specifici.

Ciascun curriculum racchiude al suo interno più anni di corso: ciascun anno di corso riferito ad uno specifico curriculum, ha un numero di iscritti per un determinato anno accademico.
A partire da questi requisiti sono state individuate le seguenti entità:

- CdL: rappresenta un corso di laurea identificato da un codice specifico e articolato in uno o più curriculum.
- Curriculum: rappresenta un percorso formativo all'interno di un CdL, composto da più anni accademici.
- AnnoCdl: rappresenta un singolo anno (es. 1°, 2°, 3°) di un determinato curriculum all'interno di un CdL.
- AnnoCdlInAnnoAccademico: rappresenta un preciso anno di un curriculum in uno specifico anno accademico, e consente di memorizzare gli iscritti a ogni anno del curriculum per ciascun anno accademico.
- AnnoAccademico: rappresenta un anno accademico specifico (es. 2023/2024).

Di seguito, nella *Figura 3*, è mostrato il diagramma ER completo inerente alla vista "CdlL".

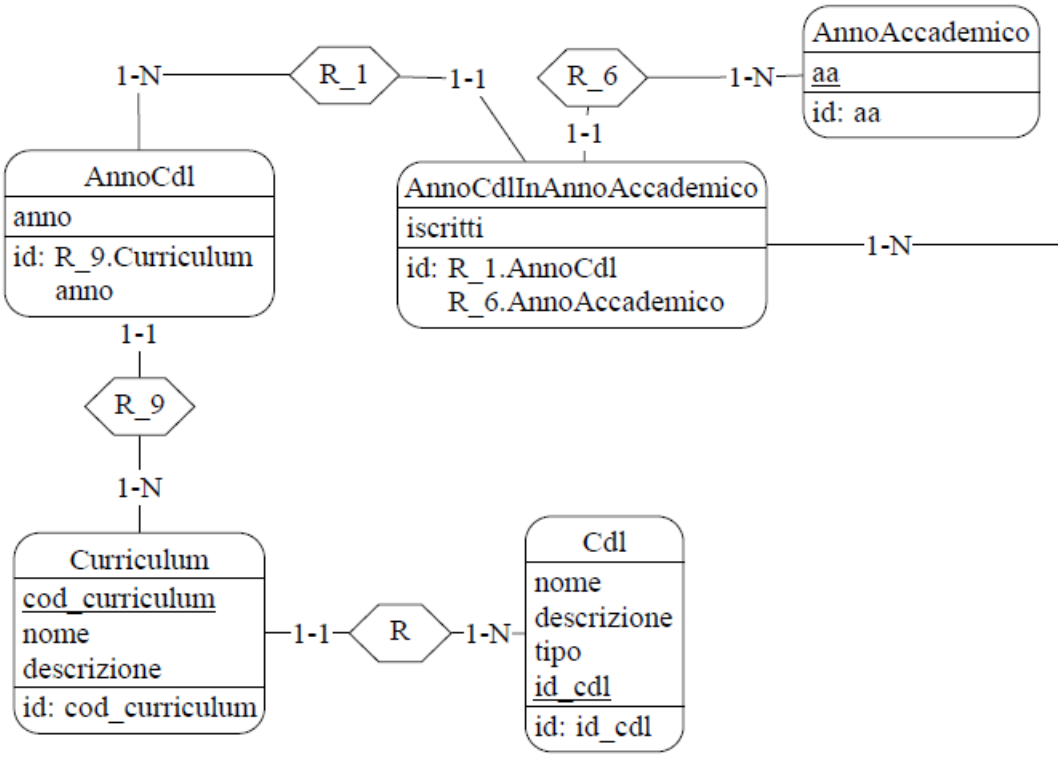

Figura 3: Diagramma ER inerente alla vista "Cdl"

3.3.1.2 Vista Insegnamenti

Ogni anno di corso, all'interno di un determinato curriculum, comprende una serie di insegnamenti che possono variare a seconda dell'anno accademico. Ciascun insegnamento è caratterizzato da un codice identificativo univoco, un nome, una descrizione sintetica e può essere articolato in due o più moduli. Gli insegnamenti sono distribuiti tra il primo e il secondo semestre dell'anno accademico di riferimento e sono corredati da informazioni quali il numero di crediti formativi universitari (CFU), il settore scientifico-disciplinare (SSD), il monte ore complessivo, le ore di laboratorio e quelle dedicate alle esercitazioni. Ogni insegnamento è classificato come obbligatorio o complementare: in particolare, quelli obbligatori appartenenti allo stesso curriculum non devono mai sovrapporsi nell'orario delle lezioni.

Nel caso in cui l'insegnamento preveda più moduli, ciascuno di essi sarà assegnato a un docente specifico e presenterà un proprio numero di CFU, un SSD di riferimento, un totale di ore, un numero definito di ore di laboratorio ed

esercitazioni. Alcuni insegnamenti possono essere mutuati o integrati. Gli insegnamenti mutuati si distinguono in due tipologie: l'intero insegnamento mutuato, che viene incluso senza modifiche in un altro curriculum, oppure il modulo mutuato, in cui una o più parti dell'insegnamento è condivisa con un curriculum differente. L'insegnamento integrato, invece, è costituito da più insegnamenti distinti, che possono appartenere allo stesso curriculum o a curricula differenti.

È rilevante notare che uno stesso Insegnamento, pur mantenendo nome, codice e descrizione invariati, può assumere caratteristiche differenti in termini di CFU, numero di ore, docenti e struttura modulare, a seconda del curriculum in cui è inserito. Inoltre, in alcuni casi l'insegnamento può essere sdoppiato in più classi, ognuna con un proprio corpo docente. Se l'insegnamento sdoppiato è articolato in più moduli, ciascuna classe disporrà dei propri moduli con relative ore e docenti assegnati.

Dopo aver esaminato attentamente tutti requisiti inerenti agli insegnamenti, sono state estrapolate le seguenti entità:

- Insegnamento: rappresenta un insegnamento a livello generale, identificato da un codice, una descrizione e un settore scientifico-disciplinare, senza dettagliare moduli o CFU.
- InsegnamentoCdl: rappresenta l'assegnazione di un insegnamento a uno specifico anno di un curriculum, in un determinato anno accademico.
- Modulo: rappresenta un singolo modulo didattico, parte di un Insegnamento.
- ModuliInsegnamenti: definisce l'associazione tra insegnamenti e i moduli che li compongono; consente di collegare ciascun modulo ai relativi insegnamenti.
- Docente: rappresenta un docente con i relativi dati identificativi.
- MutuazioniModuli: rappresenta la mutuazione di un modulo, ovvero il riutilizzo di un modulo da parte di un insegnamento diverso da quello a cui è originariamente associato.

Di seguito, nella *Figura 4*, è mostrato il diagramma ER completo inerente alla vista “insegnamenti”.

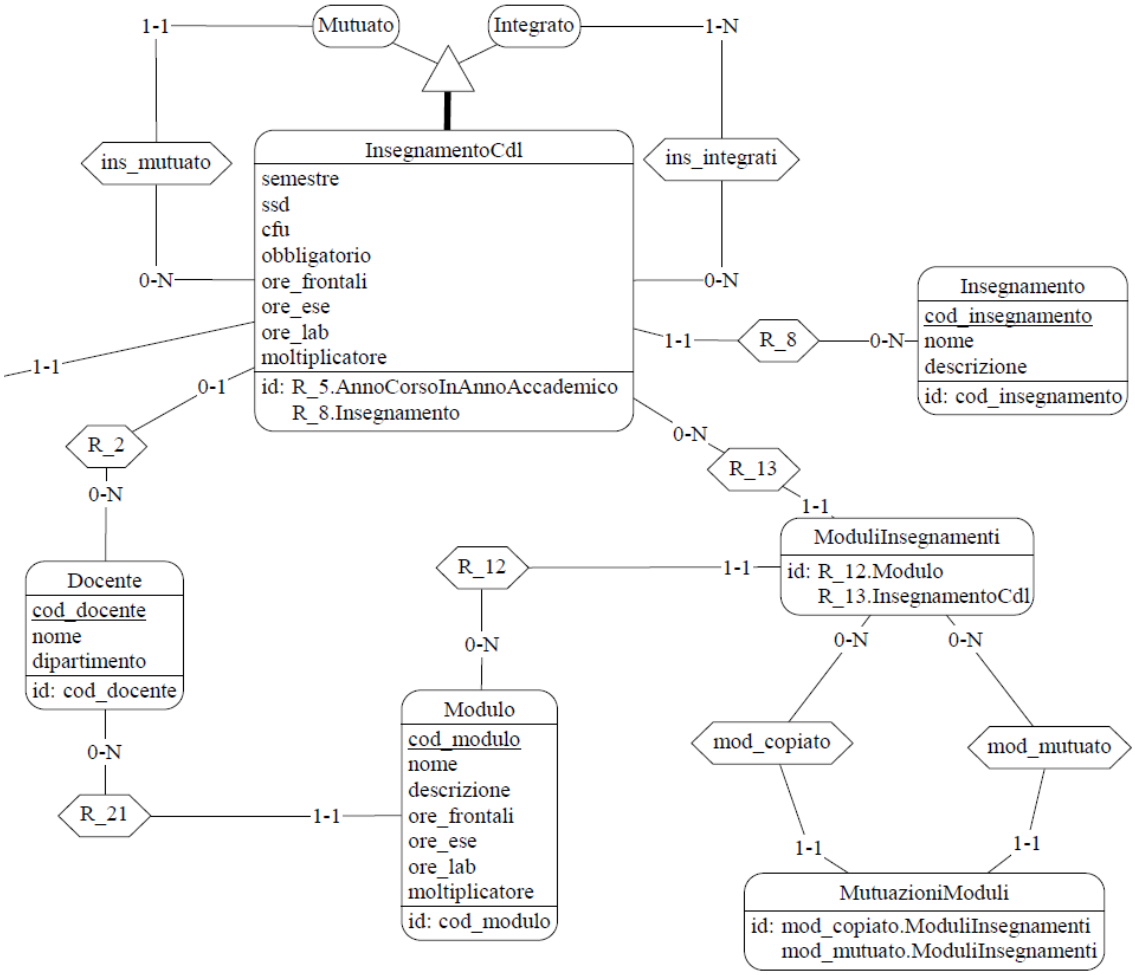

Figura 4: Diagramma ER inerente alla vista "insegnamenti"

Nel contesto della gestione degli insegnamenti universitari, è fondamentale disporre di un modello informativo in grado di rappresentare tutte le variazioni di un determinato insegnamento nel tempo. A tal fine, è stata introdotta l'entità “InsegnamentoCdl”, concepita per identificare una specifica istanza di insegnamento, associata a un determinato anno accademico, a uno specifico anno di corso e a un preciso curriculum. Questo approccio consente di monitorare in modo coerente le evoluzioni che un insegnamento può subire nel corso degli anni.

L’articolazione degli insegnamenti in moduli è gestita attraverso l’entità “ModuliInsegnamenti”, che permette sia di suddividere un insegnamento in più componenti distinte, sia di condividere un singolo modulo tra più insegnamenti appartenenti a corsi diversi. Questo modello assicura la flessibilità necessaria per rappresentare con efficacia sia le articolazioni interne di un insegnamento, sia i meccanismi di condivisione tra percorsi formativi differenti, come nel caso dei moduli mutuati.

Per quanto riguarda gli insegnamenti integrati, si è optato per l’introduzione della relazione “InsegnamentiIntegrati”, che collega un Insegnamento “padre” a due o più insegnamenti “figlio”. In tal modo, è possibile modellare in maniera esplicita la struttura degli insegnamenti integrati previsti dai vari curricula, rendendo il sistema informativo coerente con le esigenze didattiche.

La gestione degli insegnamenti mutuati avviene attraverso un’associazione ricorsiva opzionale sull’entità “InsegnamentoCdl”, che consente di specificare, per ciascun insegnamento mutuato, il riferimento all’insegnamento originario da cui è stato mutuato. Qualora quest’ultimo fosse suddiviso in moduli, l’insegnamento mutuato ne eredita direttamente la struttura modulare. Il compito di generare automaticamente gli insegnamenti mutuati, replicando dati e moduli del corrispondente insegnamento di origine, verrà gestito a livello applicativo per garantire un buon livello di generalità del modello.

L’intero modello è stato pensato per garantire un alto livello di generalità, al fine di poter accogliere eventuali evoluzioni dei requisiti funzionali senza necessità di intervenire sulla struttura concettuale del sistema. In questo modo, l’introduzione di nuove funzionalità potrà avvenire direttamente a livello applicativo, preservando l’integrità e la coerenza del modello.

Infine, ogni insegnamento è sempre associato a un curriculum specifico. Questa scelta riflette la necessità di distinguere tra insegnamenti che, pur condividendo codice, nome e descrizione, possono variare nei contenuti o nella strutturazione a seconda del curriculum di riferimento, anche senza passare attraverso i meccanismi della mutuazione.

3.3.1.3 Vista Classi

Un concetto rilevante da implementare riguarda lo sdoppiamento in classi di alcuni insegnamenti: un determinato Insegnamento di uno specifico curriculum può essere suddiviso in 2 classi; ciascuna classe avrà il relativo docente

associato. Nel caso in cui l'insegnamento sia suddiviso in moduli, ciascuna classe avrà i relativi moduli associati con il rispettivo docente.

Dopo aver esaminato attentamente tutti i requisiti inerenti al concetto di "classe" sono state individuate le seguenti entità:

- ClasseInsegnamento: rappresenta una suddivisione di un insegnamento in più classi, ciascuna identificata da un nome specifico (es. "Classe A", "Classe B") ed eventualmente associata a uno o più moduli didattici.

Di seguito, nella *Figura 5*, è mostrato il diagramma ER inerente alla vista "classi".

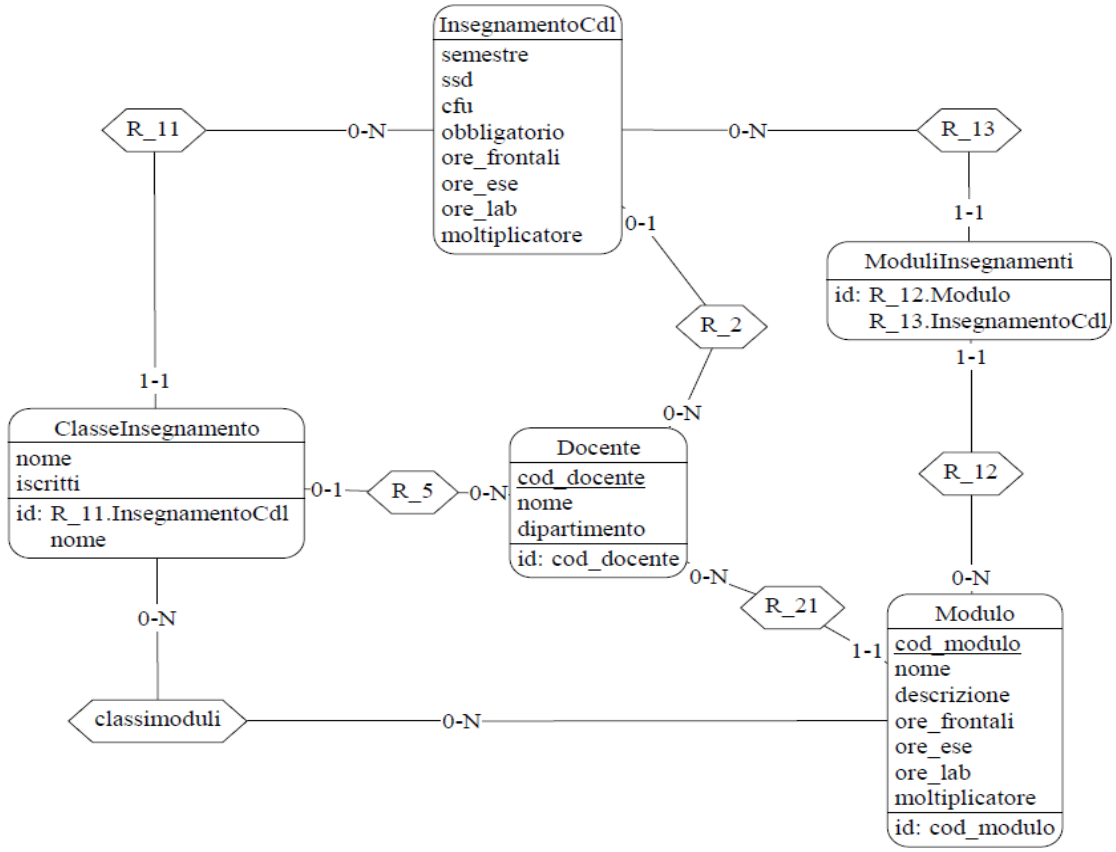

Figura 5: Diagramma ER della vista "classi"

3.3.1.4 Vista Lezioni

Le lezioni si tengono durante i semestri dell'anno accademico, secondo una programmazione che prevede giorni e orari definiti. Possono svolgersi in una o più aule contemporaneamente, qualora siano previsti più turni per lo stesso Insegnamento. In tali casi, pur trattandosi della medesima unità didattica, ciascuna aula accoglie la lezione tenuta da un docente diverso: tale modalità non corrisponde a uno sdoppiamento in classi ma rappresenta una suddivisione della lezione in turni.

Ogni lezione può riferirsi all'intero insegnamento, a una sua articolazione per classe o a uno dei suoi moduli. Nel caso di insegnamenti mutuati o integrati, la lezione deve includere tutti i moduli coinvolti. Una singola lezione può quindi risultare associata a più insegnamenti nei casi in cui vi siano mutuazioni, a un insegnamento specifico previsto per una determinata classe, oppure a un modulo particolare.

Ogni aula è identificata univocamente dal proprio numero all'interno della sede di appartenenza. Per ciascuna aula si intende memorizzare il numero identificativo e il numero complessivo di postazioni disponibili. Relativamente alle sedi, si prevede di conservare le informazioni relative al codice identificativo, all'indirizzo e alla denominazione.

Nel caso in cui un'aula sia classificata come laboratorio, oltre al numero totale di postazioni, viene registrato anche il numero di postazioni dotate di PC. Questa distinzione si rende necessaria poiché in alcuni laboratori possono essere presenti postazioni non attrezzate con computer.

Dopo aver analizzato dettagliatamente i requisiti inerenti al concetto di "Lezione" sono state individuate le seguenti entità:

- Lezione: rappresenta un'attività didattica programmata in un determinato giorno e orario, svolta in un'aula specifica, valida per uno o più insegnamenti, e riferita a un semestre e ad un anno accademico.
- Sede: rappresenta una delle sedi dell'Università di Bologna in cui si svolgono lezioni.
- Aula: rappresenta uno spazio fisico, all'interno di una sede, destinato allo svolgimento delle lezioni e identificato da un numero.

Di seguito, nella *Figura 6*, è mostrato il diagramma ER completo inerente alla vista “lezioni”.

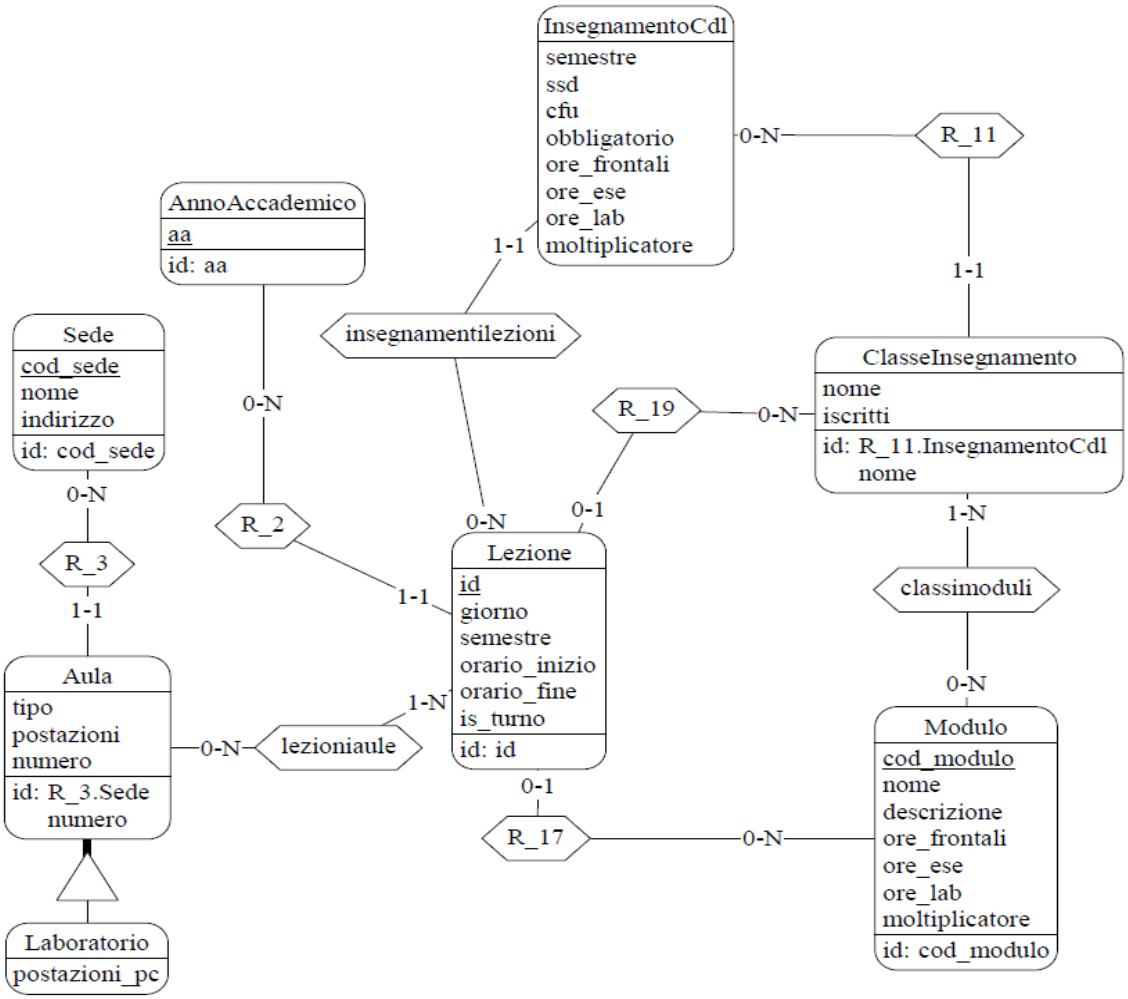

Figura 6: Diagramma ER della vista "lezioni"

La relazione “LezioniAule” svolge una funzione essenziale nella rappresentazione del concetto di lezione, consentendo l’associazione tra ciascuna lezione e le aule in cui essa si svolge. Nel caso di lezioni suddivise in turni, tale relazione può coinvolgere più aule contemporaneamente. Ogni lezione è inoltre legata a uno specifico anno accademico, garantendo così la conservazione dello storico delle lezioni per ciascun anno. Qualora una lezione sia associata a più insegnamenti, a un insegnamento sdoppiato o a un modulo, è necessario che tali elementi appartengano allo stesso anno accademico della lezione stessa.

La relazione “InsegnamentiLezioni” riveste un ruolo altrettanto cruciale, poiché permette di collegare ciascuna lezione agli insegnamenti inclusi al suo interno. Nel caso di lezioni associate a un insegnamento né mutuato né integrato, la relazione conterrà un’unica istanza di insegnamento per quella lezione. Se invece la lezione comprende un insegnamento mutuato o integrato, la relazione includerà tutte le istanze degli insegnamenti coinvolti. Tale struttura è fondamentale per garantire un elevato grado di generalità e adattabilità del database, lasciando al livello applicativo il compito di gestire dinamicamente le associazioni tra lezioni e insegnamenti al momento della loro creazione.

3.3.1.5 Vista di Sistema

All'interno della piattaforma sono previsti diversi ruoli utente, ciascuno con specifici livelli di accesso e funzionalità. L'utente base ha la possibilità di visualizzare esclusivamente le lezioni e i relativi dettagli. L'utente admin, oltre a disporre degli stessi privilegi dell'utente base, può anche modificare e creare nuove lezioni. Infine, l'utente super-admin possiede tutti i privilegi dell'admin e ha, in aggiunta, la facoltà di cancellare le lezioni. Di seguito, nella *Figura 7*, verrà mostrato il diagramma ER completo inerente alla vista “sistema”.

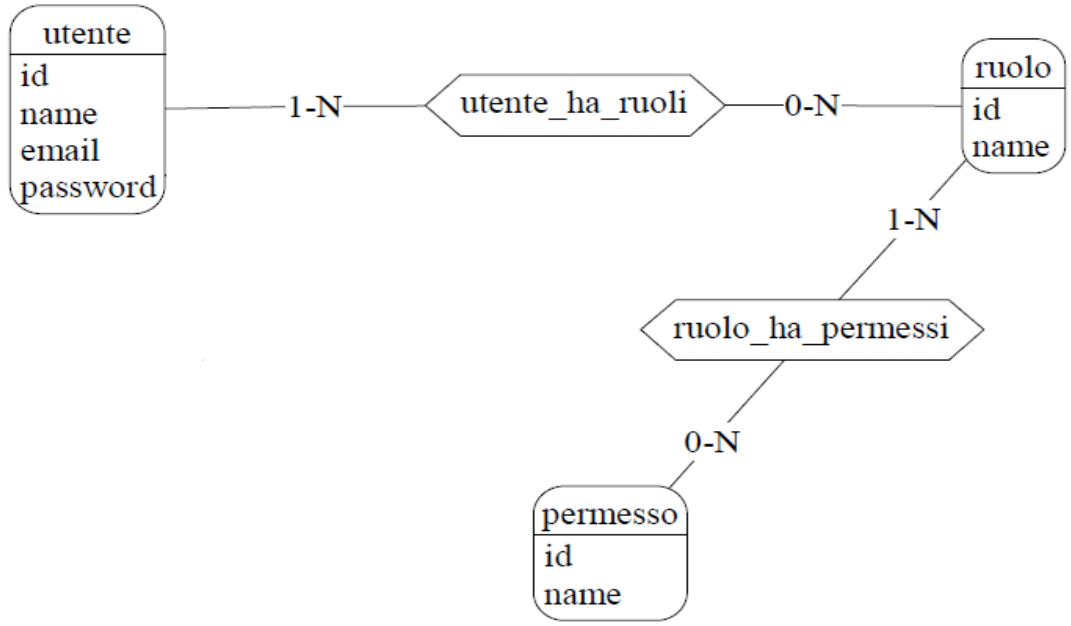

Figura 7: Diagramma ER della vista "sistema"

3.3.2 Progettazione Logica

La progettazione logica di un database consiste nella traduzione del modello concettuale, espresso mediante un diagramma ER, in uno schema conforme al modello relazionale. In questa fase, le entità individuate vengono trasformate in relazioni, i loro attributi diventano campi e gli identificatori assumono il ruolo di chiavi primarie. Le associazioni tra entità vengono rappresentate attraverso chiavi esterne, che collegano tra loro le relazioni in modo da mantenere la coerenza dei dati. I vincoli presenti nel modello concettuale, come le cardinalità o le restrizioni sull'esistenza, vengono anch'essi convertiti in vincoli relazionali. La progettazione logica è indipendente dalla piattaforma tecnologica utilizzata, ma è essenziale per garantire una struttura dei dati precisa, consistente e rispondente ai requisiti informativi rilevati nell'analisi. L'obiettivo è ottenere uno schema chiaro e pronto per la successiva fase di implementazione fisica.

Nella progettazione logica, è fondamentale tradurre correttamente le gerarchie tra entità e risolvere le associazioni molti-a-molti, affinché lo schema relazionale rifletta con precisione la struttura informativa del dominio. Le gerarchie, come quelle derivanti da generalizzazioni o specializzazioni, devono essere rappresentate attraverso strategie che permettano di mantenere l'integrità semantica dei dati, evitando ambiguità o perdita di informazione. Analogamente, le associazioni N a N non possono essere rappresentate direttamente nel modello relazionale e richiedono l'introduzione di una relazione intermedia che colleghi le entità coinvolte, includendo eventualmente gli attributi dell'associazione stessa. Questi passaggi sono essenziali per garantire che lo schema logico sia in grado di supportare correttamente le interrogazioni e le operazioni previste, senza compromettere la qualità e la coerenza dei dati archiviati. Di seguito verrà reificata ciascuna vista del dominio applicativo tramite le tecniche menzionate e verrà fornita una descrizione dettagliata di tutte le tabelle ed associazioni presenti.

Vista Cdl

Il diagramma ER della vista “Cdl” non presentava alcuna gerarchia o associazione N a N, di conseguenza il diagramma ER reificato della suddetta vista rimane invariato rispetto a quello mostrato nella *Figura 3*.

Tabella 5: Tabella descrittiva per l'entità "AnnoAccademico"

Elemento	**Descrizione**
Nome Tabella	AnnoAccademico
Descrizione	anno accademico
Attributi	aa
Chiave Primaria	aa
Vincoli	nessuno
Chiavi Esterne	nessuna

Tabella 6: Tabella descrittiva per l'entità "Cdl"

Elemento	**Descrizione**
Nome Tabella	Cdl
Descrizione	Corso di laurea
Attributi	id_cdl,nome,descrizione
Chiave Primaria	id_cdl
Vincoli	Ciascun Cdl deve possedere almeno un curriculum
Chiavi Esterne	nessuna

Tabella 7: Tabella descrittiva per l'entità "Curriculum"

Elemento	**Descrizione**
Nome Tabella	Curriculum
Descrizione	percorso di studi previsto per un corso di laurea, con insegnamenti, esami e obiettivi formativi

Attributi	cod_curriculum,nome,descrizione,id_cdl
Chiave Primaria	cod_curriculum
Vincoli	Ciascun curriculum deve essere associato ad uno specifico Cdl
Chiavi Esterne	FK_Cdl: (id_cdl)

Tabella 8: Tabella descrittiva per l'entità "AnnoCdl"

Elemento	**Descrizione**
Nome Tabella	AnnoCdl
Descrizione	anno di corso specifico di un determinato curriculum
Attributi	anno,cod_curriculum
Chiave Primaria	anno
Vincoli	nessuno
Chiavi Esterne	FK_Curriculum: (cod_curriculum)

Tabella 9: Tabella descrittiva per l'entità "AnnoCdlInAnnoAccademico"

Elemento	**Descrizione**
Nome Tabella	AnnoCdlInAnnoAccademico
Descrizione	Anno di corso specifico di un determinato curriculum riferito ad un anno accademico
Attributi	cod_curriculum,anno,aa,iscritti
Chiave Primaria	(cod_curriculum, anno)
Vincoli	nessuno
Chiavi Esterne	FK_anno_Cdl: (cod_curriculum, anno) FK_aa: (aa)

Tabella 10: Tabella descrittiva per le relazioni della vista "Cdl"

Relazione	**Descrizione**	**Entità Coinvolte**	**Cardinalità**
R	Associa uno specifico Cdl ad uno o più curriculum	Cdl, Curriculum	Cdl(1,N) → Curriculum(1,1)
R_9	Associa a ciascun curriculum i rispettivi anno di corso	Curriculum, AnnoCdl	Curriculum(1,N) → AnnoCdl(1,1)
R_1	Associa a ciascun anno di corso di uno specifico curriculum ad o o più anni accademici	AnnoCdl, AnnoCdlInAnnoAccademico	AnnoCdl(1,N) → AnnoCdlInAnnoAccademico(1,1)
R_6	Associa ciascun anno di corso al relativo anno anno accademico	AnnoAccademico, AnnoCdlInAnnoAccademico	AnnoAccademico(1,N) → AnnoCdlInAnnoAccademico(1,1)

Vista Insegnamenti

Prima di introdurre il diagramma ER reificato della vista “Insegnamenti”, viene presentato un sottoinsieme di essa, con l’obiettivo di evidenziare la modalità adottata per la gestione delle gerarchie presenti; in quanto queste ultime non sono traducibili in un modello logico relazionale e vanno risolte prima di effettuare la traduzione da modello concettuale a modello logico. In particolare, la relazione tra insegnamenti e corsi di studio si basa su una gerarchia di tipo parziale e sovrapposto. Ogni Insegnamento può essere mutuato da un altro corso, integrato con uno o più insegnamenti affini, oppure presentare entrambe le caratteristiche. In alcuni casi, può non rientrare in alcuna di queste configurazioni. Di seguito è mostrata la reificazione della gerarchia “insegnamenti” nella *Figura 8*.

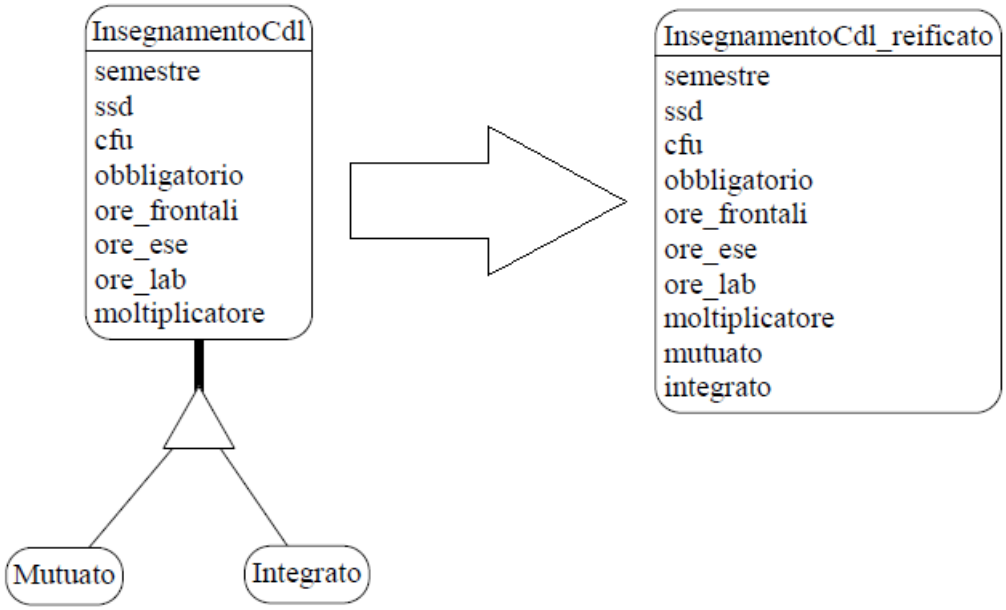

Figura 8: Reificazione gerarchia "insegnamenti"

Per la sua modellazione è stato seguito un approccio di tipo “bottom-up”, che consiste nel portare le caratteristiche specifiche delle entità figlie direttamente all’interno dell’entità padre. In questo modo, anziché introdurre entità distinte per i diversi tipi di insegnamento, si è scelto di gestire le varie configurazioni attraverso due attributi booleani, rispettivamente “mutuato” e “integrato”. Questo consente di rappresentare in modo compatto e flessibile tutte le combinazioni ammissibili, mantenendo al tempo stesso la struttura della vista coerente e facilmente interrogabile.

Di seguito, nella *Figura 9*, è mostrato il diagramma ER reificato inerente alla vista “insegnamenti”.

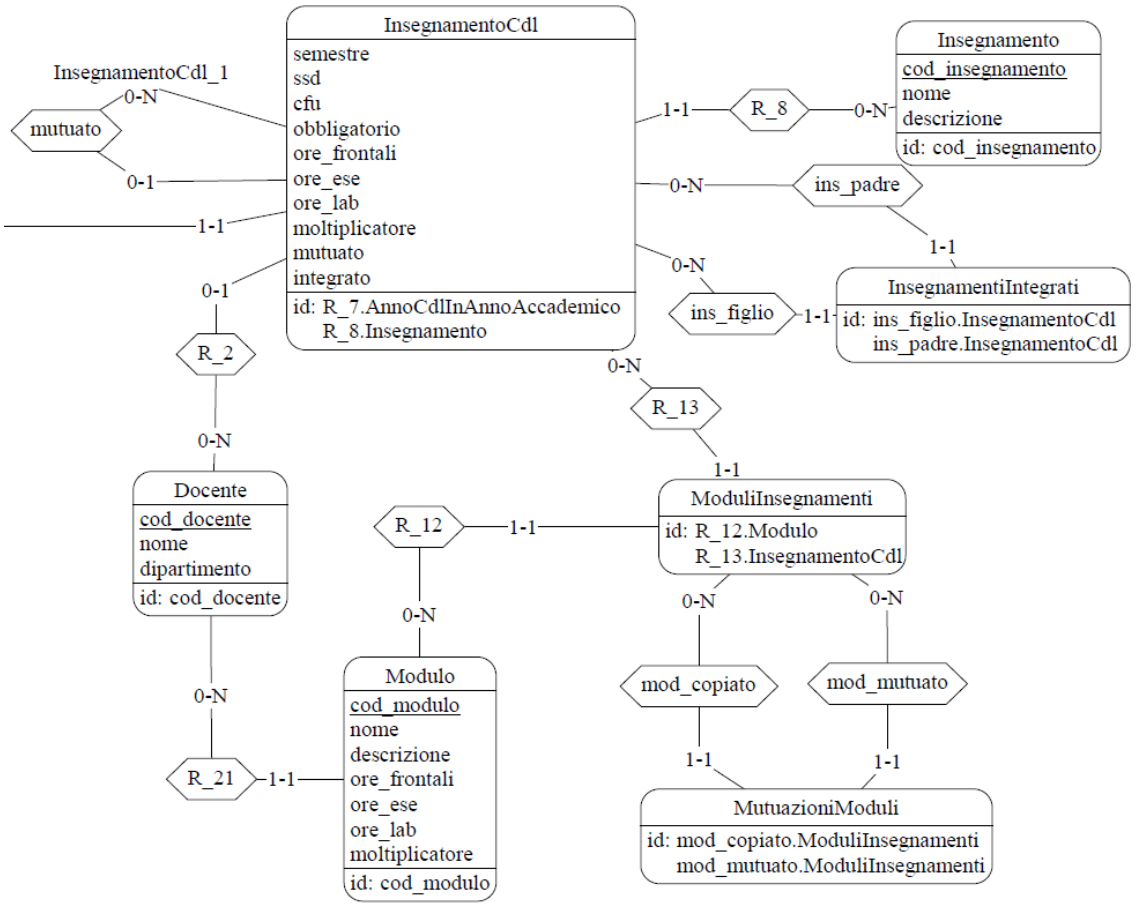

Figura 9: Diagramma ER reificato della vista "insegnamenti"

Tabella 11: Tabella descrittiva per l'entità "Insegnamento"

Elemento	**Descrizione**
Nome tabella	Insegnamento
Descrizione	Rappresenta i dati di un determinato insegnamento che rimangono invariati nel tempo
Attributi	cod_insegnamento, nome, descrizione
Chiave Primaria	cod_insegnamento

Vincoli	nessuno
Chiavi Esterne	nessuna

Tabella 12: Tabella descrittiva per l'entità "Modulo"

Elemento	**Descrizione**
Nome Tabella	Modulo
Descrizione	Rappresenta ciascun modulo di uno o più insegnamenti
Attributi	cod_modulo, nome, descrizione, ore_frontali, ore_ese, ore_lab, moltiplicatore, cod_docente
Chiave Primaria	cod_modulo
Vincoli	Ciascun modulo deve essere associato ad un docente
Chiavi Esterne	FK_Docente: (cod_docente)

Tabella 13: Tabella descrittiva per l'entità "Docente"

Elemento	**Descrizione**
Nome Tabella	Docente
Descrizione	Rappresenta ciascun docente presente all’interno di unibo
Attributi	cod_docente, nome, dipartimento
Chiave Primaria	cod_docente
Vincoli	nessuno
Chiavi Esterne	nessuna

Tabella 14: Tabella descrittiva per l'entità "InsegnamentoCdl"

Elemento	**Descrizione**

Nome Tabella	InsegnamentoCdl
Descrizione	Rappresenta i dati specifici di ciascun insegnamento associato ad un determinato anno di curriculum di un anno accademico specifico
Attributi	cod_curriculum, anno, aa, cod_insegnamento, semestre, ssd,cfu, obbligatorio, ore_frontali, ore_ese, ore_lab, moltiplicatore, mutuato, integrato, cod_docente, ins_mutuato_cod_curriculum, ins_mutuato_anno, ins_mutuato_aa, ins_mutuato_cod_insegnamento
Chiave Primaria	(cod_curriculum, anno, aa, cod_insegnamento)
Vincoli	Ciascun insegnamento, se non ha moduli, deve essere associato ad un docente. Nel caso in cui l'insegnamento sia mutuato deve contenere un riferimento all'insegnamento dal quale è stato mutuato.
Chiavi Esterne	FK_Docente: cod_docente. FK_AnnoCdlinAnnoAccademico: (cod_curriculum, anno, aa) FK_ins_mutuato: (ins_mutuato_cod_curriculum, ins_mutuato_anno, ins_mutuato_aa, ins_mutuato_cod_insegnamento)

Tabella 15: Tabella descrittiva per l'entità "InsegnamentiIntegrati"

Elemento	**Descrizione**
Nome Tabella	InsegnamentiIntegrati

Descrizione	Rappresenta ciascuna coppia di insegnamento figlio ed insegnamento padre nel caso in cui vi sia un insegnamento integrato (ovvero composto da più insegnamenti figlio)
Attributi	ins_padre_cod_curriculum, ins_padre_anno, ins_padre_aa, ins_padre_cod_insegnamento, ins_figlio_cod_curriculum, ins_figlio_anno, ins_figlio_aa, ins_figlio_cod_insegnamento
Chiave Primaria	(ins_padre_cod_curriculum, ins_padre_anno, ins_padre_aa, ins_padre_cod_insegnamento, ins_figlio_cod_curriculum, ins_figlio_anno, ins_figlio_aa, ins_figlio_cod_insegnamento)
Vincoli	I 2 insegnamenti devono appartenere allo stesso anno accademico
Chiavi Esterne	FK_ins_padre: (ins_padre_cod_curriculum, ins_padre_anno, ins_padre_aa, ins_padre_cod_insegnamento) FK_ins_figlio: (ins_figlio_cod_curriculum, ins_figlio_anno, ins_figlio_aa, ins_figlio_cod_insegnamento)

Tabella 16: Tabella descrittiva per l'entità "ModuliInsegnamenti"

Elemento	**Descrizione**
Nome Tabella	ModuliInsegnamenti
Descrizione	Rappresenta ciascuna coppia di Insegnamento e modulo in modo tale da realizzare la suddivisione di un

	determinato insegnamento in 2 o più moduli
Attributi	cod_curriculum, anno, aa, cod_insegnamento, cod_modulo
Chiave Primaria	(cod_curriculum, anno, aa, cod_insegnamento, cod_modulo)
Vincoli	Ciascun insegnamento non può essere associato ad un solo modulo, nel caso in cui l'insegnamento sia sdoppiato deve possedere almeno 2 moduli associati
Chiavi Esterne	FK_InsegnamentoCdl: (cod_curriculum, anno, aa, cod_insegnamento) FK_Modulo: (cod_modulo)

Tabella 17: Tabella descrittiva per l'entità "MutuazioniModuli"

Elemento	**Descrizione**
Nome Tabella	MutuazioniModuli
Descrizione	Rappresenta ciascuna coppia di modulo mutuato e modulo di origine: serve per tener traccia di tutti i moduli mutuati.
Attributi	Mod_cod_curriculum, Mod_anno, Mod_aa, Mod_cod_insegnamento, Mod_cod_modulo, cod_curriculum, anno, aa, cod_insegnamento, cod_modulo
Chiave Primaria	(Mod_cod_curriculum, Mod_anno, Mod_aa, Mod_cod_insegnamento, Mod_cod_modulo, cod_curriculum, anno, aa, cod_insegnamento, cod_modulo)
Vincoli	I moduli mutuati devono appartenere allo stesso anno accademico

Chiavi Esterne	FK_mod_mutuato: (Mod_cod_curriculum, Mod_anno, Mod_aa, Mod_cod_insegnamento, Mod_cod_modulo) FK_mod_copiato: (cod_curriculum, anno, aa, cod_insegnamento, cod_modulo)

Tabella 18: Tabela descrittiva per le relazioni della vista "Insegnamento"

Relazione	**Descrizione**	**Entità Coinvolte**	**Cardinalità**
mutuato	Se un determinato insegnamento è mutuato (attributo mutuato=1) allora questa relazione assocerà all'insegnamento mutuato, l'insegnamento dal quale è stato mutuato	InsegnamentoCdl	InsegnamentoCdl(0,1) → InsegnamentoCdl(0,N)
R_2	Nel caso in cui un insegnamento non sia sdoppiato e non sia suddiviso in classi, deve essere obbligatoriamente associato ad un docente	InsegnamentoCdl, Docente	InsegnamentoCdl(0,1) → Docente(0,N)
R_8	Associa a ciascuna istanza di "InsegnamentoCdl" l'insegnamento relativo	InsegnamentoCdl, Insegnamento	InsegnamentoCdl(1,1) → Insegnamento(0,N)

R_12	Associa a ciascuna istanza di “ModuliInsegnamenti” il relativo modulo	ModuliInsegnamenti, Modulo	ModuliInsegnamenti(1,1) → Modulo(0,N)
R_13	Nel caso in cui un insegnamento sia sdoppiato, questa relazione associa ad un insegnamento i moduli dai quali è composto	ModuliInsegnamenti, InsegnamentoCdl	ModuliInsegnamenti(1,1) → InsegnamentoCdl(0,N)
R_21	Associa ad un modulo il relativo docente	Modulo, Docente	Modulo (1,1) → Docente (0,N)
ins_figlio	Associa ad un insegnamento integrato i rispettivi insegnamenti figlio	InsegnamentoCdl, InsegnamentiIntegrati	InsegnamentiIntegrati(1,1) → InsegnamentoCdl(0,N)
ins_padre	Associa ad uno o più insegnamenti figlio il rispettivo insegnamento padre	InsegnamentoCdl, InsegnamentiIntegrati	InsegnamentoCdl (0,N) → InsegnamentiIntegrati (1,1)
mod_copiato	Associa ad un modulo mutuato il modulo dal quale è stato mutuato	ModuliInsegnamenti, MutuazioniModuli	MutuazioniModuli (1,1) → ModuliInsegnamenti (0,N)
mod_mutuato	Associa ad un modulo il modulo in cui è stato mutuato	ModuliInsegnamenti, MutuazioniModuli	MutuazioniModuli (1,1) → ModuliInsegnamenti(0,N)

Vista Classi

Di seguito, nella *Figura 10*, è mostrato il diagramma ER reificato inerente alla vista “classi”.

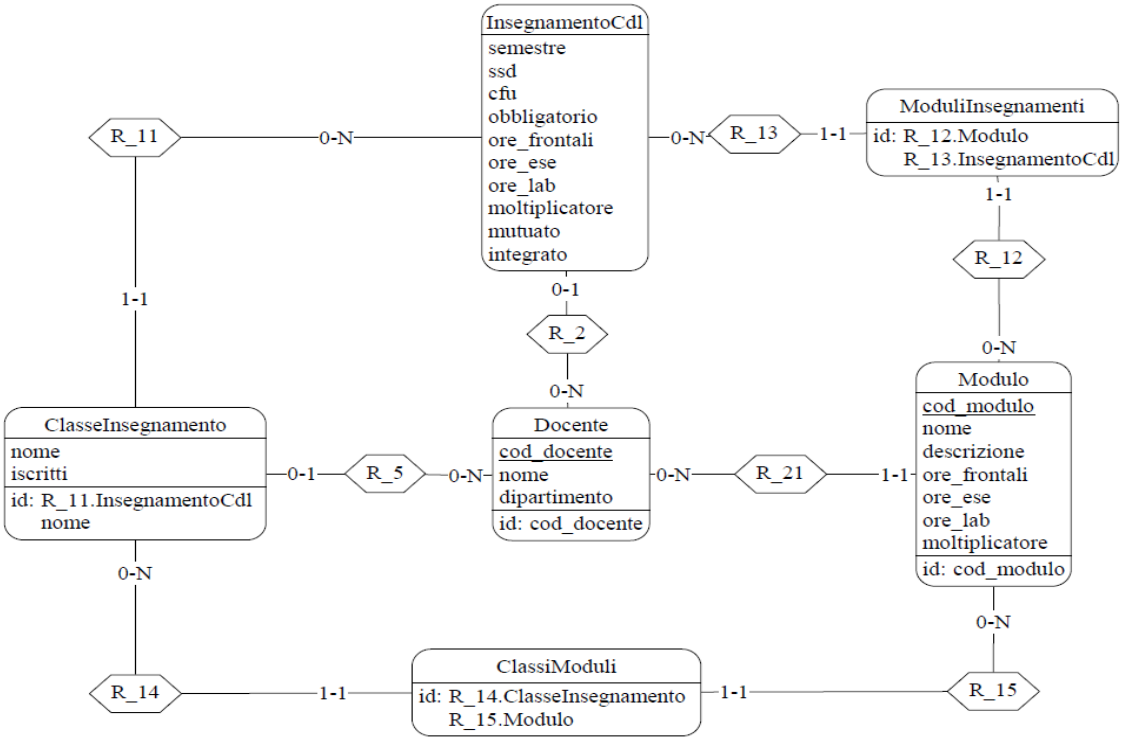

Figura 10: Diagramma ER reificato della vista "classi"

Tabella 19: Tabella descrittiva per l'entità "ClasseInsegnamento"

Elemento	**Descrizione**
Nome Tabella	ClasseInsegnamento
Descrizione	rappresenta ciascuna classe appartentne ad un determinato insegnamento
Attributi	cod_curriculum, anno, aa, cod_insegnamento, classe, iscritti, cod_docente
Chiave Primaria	(cod_curriculum, anno, aa, cod_insegnamento, classe)
Vincoli	• Un insegnamento di un classe

	deve avere un docente associato nel caso in cui l'insegnamento sdoppiato non sia suddiviso in moduli • CHECK_CLASSI_MEMBERS
Chiavi Esterne	FK_Docente: (cod_docente) FK_InsegnamentoCdl: (cod_curriculum, anno, aa, cod_insegnamento)

Tabella 20: Tabella descrittiva per l'entità "ClassiModuli"

Elemento	**Descrizione**
Nome Tabella	ClassiModuli
Descrizione	Rappresenta ciascuna coppia di "ClasseInsegnamento" e "Modulo" in modo tale da associare a ciascun Insegnamento sdoppiato i moduli di ciascuna classe
Attributi	cod_curriculum, anno, aa, cod_insegnamento, cod_modulo
Chiave Primaria	(cod_curriculum, anno, aa, cod_insegnamento, cod_modulo)
Vincoli	nessuno
Chiavi Esterne	FK_InsegnamentoCdl: (cod_curriculum, anno, aa, cod_insegnamento) FK_Modulo: (cod_modulo)

Tabella 21: Tabella descrittiva per le relazioni della vista "classi"

Relazione	**Descrizione**	**Entità Coinvolte**	**Cardinalità**
R_5	Relazione opzionale che associa ad un	ClasseInsegnamento, Docente	ClassInsegnamento(0,1) → Docente(0,N)

	determinato insegnamento sdoppiato il relativo docente		
R_11	Associa ad una determinata classe il relativo insegnamento sdoppiato	ClasseInsegnamento, InsegnamentoCdl	ClasseInsegnamento (1,1) → InsegnamentoCdl(0, N)
R_14	Relazione opzionale che associa ad una determinata istanza di "ClasseInsegnamento" i relativi moduli	ClassiModuli, ClasseInsegnamento	ClasseInsegnamento (0,N) → ClassiModuli(1,1)
R_15	Associa a ciascuna istanza dell'entità "ClassiModuli" il relativo modulo	ClassiModuli, Modulo	ClassiModuli(1,1) → Modulo(0,N)

Vista Lezioni

Di seguito è mostrata la reificazione della gerarchia "aule" nella *Figura 11*.

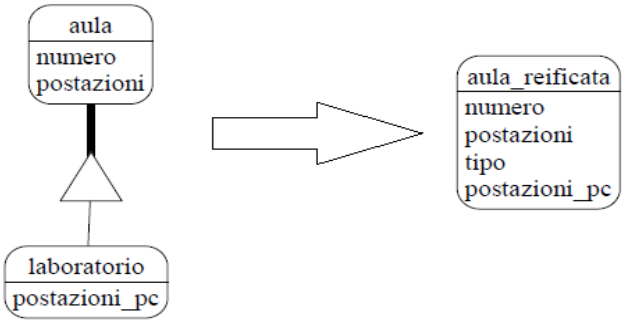

Figura 11: Reificazione gerarchia "aule"

Dal diagramma è possibile osservare come il subset relativo all'entità "Aula" sia stata reificata. Per risolvere questa situazione, che non risulta direttamente traducibile in un modello logico relazionale, è stato adottato un approccio di tipo “bottom-up”. In particolare, gli attributi originariamente collocati a livello delle entità figlie sono stati spostati all'entità padre, alla quale è stato aggiunto un attributo discriminante denominato "tipo" (è stato adottato un attributo selettore per garantire la flessibilità necessaria a rappresentare eventuali nuove tipologie di aule in scenari futuri). Tale attributo, che per default assume il valore "Aula", è definito come un tipo enumerato i cui valori ammissibili sono "Aula" e "laboratorio", permettendo così di distinguere le diverse specializzazioni dell'entità originaria. Di seguito, nella *Figura 12*, è mostrato il diagramma ER reificato inerente alla vista “lezioni”.

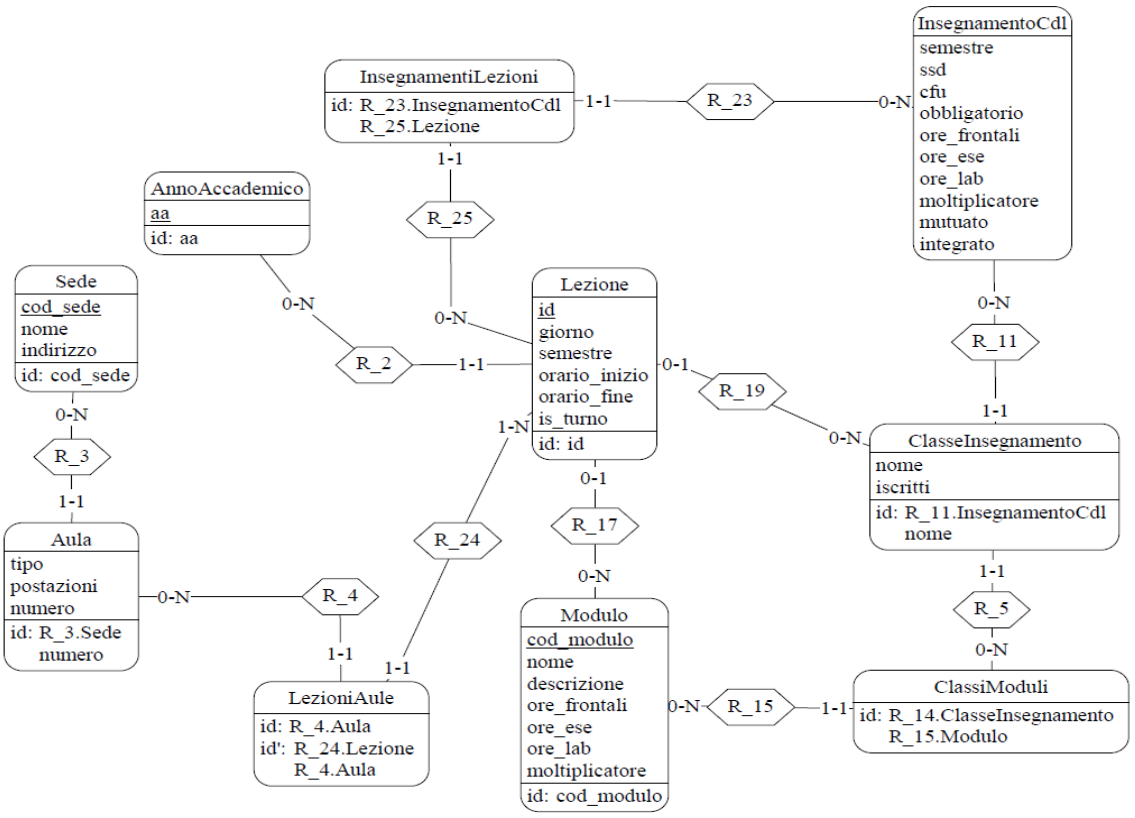

Figura 12: Diagramma ER reificato della vista "Lezioni"

Tabella 22: Tabella descrittiva per l'entità "Sede"

Elemento	**Descrizione**
Nome Tabella	Sede
Descrizione	Rappresenta ciascuna sede dell'università di Bologna
Attributi	cod_sede, nome, indirizzo
Chiave Primaria	cod_sede
Vincoli	nessuno
Chiavi Esterne	nessuna

Tabella 23: Tabella descrittiva dell'entità "Aula"

Elemento	**Descrizione**
Nome Tabella	Aula
Descrizione	Rappresenta ciascuna aula identificata univocamente da un numero all'interno della relativa sede. All'interno dell'entità "Aula" sono racchiuse anche tutte le istanze di tipo "laboratorio"
Attributi	numero, cod_sede, tipo, postazioni, postazioni_pc
Chiave Primaria	(numero, cod_sede)
Vincoli	nessuno
Chiavi Esterne	FK_Sede: (cod_sede)

Tabella 24: Tabella descrittiva dell'entità "Lezione"

Elemento	**Descrizione**
Nome Tabella	Lezione
Descrizione	Rappresenta ciascuna lezione di un determinato semestre all'interno di uno specifico anno accademico

Attributi	id, giorno, semestre, aa, ora_inizio, ora_fine, is_turno, ClasseInsegnamento_cod_curriculum, claseeInsegnamento_aa, ClasseInsegnamento_anno, ClasseInsegnamento_cod_insegnamento, classe, cod_modulo
Chiave Primaria	id
Vincoli	• L'anno accademico deve corrispondere con l'insegnamento, la classe o il modulo associato • UNIQUE_AULA_LEZINE • CHECK_AULA_CAPIENZA • UNIQUE_DOCENTE_LEZIONE • CHECK_INSEGNAMENTI_OVERLAPPING • CHECK_INSEGNAMENTO_HOURS_AMOUNT
Chiavi Esterne	FK_AnnoAccademico: (aa) FK_ClasseInsegnamento: (ClasseInsegnamento_cod_curriculum, claseeInsegnamento_aa, ClasseInsegnamento_anno, ClasseInsegnamento_cod_insegnamento, classe) FK_Modulo: (cod_modulo)

Tabella 25: Tabella descrittiva per l'entità "LezioniAule"

Elemento	**Descrizione**
Nome Tabella	LezioniAule
Descrizione	Rappresenta ciascuna coppia di "Lezione-Aula" in modo tale da associare ad un determinata lezione le

	aule in cui si svolge
Attributi	cod_sede, numero, id
Chiave Primaria	(cod_sede, numero, id)
Vincoli	nessuno
Chiavi Esterne	FK_Aula: (cod_sede, numero) FK_Lezione: (id)

Tabella 26: Tabella descrittiva per l'entità "InsegnamentiLezioni"

Elemento	**Descrizione**
Nome Tabella	InsegnamentiLezioni
Descrizione	Rappresenta l'insieme degli insegnamenti associati ad una specifica lezione
Attributi	cod_curriculum, aa, anno, cod_insegnamento, id
Chiave Primaria	(cod_curriculum, aa, anno, cod_insegnamento, id)
Vincoli	nessuno
Chiavi Esterne	FK_InsegnamentoCdl: (cod_curriculum, aa, anno, cod_insegnamento) FK_Lezione: (id)

Tabella 27: Tabella descrittiva per le relazioni della vista "lezioni"

Relazione	**Descrizione**	**Entità Coinvolte**	**Cardinalità**
R_2	Associa una determinata lezione al relativo anno accademico	Lezione, AnnoAccademico	Lezione (1,1) → AnnoAccdemico(0,N)
R_3	Associa ciascuna aula alla relativa	Aula, Sede	Aula (1,1) → Sede (0,N)

	sede		
R_4	Associa ciascuna ciascuna istanza di “LezioniAule” alla relativa aula	Aula, LezioniAule	LezioniAule (1,1) → Aula (0,N)
R_17	Relazione opzionale che associa una lezione al relativo modulo	Lezione, Modulo	Lezione (0,1) → Modulo (0,N)
R_19	Relazione opzionale che associa la lezione al relativo insegnamento sdoppiato	Lezione, ClasseInsegnamento	Lezione (0,1) → ClasseInsegnamento (0,N)
R_24	Associa ciascuna istanza di “Lezione” ad una o più istanze di “LezioniAule”	LezioniAule, Aula	LezioniAule (1,1) → Aula (0,N)
R_25	Associa ciascuna istanza di “Lezione” ad una o più istanze di “LezioniInsegnamenti”	Lezione, LezioniInsegnamenti	Lezione (0,N) → LezioniInsegnamenti (1,1)
R_23	Associa ciascuna istanza di “LezioniInsegnamenti” ad il relativo insegnamento	LezioniInsegnamenti, InsegnamentoCdl	Lezioninsegnamenti (1,1) → InsegnamentoCdl (0,N)

Vista di Sistema

Di seguito, nella *Figura 13*, è mostrato il diagramma ER reificato inerente alla vista "sistema".

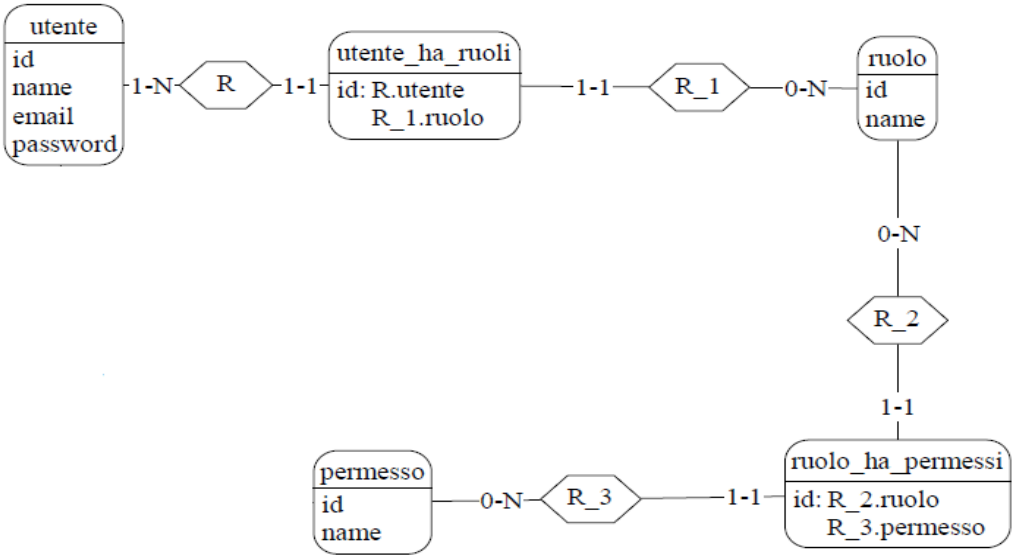

Figura 13: Diagramma ER reificato della vista "sistema"

Tabella 28: Tabella descrittiva dell'entità "utente"

Elemento	**Descrizione**
Nome Tabella	utente
Descrizione	Rappresenta ciascun utente del sistema
Attributi	id, name, email, password
Chiave Primaria	id
Vincoli	Ciascun utente deve avere almeno un ruolo ed un permesso associati
Chiavi Esterne	nessuna

Tabella 29: Tabella descrittiva dell'entità "permesso"

Elemento	**Descrizione**
Nome Tabella	permesso
Descrizione	Rappresenta ciascun permesso

	specifico all’interno del sistema, ciascun ruolo contiene più permessi al suo interno
Attributi	id, name
Chiave Primaria	id
Vincoli	nessuno
Chiavi Esterne	nessuna

Tabella 30: Tabella descrittiva dell'entità "ruolo"

Elemento	**Descrizione**
Nome Tabella	ruolo
Descrizione	Rappresenta ciascun ruolo che può assere assegnato agli utenti del sistema
Attributi	id, name
Chiave Primaria	id
Vincoli	nessuno
Chiavi Esterne	nessuna

Tabella 31: Tabella descrittiva dell'entità "utente_ha_ruoli"

Elemento	**Descrizione**
Nome Tabella	utente_ha_ruoli
Descrizione	Associa a ciascun utente i rispettivi ruoli
Attributi	id_utente, id_ruolo
Chiave Primaria	(id_utente, id_ruolo)
Vincoli	nessuno
Chiavi Esterne	FK_utente: (id_utente) FK_ruolo: (id_ruolo)

Tabella 32: Tabella descrittiva dell'entità "utente_ha_permessi"

Elemento	**Descrizione**
Nome Tabella	utente_ha_permessi
Descrizione	Associa a ciascun utente i relativi permessi
Attributi	id_utente, id_permesso
Chiave Primaria	(id_utente, id_permesso)
Vincoli	nessuno
Chiavi Esterne	FK_utente: (id_utente) FK_permesso: (id_permesso)

Tabella 33: Tabella descrittiva dell'entità "ruolo_ha_permessi"

Elemento	**Descrizione**
Nome Tabella	ruolo_ha_permessi
Descrizione	Associa a ciascun ruolo i relativi permessi che lo compongono
Attributi	id_ruolo, id_permesso
Chiave Primaria	(id_ruolo, id_permesso)
Vincoli	nessuno
Chiavi Esterne	FK_ruolo: (id_ruolo) FK_permesso: (id_permesso)

Modellazione Logica

Di seguito, tutte le entità e le relazioni individuate verranno convertite in tabelle e associazioni secondo i principi del modello logico relazionale. La rappresentazione avverrà in forma testuale, adottando la sintassi e le convenzioni proprie del paradigma relazionale, con l'obiettivo di descrivere in modo chiaro e rigoroso la struttura dei dati e i vincoli associati.

AnnoAccademico(aa, PK: aa)

AnnoCdl(cod_curriculum, anno, PK: cod_ curriculum, anno, FK: cod_curriculum → Curriculum)

AnnoCdlInAnnoAccademico(cod_curriculum, anno, aa, iscritti, PK: cod_curriculum, anno, aa, FK: cod_curriculum, anno → AnnoCdl, FK: aa → AnnoAccademico)

Sede(cod_sede, nome, indirizzo, PK: cod_sede)

Aula(cod_sede, tipo, postazioni, postazioni_pc, numero, PK: cod_sede, numero, FK: cod_sede → Sede)

Cdl(id_cdl, nome, descrizione, tipo, PK: id_cdl)

ClasseInsegnamento(cod_curriculum, anno, aa, cod_insegnamento, classe, iscritti, cod_docente, PK: cod_curriculum, anno, aa, cod_insegnamento, classe, FK: cod_curriculum, anno, aa, cod_insegnamento → InsegnamentoCdl, FK: cod_docente → Docente)

ClassiModuli(cod_curriculum, anno, aa, cod_insegnamento, classe, cod_modulo, PK: cod_curriculum, anno, aa, cod_insegnamento, classe, cod_modulo, FK: cod_curriculum, anno, aa, cod_insegnamento, classe → ClasseInsegnamento)

Curriculum(cod_curriculum, nome, descrizione, id_cdl, PK: cod_curriculum, FK: id_cdl → Cdl)

Docente(cod_docente, nome, dipartimento, PK: cod_docente)

Insegnamento(cod_insegnamento, nome, descrizione, PK: cod_insegnamento)

InsegnamentoCdl(cod_curriculum, anno, aa, cod_insegnamento, semestre, ssd, cfu, obbligatorio, ore_frontali, ore_ese, ore_lab, moltiplicatore, mutuato, integrato, cod_docente, ins_mutuato_cod_curriculum, ins_mutuato_anno, ins_mutuato_aa, ins_mutuato_cod_insegnamento, PK: cod_curriculum, anno, aa, cod_insegnamento, FK: cod_curriculum → Curriculum, FK: cod_docente → Docente, FK: (ins_mutuato_...) → InsegnamentoCdl)

InsegnamentiIntegrati(ins_padre_cod_curriculum, ins_padre_anno, ins_padre_aa, ins_padre_cod_insegnamento, ins_figlio_cod_curriculum,

ins_figlio_anno, ins_figlio_aa, ins_figlio_cod_insegnamento, PK: ins_padre_..., ins_figlio_..., FK: ins_padre → InsegnamentoCdl, FK: ins_figlio → InsegnamentoCdl)

InsegnamentiLezioni(cod_curriculum, anno, aa, cod_insegnamento, id, PK: cod_curriculum, anno, aa, cod_insegnamento, id, FK: cod_curriculum, anno, aa, cod_insegnamento → InsegnamentoCdl, FK: id → Lezione)

Lezione(id, giorno, semestre, aa, orario_inizio, orario_fine, is_turno, cod_modulo, ClasseInsegnamento_cod_curriculum, ClasseInsegnamento_anno, ClasseInsegnamento_aa, ClasseInsegnamento_cod_insegnamento, ClasseInsegnamento_classe, PK: id, FK: aa → AnnoAccademico, FK: cod_modulo → Modulo, FK: ClasseInsegnamento_cod_curriculum, ClasseInsegnamento_anno, ClasseInsegnamento_aa, ClasseInsegnamento_cod_insegnamento, ClasseInsegnamento_classe → ClasseInsegnamento)

LezioniAule(id, cod_sede, numero, PK: id, cod_sede, numero, FK: cod_sede, numero → Aula, FK: id → Lezione)

ModuliInsegnamenti(cod_curriculum, anno, aa, cod_insegnamento, cod_modulo, PK: cod_curriculum, anno, aa, cod_insegnamento, cod_modulo, FK: cod_curriculum, anno, aa, cod_insegnamento → InsegnamentoCdl, FK: cod_modulo → modulo)

utente(id, name, email, password, PK: id)

utente_ha_ruoli(id_utente, id_ruolo, PK: id_utente, id_ruolo, FK: id_utente → utente. FK: id_ruolo → ruolo)

ruolo(id_ruolo, name, PK: id_ruolo)

permesso(id_permesso, name, PK: id_permesso)

ruolo_ha_permessi(id_ruolo, id_permesso, PK: id_ruolo, id_permesso, FK: id_ruolo → ruolo. FK: id_permesso → permesso)

Normalizzazione

La normalizzazione è un processo utilizzato nella progettazione dei database per organizzare i dati in modo ordinato, ridurre le ridondanze e prevenire problemi durante le operazioni di aggiornamento, inserimento o cancellazione. Questo processo si basa su una serie di regole chiamate forme normali, che si applicano progressivamente per migliorare la struttura delle tabelle.

La prima forma normale (1NF) richiede che ogni campo di una tabella contenga un solo valore, quindi non sono ammessi elenchi o gruppi di dati all'interno di una stessa cella. Questo permette di avere dati più semplici da gestire e interrogare.

La seconda forma normale (2NF) si applica solo se la tabella è già in 1NF e prevede che ogni attributo non chiave dipenda dall'intera chiave primaria, non solo da una parte di essa. Serve ad evitare che vi siano dati che dipendono solo parzialmente dalla chiave principale.

La terza forma normale (3NF) aggiunge un ulteriore livello di integrità: oltre a rispettare le regole della 2NF, richiede che gli attributi non chiave dipendano direttamente dalla chiave primaria e non da altri attributi non chiave. Questo elimina le dipendenze transitive, che potrebbero causare ridondanze o incongruenze nei dati.

Sulla base di quanto descritto in precedenza, il modello logico ottenuto a seguito della reificazione del diagramma ER sviluppato nella fase concettuale risulta conforme alla terza forma normale (3NF). Ciò indica che la struttura del database è correttamente modellata, con un'organizzazione dei dati che minimizza le ridondanze e riduce al minimo il rischio di incoerenze. Un modello in 3NF garantisce inoltre una maggiore efficienza nelle operazioni di aggiornamento e una gestione più semplice e affidabile delle informazioni nel tempo.

3.3.3 Progettazione Fisica

La progettazione fisica di un database rappresenta la fase in cui il modello logico, già strutturato secondo le regole del paradigma relazionale, viene tradotto in una forma eseguibile da un sistema di gestione di basi di dati. Nel sistema in questione verrà utilizzato un MySQL, ovvero un RDBMS (Relational Database Management System). Questo passaggio implica la definizione concreta delle strutture di memorizzazione, dei tipi di dato specifici, dei vincoli di integrità e degli indici, tenendo conto delle caratteristiche e delle ottimizzazioni offerte dal

sistema utilizzato. A differenza del modello logico, che è indipendente dalla tecnologia adottata, il modello fisico si adatta al contesto implementativo e introduce dettagli tecnici legati alla gestione reale delle informazioni. Per realizzare il modello fisico direttamente eseguibile da un RDBMS, si utilizza il linguaggio SQL.

3.3.3.1 Modello Fisico

```
--
-- Struttura della tabella `AnnoAccademico`
--
CREATE TABLE `AnnoAccademico` (
  `aa` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `AnnoCdl`
--
CREATE TABLE `AnnoCdl` (
  `cod_curriculum` int(11) NOT NULL,
  `anno` int(11) NOT NULL,
  `border_color` varchar(7) DEFAULT NULL,
  `border_weight` int(11) DEFAULT NULL,
  `background_color` varchar(7) DEFAULT NULL,
  `text_color` varchar(7) DEFAULT NULL,
  `text_override_opzionale` varchar(7) DEFAULT NULL,
  `background_override_opzionale` varchar(7) DEFAULT NULL,
  `text_override_mutuato` varchar(7) DEFAULT NULL,
  `background_override_mutuato` varchar(7) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `AnnoCdlInAnnoAccademico`
--
CREATE TABLE `AnnoCdlInAnnoAccademico` (
  `cod_curriculum` int(11) NOT NULL,
  `anno` int(11) NOT NULL,
  `aa` int(11) NOT NULL,
  `iscritti` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `Aula`
--
CREATE TABLE `Aula` (
```

```
  `cod_sede` varchar(50) NOT NULL,
  `tipo` varchar(100) NOT NULL,
  `postazioni` int(11) NOT NULL,
  `postazioni_pc` int(11) NOT NULL,
  `numero` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `Cdl`
--
CREATE TABLE `Cdl` (
  `nome` varchar(255) NOT NULL,
  `descrizione` varchar(255) NOT NULL,
  `tipo` varchar(50) NOT NULL,
  `id_Cdl` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `ClasseInsegnamento`
--
CREATE TABLE `ClasseInsegnamento` (
  `cod_curriculum` int(11) NOT NULL,
  `anno` int(11) NOT NULL,
  `aa` int(11) NOT NULL,
  `cod_insegnamento` varchar(50) NOT NULL,
  `classe` varchar(20) NOT NULL,
  `iscritti` int(11) NOT NULL,
  `cod_docente` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `ClassiModuli`
--
CREATE TABLE `ClassiModuli` (
  `cod_curriculum` int(11) NOT NULL,
  `anno` int(11) NOT NULL,
  `aa` int(11) NOT NULL,
  `cod_insegnamento` varchar(50) NOT NULL,
  `classe` varchar(20) NOT NULL,
  `cod_modulo` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `Curriculum`
--
CREATE TABLE `Curriculum` (
```

```
  `cod_curriculum` int(11) NOT NULL,
  `nome` varchar(100) NOT NULL,
  `descrizione` varchar(255) NOT NULL,
  `id_Cdl` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `Docente`
--
CREATE TABLE `Docente` (
  `cod_docente` int(11) NOT NULL,
  `nome` varchar(100) NOT NULL,
  `dipartimento` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `InsegnamentiIntegrati`
--
CREATE TABLE `InsegnamentiIntegrati` (
  `ins_padre_cod_curriculum` int(11) NOT NULL,
  `ins_padre_anno` int(11) NOT NULL,
  `ins_padre_aa` int(11) NOT NULL,
  `ins_padre_cod_insegnamento` varchar(50) NOT NULL,
  `ins_figlio_cod_curriculum` int(11) NOT NULL,
  `ins_figlio_anno` int(11) NOT NULL,
  `ins_figlio_aa` int(11) NOT NULL,
  `ins_figlio_cod_insegnamento` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `InsegnamentiLezioni`
--
CREATE TABLE `InsegnamentiLezioni` (
  `cod_curriculum` int(11) NOT NULL,
  `anno` int(11) NOT NULL,
  `aa` int(11) NOT NULL,
  `cod_insegnamento` varchar(50) NOT NULL,
  `id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `Insegnamento`
--
CREATE TABLE `Insegnamento` (
  `cod_insegnamento` varchar(50) NOT NULL,
  `nome` varchar(100) NOT NULL,
```

```
  `descrizione` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `InsegnamentoCdl`
--
CREATE TABLE `InsegnamentoCdl` (
  `cod_curriculum` int(11) NOT NULL,
  `anno` int(11) NOT NULL,
  `aa` int(11) NOT NULL,
  `cod_insegnamento` varchar(50) NOT NULL,
  `semestre` decimal(1,0) NOT NULL,
  `ssd` varchar(20) NOT NULL,
  `cfu` int(11) NOT NULL,
  `obbligatorio` char(1) NOT NULL,
  `ore_frontali` int(11) DEFAULT NULL,
  `ore_ese` int(11) DEFAULT NULL,
  `ore_lab` int(11) DEFAULT NULL,
  `moltiplicatore` int(11) DEFAULT NULL,
  `mutuato` char(1) NOT NULL,
  `integrato` char(1) NOT NULL,
  `cod_docente` int(11) DEFAULT NULL,
  `ins_mutuato_cod_curriculum` int(11) DEFAULT NULL,
  `ins_mutuato_anno` int(11) DEFAULT NULL,
  `ins_mutuato_aa` int(11) DEFAULT NULL,
  `ins_mutuato_cod_insegnamento` varchar(50) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `LezioniAule`
--
CREATE TABLE `LezioniAule` (
  `id` int(11) NOT NULL,
  `cod_sede` varchar(50) NOT NULL,
  `numero` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `utente_ha_permessi`
--
CREATE TABLE `utente_ha_permessi` (
  `permission_id` bigint(20) UNSIGNED NOT NULL,
  `model_type` varchar(255) NOT NULL,
  `model_id` bigint(20) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```sql
-- --------------------------------------------------------
--
-- Struttura della tabella `utente_ha_ruoli`
--
CREATE TABLE `utente_ha_ruoli` (
  `role_id` bigint(20) UNSIGNED NOT NULL,
  `model_type` varchar(255) NOT NULL,
  `model_id` bigint(20) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `ModuliInsegnamenti`
--
CREATE TABLE `ModuliInsegnamenti` (
  `cod_curriculum` int(11) NOT NULL,
  `anno` int(11) NOT NULL,
  `aa` int(11) NOT NULL,
  `cod_insegnamento` varchar(50) NOT NULL,
  `cod_modulo` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `Modulo`
--
CREATE TABLE `Modulo` (
  `cod_modulo` int(11) NOT NULL,
  `nome` varchar(50) NOT NULL,
  `descrizione` varchar(255) NOT NULL,
  `ore_frontali` int(11) NOT NULL,
  `ore_ese` int(11) NOT NULL,
  `ore_lab` int(11) NOT NULL,
  `moltiplicatore` int(11) NOT NULL,
  `cod_docente` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `MutuazioniModuli`
--
CREATE TABLE `MutuazioniModuli` (
  `Mod_cod_modulo` int(11) NOT NULL,
  `Mod_cod_curriculum` int(11) NOT NULL,
  `Mod_anno` int(11) NOT NULL,
  `Mod_aa` int(11) NOT NULL,
  `Mod_cod_insegnamento` varchar(50) NOT NULL,
  `cod_modulo` int(11) NOT NULL,
  `cod_curriculum` int(11) NOT NULL,
```

```
  `anno` int(11) NOT NULL,
  `aa` int(11) NOT NULL,
  `cod_insegnamento` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `Lezione`
--
CREATE TABLE `Lezione` (
  `id` int(11) NOT NULL,
  `giorno` int(11) NOT NULL,
  `semestre` int(11) NOT NULL,
  `aa` int(11) NOT NULL,
  `orario_inizio` time NOT NULL,
  `orario_fine` time NOT NULL,
  `is_turno` decimal(1,0) NOT NULL,
  `cod_modulo` int(11) DEFAULT NULL,
  `classeinsegnamento_cod_curriculum` int(11) DEFAULT NULL,
  `classeinsegnamento_anno` int(11) DEFAULT NULL,
  `classeinsegnamento_aa` int(11) DEFAULT NULL,
  `classeinsegnamento_cod_insegnamento` varchar(50) DEFAULT NULL,
  `classeinsegnamento_classe` varchar(20) DEFAULT NULL,
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `permesso`
--
CREATE TABLE `permesso` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `name` varchar(255) NOT NULL,
  `guard_name` varchar(255) NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `ruolo`
--
CREATE TABLE `ruolo` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `name` varchar(255) NOT NULL,
  `guard_name` varchar(255) NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```sql
-- --------------------------------------------------------
--
-- Struttura della tabella `ruolo_ha_permesso`
--
CREATE TABLE `ruolo_ha_permesso` (
  `permission_id` bigint(20) UNSIGNED NOT NULL,
  `role_id` bigint(20) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `sede`
--
CREATE TABLE `sede` (
  `cod_sede` varchar(50) NOT NULL,
  `nome` varchar(100) NOT NULL,
  `indirizzo` varchar(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- --------------------------------------------------------
--
-- Struttura della tabella `utente`
--
CREATE TABLE `utente` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `name` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `email_verified_at` timestamp NULL DEFAULT NULL,
  `password` varchar(255) NOT NULL,
  `remember_token` varchar(100) DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
--
-- Indici per le tabelle scaricate
--
--
-- Indici per le tabelle `AnnoAccademico`
--
ALTER TABLE `AnnoAccademico`
  ADD PRIMARY KEY (`aa`),
  ADD UNIQUE KEY `ID_AnnoAccademico_IND` (`aa`);
--
-- Indici per le tabelle `AnnoCdl`
--
ALTER TABLE `AnnoCdl`
  ADD PRIMARY KEY (`cod_curriculum`,`anno`),
  ADD UNIQUE KEY `ID_AnnoCdl_IND` (`cod_curriculum`,`anno`);
--
-- Indici per le tabelle `AnnoCdlInAnnoAccademico`
```

```
--
ALTER TABLE `AnnoCdlInAnnoAccademico`
  ADD PRIMARY KEY (`cod_curriculum`,`anno`,`aa`),
  ADD UNIQUE KEY `ID_AnnoCdlInAnnoAccademico_IND` (`cod_curriculum`,`anno`,`aa`),
  ADD KEY `REF_AnnoC_AnnoA_IND` (`aa`);
--
-- Indici per le tabelle `Aula`
--
ALTER TABLE `Aula`
  ADD PRIMARY KEY (`cod_sede`,`numero`),
  ADD UNIQUE KEY `ID_Aula_IND` (`cod_sede`,`numero`);
--
-- Indici per le tabelle `Cdl`
--
ALTER TABLE `Cdl`
  ADD PRIMARY KEY (`id_Cdl`),
  ADD UNIQUE KEY `ID_Cdl_IND` (`id_Cdl`);
--
-- Indici per le tabelle `ClasseInsegnamento`
--
ALTER TABLE `ClasseInsegnamento`
  ADD PRIMARY KEY (`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`,`classe`),
  ADD UNIQUE KEY `ID_ClasseInsegnamento_IND`
(`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`,`classe`),
  ADD KEY `REF_Class_Docen_IND` (`cod_docente`);
--
-- Indici per le tabelle `ClassiModuli`
--
ALTER TABLE `ClassiModuli`
  ADD PRIMARY KEY (`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`,`classe`,`cod_modulo`),
  ADD UNIQUE KEY `ID_ClassiModuli_IND`
(`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`,`classe`,`cod_modulo`),
  ADD KEY `REF_Class_Modul_IND` (`cod_modulo`);
--
-- Indici per le tabelle `Curriculum`
--
ALTER TABLE `Curriculum`
  ADD PRIMARY KEY (`cod_curriculum`),
  ADD UNIQUE KEY `ID_Curriculum_IND` (`cod_curriculum`),
  ADD KEY `REF_Curri_Cdl_IND` (`id_Cdl`);
--
-- Indici per le tabelle `Docente`
--
ALTER TABLE `Docente`
  ADD PRIMARY KEY (`cod_docente`),
  ADD UNIQUE KEY `ID_Docente_IND` (`cod_docente`);
--
-- Indici per le tabelle `InsegnamentiIntegrati`
```

```
--
ALTER TABLE `InsegnamentiIntegrati`
 ADD PRIMARY KEY
(
`ins_padre_cod_curriculum`,`ins_padre_anno`,`ins_padre_aa`,`ins_padre_cod_insegnamento`,`i
ns_figlio_cod_curriculum`,`ins_figlio_anno`,`ins_figlio_aa`,`ins_figlio_cod_insegnamento`)
,
 ADD UNIQUE KEY `ID_InsegnamentiIntegrati_IND`
(
`ins_padre_cod_curriculum`,`ins_padre_anno`,`ins_padre_aa`,`ins_padre_cod_insegnamento`,`i
ns_figlio_cod_curriculum`,`ins_figlio_anno`,`ins_figlio_aa`,`ins_figlio_cod_insegnamento`)
,
 ADD KEY `REF_Inseg_Inseg_4_IND`
(
`ins_figlio_cod_curriculum`,`ins_figlio_anno`,`ins_figlio_aa`,`ins_figlio_cod_insegnamento
`);
--
-- Indici per le tabelle `InsegnamentiLezioni`
--
ALTER TABLE `InsegnamentiLezioni`
 ADD PRIMARY KEY (`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`,`id`),
 ADD UNIQUE KEY `ID_InsegnamentiLezioni_IND`
(`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`,`id`),
 ADD KEY `REF_Inseg_Orari_IND` (`id`);

--
-- Indici per le tabelle `Insegnamento`
--
ALTER TABLE `Insegnamento`
 ADD PRIMARY KEY (`cod_insegnamento`),
 ADD UNIQUE KEY `ID_Insegnamento_IND` (`cod_insegnamento`);
--
-- Indici per le tabelle `InsegnamentoCdl`
--
ALTER TABLE `InsegnamentoCdl`
 ADD PRIMARY KEY (`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`),
 ADD UNIQUE KEY `ID_InsegnamentoCdl_IND`
(`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`),
 ADD KEY `REF_Inseg_Inseg_1_IND` (`cod_insegnamento`),
 ADD KEY `REF_Inseg_Docen_IND` (`cod_docente`),
 ADD KEY `REF_Inseg_Inseg_FK`
(
`ins_mutuato_cod_curriculum`,`ins_mutuato_anno`,`ins_mutuato_aa`,`ins_mutuato_cod_insegnam
ento`);
--
-- Indici per le tabelle `LezioniAule`
--
ALTER TABLE `LezioniAule`
 ADD PRIMARY KEY (`id`,`cod_sede`,`numero`),
```

```
  ADD UNIQUE KEY `SID_LezioniAule_ID` (`id`,`cod_sede`,`numero`),
  ADD KEY `ID_Lezio_Aula_FK` (`cod_sede`,`numero`);
--
-- Indici per le tabelle `utente_ha_permessi`
--
ALTER TABLE `utente_ha_permessi`
  ADD PRIMARY KEY (`permission_id`,`model_id`,`model_type`),
  ADD KEY `model_has_permissions_model_id_model_type_index` (`model_id`,`model_type`);
--
-- Indici per le tabelle `utente_ha_ruoli`
--
ALTER TABLE `utente_ha_ruoli`
  ADD PRIMARY KEY (`role_id`,`model_id`,`model_type`),
  ADD KEY `model_has_roles_model_id_model_type_index` (`model_id`,`model_type`);
--
-- Indici per le tabelle `ModuliInsegnamenti`
--
ALTER TABLE `ModuliInsegnamenti`
  ADD PRIMARY KEY (`cod_modulo`,`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`),
  ADD UNIQUE KEY `ID_ModuliInsegnamenti_IND`
(`cod_modulo`,`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`),
  ADD KEY `REF_Modul_Inseg_IND` (`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`);
--
-- Indici per le tabelle `Modulo`
--
ALTER TABLE `Modulo`
  ADD PRIMARY KEY (`cod_modulo`),
  ADD UNIQUE KEY `ID_Modulo_IND` (`cod_modulo`),
  ADD KEY `REF_Modul_Docen_IND` (`cod_docente`);
--
-- Indici per le tabelle `MutuazioniModuli`
--
ALTER TABLE `MutuazioniModuli`
  ADD PRIMARY KEY
(
`Mod_cod_modulo`,`Mod_cod_curriculum`,`Mod_anno`,`Mod_aa`,`Mod_cod_insegnamento`,`cod_modu
lo`,`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`),
  ADD UNIQUE KEY `ID_MutuazioniModuli_IND`
(
`Mod_cod_modulo`,`Mod_cod_curriculum`,`Mod_anno`,`Mod_aa`,`Mod_cod_insegnamento`,`cod_modu
lo`,`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`),
  ADD KEY `REF_Mutua_Modul_1_IND`
(`cod_modulo`,`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`);
--
-- Indici per le tabelle `Lezione`
--
ALTER TABLE `Lezione`
  ADD PRIMARY KEY (`id`),
```

```
  ADD UNIQUE KEY `ID_OrarioLezioni_IND` (`id`),
  ADD KEY `REF_Orari_Modul_IND` (`cod_modulo`),
  ADD KEY `REF_Orari_Class_1_IND`
(
`ClasseInsegnamento_cod_curriculum`,`ClasseInsegnamento_anno`,`ClasseInsegnamento_aa`,`Cla
sseInsegnamento_cod_insegnamento`,`ClasseInsegnamento_classe`),
  ADD KEY `FK_aa` (`aa`);
--
-- Indici per le tabelle `permesso`
--
ALTER TABLE `permesso`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `permissions_name_guard_name_unique` (`name`,`guard_name`);
--
-- Indici per le tabelle `ruolo`
--
ALTER TABLE `ruolo`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `roles_name_guard_name_unique` (`name`,`guard_name`);
--
-- Indici per le tabelle `ruolo_ha_permesso`
--
ALTER TABLE `ruolo_ha_permesso`
  ADD PRIMARY KEY (`permission_id`,`role_id`),
  ADD KEY `role_has_permissions_role_id_foreign` (`role_id`);

--
-- Indici per le tabelle `sede`
--
ALTER TABLE `sede`
  ADD PRIMARY KEY (`cod_sede`),
  ADD UNIQUE KEY `ID_Sede_IND` (`cod_sede`);
--
-- Indici per le tabelle `utente`
--
ALTER TABLE `utente`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `users_email_unique` (`email`);
--
-- AUTO_INCREMENT per la tabella `Docente`
--
ALTER TABLE `Docente`
  MODIFY `cod_docente` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT per la tabella `Modulo`
--
ALTER TABLE `Modulo`
  MODIFY `cod_modulo` int(11) NOT NULL AUTO_INCREMENT;
```

```
--
-- AUTO_INCREMENT per la tabella `Lezione`
--
ALTER TABLE `Lezione`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT per la tabella `permesso`
--
ALTER TABLE `permesso`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT per la tabella `ruolo`
--
ALTER TABLE `ruolo`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT per la tabella `utente`
--
ALTER TABLE `utente`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;
--
-- Limiti per la tabella `AnnoCdl`
--
ALTER TABLE `AnnoCdl`
  ADD CONSTRAINT `REF_AnnoC_Curri` FOREIGN KEY (`cod_curriculum`) REFERENCES `Curriculum`
(`cod_curriculum`);
--
-- Limiti per la tabella `AnnoCdlInAnnoAccademico`
--
ALTER TABLE `AnnoCdlInAnnoAccademico`
  ADD CONSTRAINT `REF_AnnoC_AnnoA_FK` FOREIGN KEY (`aa`) REFERENCES `AnnoAccademico`
(`aa`),
  ADD CONSTRAINT `REF_AnnoC_AnnoC` FOREIGN KEY (`cod_curriculum`,`anno`) REFERENCES
`AnnoCdl` (`cod_curriculum`, `anno`);
--
-- Limiti per la tabella `Aula`
--
ALTER TABLE `Aula`
  ADD CONSTRAINT `REF_Aula_Sede` FOREIGN KEY (`cod_sede`) REFERENCES `sede` (`cod_sede`);
--
-- Limiti per la tabella `ClasseInsegnamento`
--
ALTER TABLE `ClasseInsegnamento`
  ADD CONSTRAINT `REF_Class_Docen_FK` FOREIGN KEY (`cod_docente`) REFERENCES `Docente`
(`cod_docente`),
  ADD CONSTRAINT `REF_Class_Inseg` FOREIGN KEY
(`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`) REFERENCES `InsegnamentoCdl`
(`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`);
--
-- Limiti per la tabella `ClassiModuli`
```

```
--
ALTER TABLE `ClassiModuli`
 ADD CONSTRAINT `REF_Class_Class` FOREIGN KEY
(`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`,`classe`) REFERENCES `ClasseInsegnamento`
(`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`, `classe`),
 ADD CONSTRAINT `REF_Class_Modul_FK` FOREIGN KEY (`cod_modulo`) REFERENCES `Modulo`
(`cod_modulo`);
--
-- Limiti per la tabella `Curriculum`
--
ALTER TABLE `Curriculum`
 ADD CONSTRAINT `REF_Curri_Cdl_FK` FOREIGN KEY (`id_Cdl`) REFERENCES `Cdl` (`id_Cdl`);
--
-- Limiti per la tabella `InsegnamentiIntegrati`
--
ALTER TABLE `InsegnamentiIntegrati`
 ADD CONSTRAINT `REF_Inseg_Inseg_3` FOREIGN KEY
(`ins_padre_cod_curriculum`,`ins_padre_anno`,`ins_padre_aa`,`ins_padre_cod_insegnamento`)
REFERENCES `InsegnamentoCdl` (`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`),
 ADD CONSTRAINT `REF_Inseg_Inseg_4_FK` FOREIGN KEY
(
`ins_figlio_cod_curriculum`,`ins_figlio_anno`,`ins_figlio_aa`,`ins_figlio_cod_insegnamento
`) REFERENCES `InsegnamentoCdl` (`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`);
--
-- Limiti per la tabella `InsegnamentiLezioni`
--
ALTER TABLE `InsegnamentiLezioni`
 ADD CONSTRAINT `REF_Inseg_Inseg_2` FOREIGN KEY
(`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`) REFERENCES `InsegnamentoCdl`
(`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`),
 ADD CONSTRAINT `REF_Inseg_Orari_FK` FOREIGN KEY (`id`) REFERENCES `Lezione` (`id`);
--
-- Limiti per la tabella `InsegnamentoCdl`
--
ALTER TABLE `InsegnamentoCdl`
 ADD CONSTRAINT `REF_Inseg_AnnoC` FOREIGN KEY (`cod_curriculum`,`anno`,`aa`) REFERENCES
`AnnoCdlInAnnoAccademico` (`cod_curriculum`, `anno`, `aa`),
 ADD CONSTRAINT `REF_Inseg_Docen_FK` FOREIGN KEY (`cod_docente`) REFERENCES `Docente`
(`cod_docente`),
 ADD CONSTRAINT `REF_Inseg_Inseg_1_FK` FOREIGN KEY (`cod_insegnamento`) REFERENCES
`Insegnamento` (`cod_insegnamento`),
 ADD CONSTRAINT `REF_Inseg_Inseg_FK` FOREIGN KEY
(
`ins_mutuato_cod_curriculum`,`ins_mutuato_anno`,`ins_mutuato_aa`,`ins_mutuato_cod_insegnam
ento`) REFERENCES `InsegnamentoCdl` (`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`);
--
-- Limiti per la tabella `LezioniAule`
--
ALTER TABLE `LezioniAule`
```

```
  ADD CONSTRAINT `ID_Lezio_Aula_FK` FOREIGN KEY (`cod_sede`,`numero`) REFERENCES `Aula`
(`cod_sede`, `numero`),
  ADD CONSTRAINT `REF_Lezio_Orari` FOREIGN KEY (`id`) REFERENCES `Lezione` (`id`);
--
-- Limiti per la tabella `utente_ha_permessi`
--
ALTER TABLE `utente_ha_permessi`
  ADD CONSTRAINT `model_has_permissions_permission_id_foreign` FOREIGN KEY
(`permission_id`) REFERENCES `permesso` (`id`) ON DELETE CASCADE;
--
-- Limiti per la tabella `utente_ha_ruoli`
--
ALTER TABLE `utente_ha_ruoli`
  ADD CONSTRAINT `model_has_roles_role_id_foreign` FOREIGN KEY (`role_id`) REFERENCES
`ruolo` (`id`) ON DELETE CASCADE;
--
-- Limiti per la tabella `ModuliInsegnamenti`
--
ALTER TABLE `ModuliInsegnamenti`
  ADD CONSTRAINT `REF_Modul_Inseg_FK` FOREIGN KEY
(`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`) REFERENCES `InsegnamentoCdl`
(`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`),
  ADD CONSTRAINT `REF_Modul_Modul` FOREIGN KEY (`cod_modulo`) REFERENCES `Modulo`
(`cod_modulo`);
--
-- Limiti per la tabella `Modulo`
--
ALTER TABLE `Modulo`
  ADD CONSTRAINT `REF_Modul_Docen_FK` FOREIGN KEY (`cod_docente`) REFERENCES `Docente`
(`cod_docente`);
--
-- Limiti per la tabella `MutuazioniModuli`
--
ALTER TABLE `MutuazioniModuli`
  ADD CONSTRAINT `REF_Mutua_Modul` FOREIGN KEY
(`Mod_cod_modulo`,`Mod_cod_curriculum`,`Mod_anno`,`Mod_aa`,`Mod_cod_insegnamento`)
REFERENCES `ModuliInsegnamenti` (`cod_modulo`, `cod_curriculum`, `anno`, `aa`,
`cod_insegnamento`),
  ADD CONSTRAINT `REF_Mutua_Modul_1_FK` FOREIGN KEY
(`cod_modulo`,`cod_curriculum`,`anno`,`aa`,`cod_insegnamento`) REFERENCES
`ModuliInsegnamenti` (`cod_modulo`, `cod_curriculum`, `anno`, `aa`, `cod_insegnamento`);
--
-- Limiti per la tabella `Lezione`
--
ALTER TABLE `Lezione`
  ADD CONSTRAINT `FK_aa` FOREIGN KEY (`aa`) REFERENCES `AnnoAccademico` (`aa`),
  ADD CONSTRAINT `REF_Orari_Class_1_FK` FOREIGN KEY
(
`ClasseInsegnamento_cod_curriculum`,`ClasseInsegnamento_anno`,`ClasseInsegnamento_aa`,`Cla
```

```
sseInsegnamento_cod_insegnamento`,`ClasseInsegnamento_classe`) REFERENCES
`ClasseInsegnamento` (`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`, `classe`),
 ADD CONSTRAINT `REF_Orari_Class_FK` FOREIGN KEY
(
`classemodulo_cod_curriculum`,`classemodulo_anno`,`classemodulo_aa`,`classemodulo_cod_inse
gnamento`,`classemodulo_classe`,`classemodulo_cod_modulo`) REFERENCES `ClassiModuli`
(`cod_curriculum`, `anno`, `aa`, `cod_insegnamento`, `classe`, `cod_modulo`),
 ADD CONSTRAINT `REF_Orari_Modul_FK` FOREIGN KEY (`cod_modulo`) REFERENCES `Modulo`
(`cod_modulo`);
--
-- Limiti per la tabella `ruolo_ha_permesso`
--
ALTER TABLE `ruolo_ha_permesso`
 ADD CONSTRAINT `role_has_permissions_permission_id_foreign` FOREIGN KEY (`permission_id`)
REFERENCES `permesso` (`id`) ON DELETE CASCADE,
 ADD CONSTRAINT `role_has_permissions_role_id_foreign` FOREIGN KEY (`role_id`) REFERENCES
`ruolo` (`id`) ON DELETE CASCADE;
```

3.3.4 Vincoli

In questa fase, tutti i criteri individuati nel paragrafo 2.1 devono essere tradotti operativamente all'interno del sistema. Come base, sono stati utilizzati i vincoli di integrità impliciti offerti dal DBMS per la gestione delle chiavi primarie e delle chiavi esterne, che generano automaticamente indici su tali attributi al fine di ottimizzare le operazioni di ricerca e accesso ai dati.

Per quanto riguarda la traduzione dei criteri appartenenti alla logica di business in vincoli applicativi, si è valutato il modo più efficace per integrarli nel sistema. In linea teorica, tali vincoli potrebbero essere implementati direttamente a livello di DBMS, tramite l'utilizzo di trigger e stored procedures, oppure gestiti interamente lato applicazione. Si è scelto di non prendere in considerazione una soluzione ibrida, in cui parte dei vincoli risieda nel database e parte nell'applicazione, poiché tale approccio comprometterebbe la coerenza progettuale e renderebbe più complessa la gestione complessiva del sistema.

Nel caso specifico, si è optato per l'implementazione dei vincoli a livello applicativo. Questa scelta consente un controllo più dettagliato del comportamento del sistema in risposta alla violazione di un vincolo, permettendo di gestire con maggiore precisione i flussi di esecuzione e di restituire messaggi d'errore personalizzati e contestualizzati. Al contrario, l'utilizzo di trigger a livello di database comporterebbe il rischio di ottenere risposte generiche in caso di errore, limitandosi a sollevare eccezioni da parte del DBMS senza fornire

informazioni utili per l'utente o per il controllo del flusso logico dell'applicazione.

3.4 UML

UML, che sta per Unified Modeling Language, è un linguaggio grafico usato per descrivere, progettare e documentare sistemi software. In pratica, serve a rappresentare in modo chiaro e visuale come funzionano le varie parti di un programma o di un sistema, facilitando la comunicazione tra sviluppatori, clienti e altri stakeholder.

Tra i vari tipi di diagrammi che UML offre, ci sono l'Activity Diagram e lo Use Case Diagram. L'Activity Diagram è utile per mostrare il flusso di attività o processi all'interno di un sistema, come una sorta di mappa delle azioni che avvengono in sequenza o in parallelo. Questo aiuta a capire meglio il funzionamento e le interazioni tra le diverse parti. Lo Use Case Diagram, invece, rappresenta le funzionalità del sistema viste dal punto di vista dell'utente, evidenziando quali azioni può compiere e quali obiettivi può raggiungere.

Entrambi i diagrammi sono strumenti molto pratici per analizzare e progettare software, rendendo più semplice sia la fase di sviluppo che quella di comunicazione tra chi realizza il sistema e chi lo utilizza.

3.4.1 Diagramma Use Case

Lo Use Case Diagram è uno strumento che permette di rappresentare in modo chiaro le funzionalità principali del sistema, evidenziando le interazioni tra gli utenti e il software. Questo diagramma facilita la comprensione e l'organizzazione dei requisiti funzionali, come quelli descritti nella sezione 2.2.2, mostrando quali azioni gli utenti possono eseguire e quali obiettivi possono raggiungere attraverso il sistema. In questo modo, lo Use Case Diagram supporta la definizione precisa delle funzionalità richieste e favorisce una migliore comunicazione tra sviluppatori e stakeholder. Di seguito, nella *Figura 14*, è mostrato lo Use Case Diagram dell'applicazione.

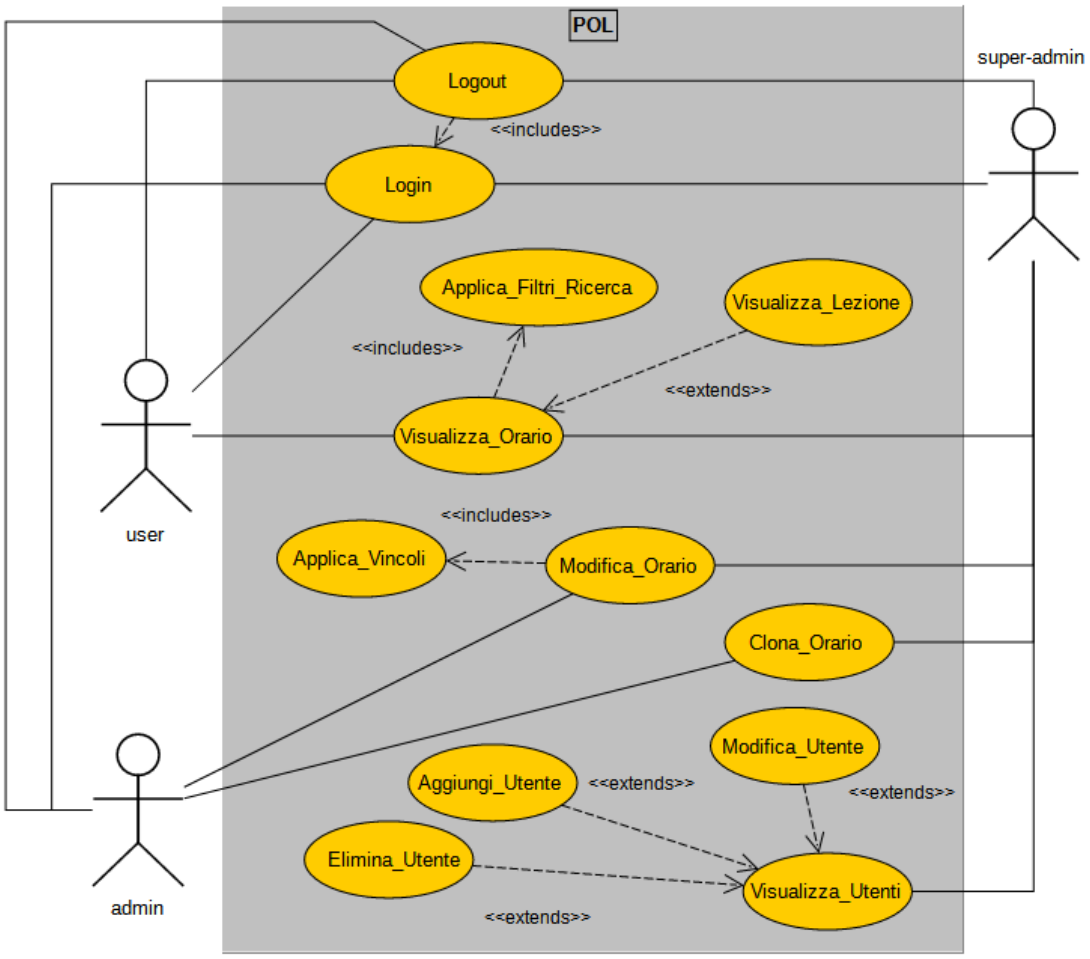

Figura 14: Diagramma Use Case dell'applicazione

Login

- **Nome Use Case:** Login
- **Attori:** user, admin, super-admin
- **Precondizioni:** L'utente deve aprire l'applicazione web
- **Postcondizioni:** L'utente è autenticato ed accede alla home page della web app
- **Scenario principale:**
  1. L'utente avvia l'applicazione
  2. L'applicazione mostra un form con 2 textbox per inserire email e password
  3. L'utente inserisce username e password

4. L’applicazione verifica che i dati di login siano corretti
    1. Se username e password non sono corretti, si torna al punto 2 con un opportuno messaggio di errore
5. Se i dati di login sono corretti l’utente viene reindirizzato alla home page dell’applicazione

Visualizza Orario

- **Nome Use Case:** Visualizza_Orario
- **Attori:** user, admin, super-admin
- **Precondizioni:** L’utente deve aver completato con successo la procedura di login
- **Postcondizioni:** L’utente visualizza in un’apposita tabella i risultati della ricerca effettuata in base ai filtri impostati di default
- **Scenario principale:**
    1. L’utente entra nella home page
    2. Di default, verranno mostrate le lezioni del primo semestre, del lunedì, dell’anno accademico corrente
- **Estensioni:**
  Applica_Filtri_Ricerca
  Visualizza_Lezioni

Applica Filtri Ricerca

- **Nome Use Case:** Applica_Filtri_Ricerca
- **Attori:** user, admin, super-admin
- **Precondizioni:** L’utente deve essere nella schermata di home page in cui vi è l’apposita sezione per visualizzare l’orario delle lezioni
- **Postcondizioni:** L’utente visualizza in un apposita tabella i risultati della ricerca effettuata in base ai filtri che ha impostato
- **Scenario principale:**
    1. L’utente seleziona degli appositi filtri di ricerca tramite un apposita toolbar
    2. Dinamicamente verranno aggiornate le opzioni per effettuare filtri di ricerca (es: se seleziono un CdL, dovranno essere mostrati nei filtri solo i curriculum e gli anni di CdL che appartengono a quel determinato CdL)
    3. L’utente può scegliere di effettuare ricerche andando a selezionare uno o più di questi filtri:
        - Cdl (selezione multipla)
        - Curricula (selezione multipla)

    - Anni Cdl (selezione multipla)
    - Docenti (selezione multipla)
    - Aule (selezione multipla)
    - Semestre (selezione singola)
    - Anno Accademico (selezione singola)
    - Mostra Aule Vuote (checkbox)
    - Giorno (selezione singola)
  4. Una volta impostati tutti i filtri, tramite un apposito bottone si effettuerà la ricerca e verranno mostrare le lezioni nella tabella di visualizzazione orario nella home page
- **Scenario Secondario (a):**

  4a. I filtri applicati non producono alcun risultato, tramite un apposito "banner" verrà informato l'utente

Visualizza Lezioni

- **Nome Use Case:** Visualizza_Lezioni
- **Attori:** user, admin, super-admin
- **Precondizioni:** L'utente deve essere nella schermata di home page in cui vi è l'apposita sezione per visualizzare l'orario delle lezioni
- **Postcondizioni:** L'utente viene reindirizzato in un'apposita pagina in cui sono contenuti tutti i dettagli della lezione selezionata
- **Scenario principale:**
  1. L'utente si trova nella home page in cui è presente la sezione per la visualizzazione dell'orario delle lezioni
  2. L'utente clicca su una lezione presente all'interno della sezione di visualizzazione orario
  3. L'utente viene reindirizzato ad un'apposita pagina in cui vengono mostrati tutti i dettagli della lezione selezionata

Applica Vincoli

- **Nome Use Case:** Applica_Vincoli
- **Attori:** admin, super-admin
- **Precondizioni:** L'utente deve aver eseguito un operazione che comporti la modifica dell'orario delle lezioni, ad esempio modificando l'orario o l'aula di una lezione
- **Postcondizioni:** L'utente viene reindirizzato alla home page visualizzando le modifiche effettuate sull'orario in caso i vincoli risultino essere tutti verificati correttamente
- **Scenario principale:**

1. L’utente esegue un’operazione che comporta la modifica dell’orario
2. Il sistema esegue la verifica di ciascun vincolo
3. Il sistema reindirizza l’utente alla home page nel caso in cui tutti i vincoli siano stati verificati con successo

- **Scenario Secondario (a):**
  2a. Il sistema rileva il fallimento della verifica di uno o più vincoli
  3a. Il sistema mostra un messaggio di errore all’utente specificando i dati del/dei vincolo/i fallito/i

Modifica Orario

- **Nome Use Case:** Modifica_Orario
- **Attori:** admin, super-admin
- **Precondizioni:** L’utente deve essere nella schermata di home page in cui vi è l’apposita sezione per visualizzare l’orario delle lezioni. Solo gli utenti con ruolo di “admin” o “speradmin” possono eseguire questa operazione.
- **Postcondizioni:** L’utente viene reindirizzato alla home page visualizzando le modifiche effettuate sull’orario in caso i vincoli risultino essere tutti verificati correttamente
- **Scenario principale:**
  1. L’utente clicca su un apposito bottone presente all’interno di ciascuna Lezione visualizzata all’interno dell’orario
  2. L’utente viene reindirizzato ad un’apposita pagina che conterrà un form con gli appositi campi per modificare i dati della Lezione selezionata. Il form conterrà i seguenti campi:
     - aule
     - orario
     - semestre
     - giorno
  3. L’utente clicca su un apposito bottone “Modifica” ed il sistema esegue automaticamente il caso d’uso “Applica_Vincoli”
- **Estensioni:**
  Include: Applica_Vincoli

Clona Orario

- **Nome Use Case:** Clona_Orario
- **Attori:** admin, super-admin
- **Precondizioni:** L’utente deve essere nell’area riservata di impostazioni.

Solo gli utenti con ruolo di “admin” o “super-admin” possono eseguire questa operazione.

- **Postcondizioni:** L’utente verrà reindirizzato alla home page mostrando l’orario delle lezioni clonato per il nuovo anno accademico
- **Scenario principale:**
  1. L’utente accede all’area “Impostazioni” dell’applicazione web
  2. L’utente cliccherà su un apposito bottone “clona orario”
  3. L’applicazione mostra un modale in cui va specificato l’anno accademico dal quale si vuole clonare l’orario delle lezioni e l’anno accademico di destinazione
  4. L’utente clicca sul bottone “invia” ed il sistema esegue l’operazione richiesta

Visualizza Utenti

- **Nome Use Case:** Visualizza_Utenti
- **Attori: s**uper-admin
- **Precondizioni:**L’utente deve essere nell’area riservata di impostazioni.
- Solo gli utenti con ruolo di “super-admin” possono eseguire questa operazione.
- **Postcondizioni:** L’utente visualizza tutti gli utenti del sistema in un’apposita tabella
- **Scenario principale:**
  1. L’utente entra nella sezione “Impostazioni” dell’applicazione web
  2. L’applicazione mostra tutti gli utenti del sistema con i relativi dati
- **Estensioni:**
  Modifica_Utente
  Elimina_Utente
  Crea_Utente

Modifica Utente

- **Nome Use Case:** Modifica_Utente
- **Attori: s**uper-admin
- **Precondizioni:** L’utente deve essere nell’area riservata di impostazioni.
- Solo gli utenti con ruolo di “super-admin” possono eseguire questa operazione.

- **Postcondizioni:** Una volta applicate le modifiche ai dati di un determinato utente, l'utente che ha eseguito l'operazione viene reindirizzato alla pagina di impostazioni dell'applicazione web
- **Scenario principale:**
  1. L'utente entra nella sezione "Impostazioni" dell'applicazione web
  2. L'utente clicca su un apposito bottone "Modifica" presente in ogni riga della tabella di visualizzazione utenti in modo tale da selezionare l'utente da modificare
  3. L'applicazione mostra un modale in cui l'utente può modificare i dati dell'utente selezionato tramite un apposito form con i seguenti campi:
     - ruolo
     - name
     - password
  4. L'utente inserisce i dati che desidera modificar all'interno degli appositi campi e poi clicca sul bottone "Salva"
  5. Il sistema modifica i dati dell'utente selezionato con i dati inseriti nel form ed esegue il logout di quest'ultimo da tutti i dispositivi

Aggiungi Utente

- **Nome Use Case:** Aggiungi_Utente
- **Attori: s**uper-admin
- **Precondizioni:** L'utente deve essere nell'area riservata di impostazioni. Solo gli utenti con ruolo di "super-admin" possono eseguire questa operazione.
- **Postcondizioni:** Una volta creato il nuovo utente, l'utente che ha eseguito l'operazione viene reindirizzato alla pagina di impostazioni dell'applicazione web
- **Scenario principale:**
  1. L'utente entra nella sezione "Impostazioni" dell'applicazione web
  2. L'utente clicca sull'apposito bottone "Aggiungi Utente"
  3. L'applicazione mostra un modale in cui è presente un form con i seguenti campi:
     - name
     - e-mail
     - password
     - ruolo
  4. L'utente compila tutti i campi del form e clicca sull'apposito bottone "Salva"
  5. Il sistema crea l'utente a partire dia campi del form compilati

Elimina Utente

- **Nome Use Case:** Elimina_Utente
- **Attori: s**uper-admin
- **Precondizioni:** L'utente deve essere nell'area riservata di impostazioni. Solo gli utenti con ruolo di "super-admin" possono eseguire questa operazione
- **Postcondizioni:** Una volta eliminato un determinato utente, l'utente che ha eseguito l'operazione viene reindirizzato alla pagina di impostazioni dell'applicazione web
- **Scenario principale:**
  1. L'utente entra nella sezione "Impostazioni" dell'applicazione web
  2. L'utente clicca su un apposito bottone "Elimina" presente in ogni riga della tabella di visualizzazione utenti in modo tale da selezionare l'utente da eliminare
  3. L'applicazione mostra un modale che richiede conferma all'utente
  4. Il sistema elimina l'utente selezionato ed esegue il logout da tutti i dispositivi

3.4.2.Diagramma delle attività

Il diagramma delle attività è uno strumento utilizzato per rappresentare il flusso delle operazioni o dei processi all'interno di un sistema. Per una web app, questo strumento serve a descrivere in modo chiaro e visuale le sequenze di azioni che l'utente o il sistema possono eseguire, evidenziando decisioni, condizioni e flussi paralleli. Nell'applicazione corrente verrà utilizzato per descrivere il flusso di navigazione e delle operazioni all'interno della Web App. Di seguito è riportato, nella *Figura 15*, il diagramma UML delle attività.

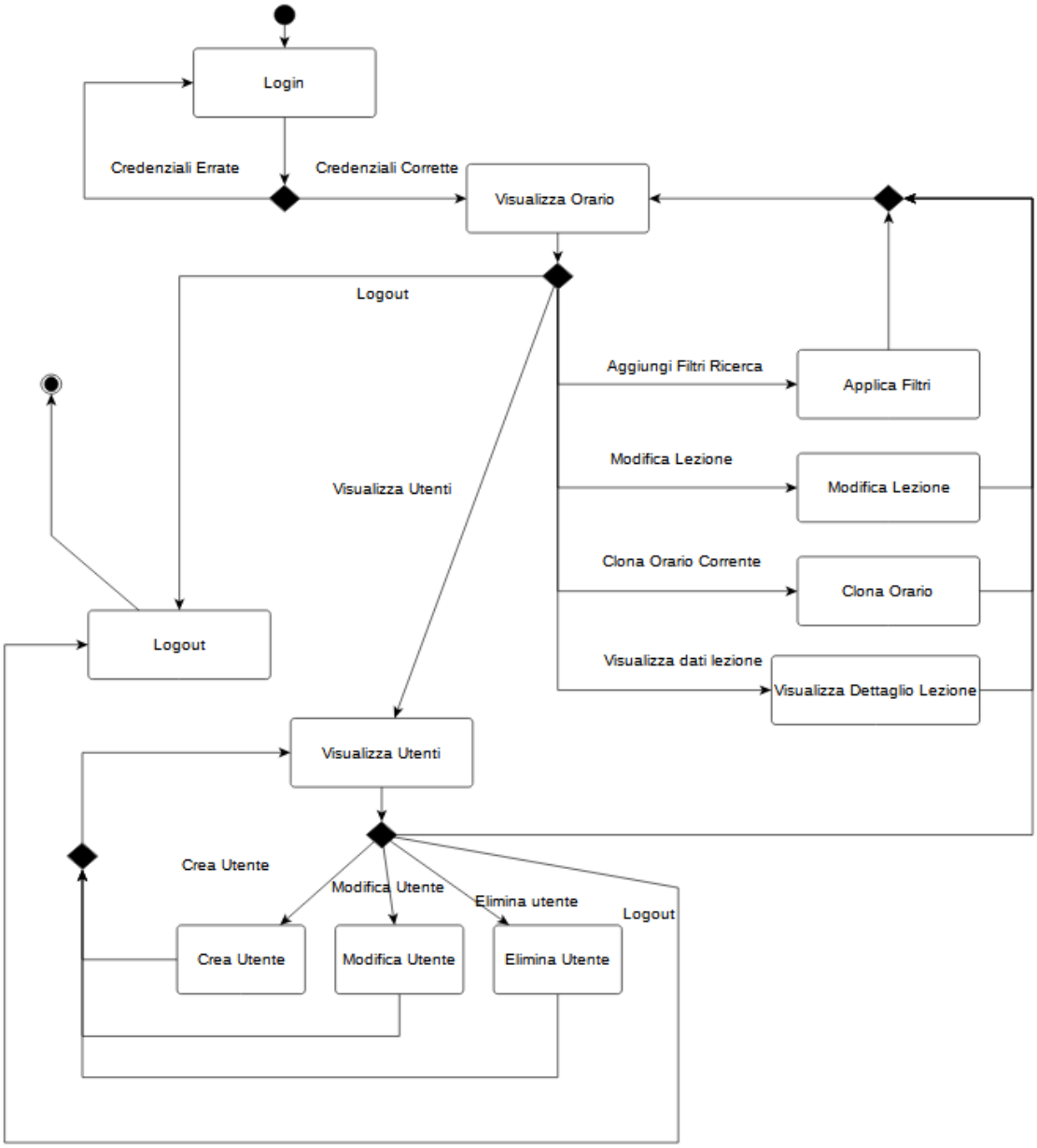

Figura 15: Diagramma UML delle attività

Capitolo 4

4 Sviluppo

La fase di sviluppo dell'applicazione comincia con la realizzazione dei mockup dell'interfaccia utente, utili per avere un'anteprima di come sarà strutturato il sistema dal punto di vista grafico e funzionale. Una volta definiti i principali elementi dell'interfaccia, si passa alla scrittura del codice vero e proprio utilizzando Laravel come framework per la parte backend. Questo consente di organizzare in modo ordinato le varie componenti dell'applicazione, come rotte, controller, modelli e viste. Il frontend viene realizzato sfruttando il sistema di template Blade e integrando HTML, CSS e JavaScript per rendere l'interfaccia interattiva. La logica di business implementata nel backend interagisce costantemente con il database MySQL, progettato nella fase precedente, attraverso il sistema ORM Eloquent integrarto In Laravel. che semplifica l'accesso e la manipolazione dei dati. Di seguito è mostrato, nella *Figura 16*, uno schema che raffigura l'architettura dell'applicazione:

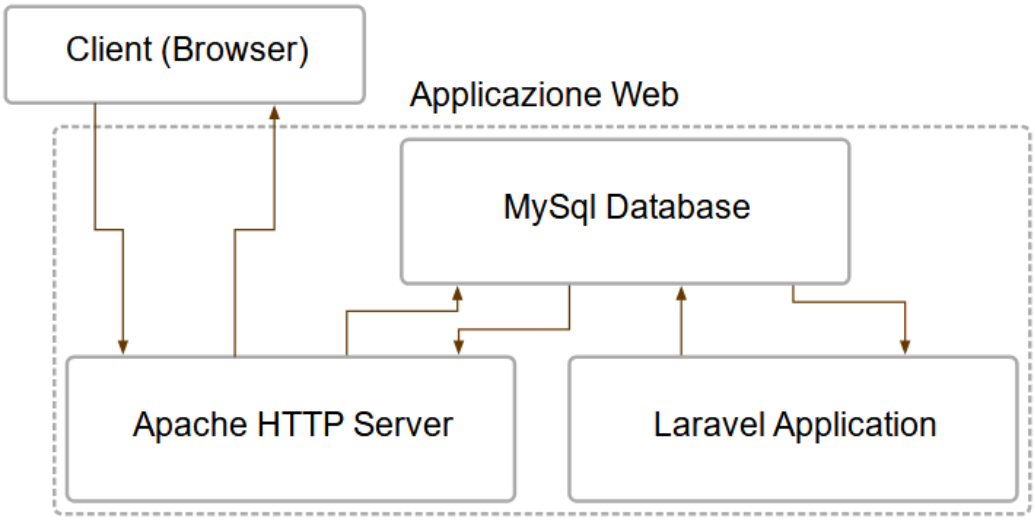

Figura 16: Architettura dell'applicazione web

4.1 Mockup Applicazione Web

Il mockup di un'applicazione web è una rappresentazione statica che riproduce in modo preciso la struttura e l'organizzazione dell'interfaccia, senza necessariamente includere elementi grafici decorativi. Il suo scopo principale è quello di definire la disposizione dei contenuti, la gerarchia delle informazioni e la logica di navigazione tra le varie sezioni dell'applicazione. Viene utilizzato nella fase progettuale per anticipare il comportamento dell'interfaccia, facilitando il confronto tra sviluppatori e committenti e riducendo il rischio di fraintendimenti nella successiva fase di implementazione. Anche privo di colori o dettagli stilistici, il mockup fornisce una base solida su cui costruire la versione interattiva dell'app

4.1.1 Login

Di seguito è riportato, nella *Figura 17*, il mockup della pagina di login.

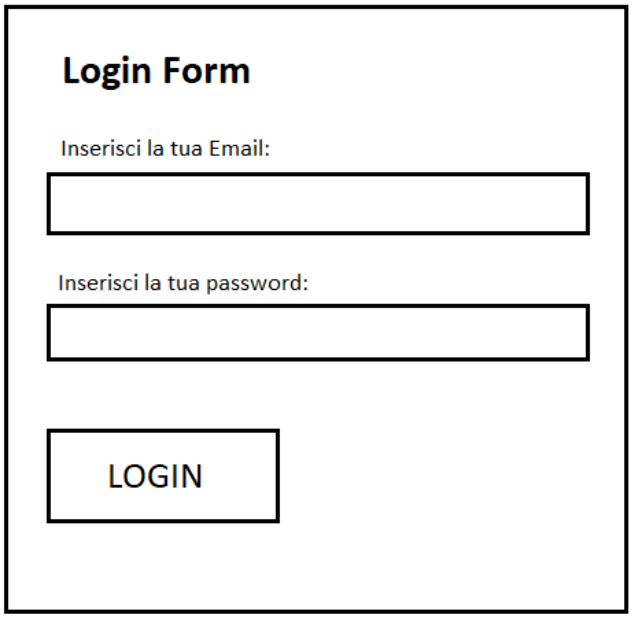

Figura 17: Mockup della pagina di Login

La schermata di login contiene 2 input per permettere all'utente del sistema di autenticarsi tramite e-mail e password. Nel caso in cui l'autenticazione fallisca, verrà mostrato all'utente un messaggio di errore all'interno della stessa schermata. Una volta autenticatosi con successo, l'utente verrà reindirizzato alla schermata da visualizzazione orario.

4.1.2 Visualizza Orario

Di seguito è riportato, nella *Figura 18*, il mockup della home page.

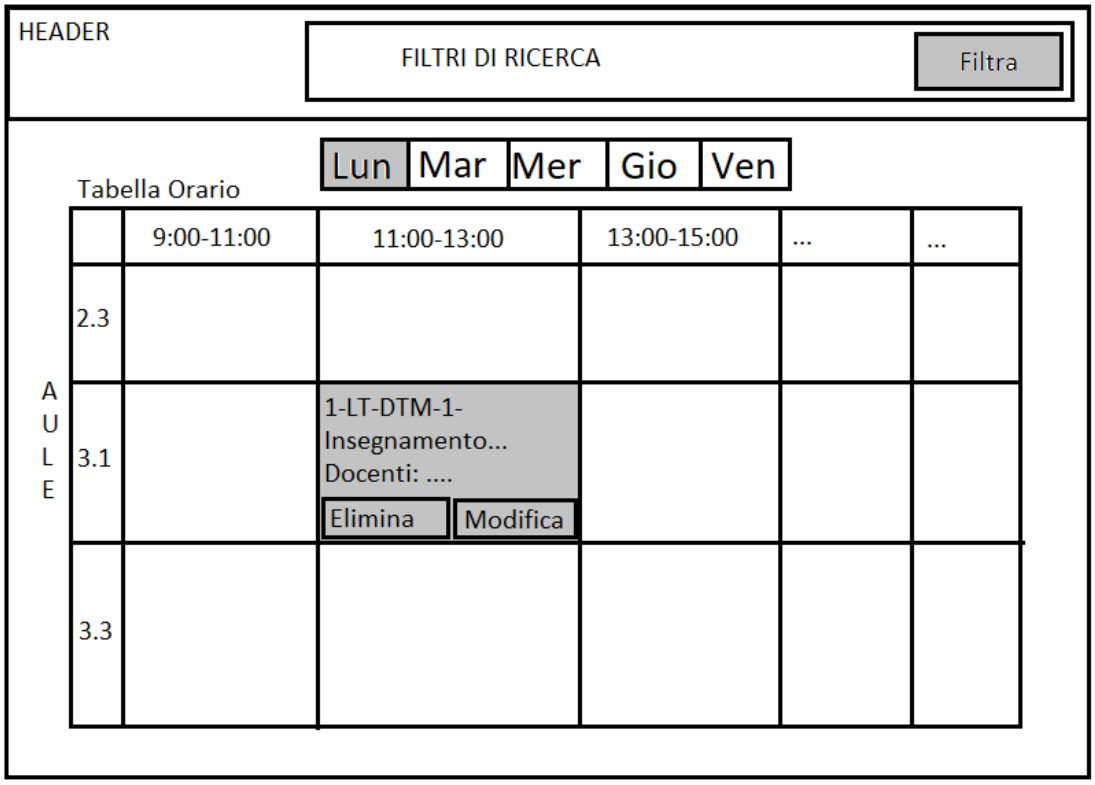

Figura 18: Mockup della Home Page

La schermata di visualizzazione dell'orario costituisce la home page dell'applicazione, ovvero il primo punto di accesso per l'utente una volta completata con successo la procedura di autenticazione. Nella sezione denominata "HEADER", sono presenti i principali filtri per la ricerca delle lezioni, accompagnati da un apposito pulsante che consente di avviare la ricerca in base ai criteri selezionati. Subito sotto si trova una barra orizzontale suddivisa in cinque sezioni, ciascuna delle quali rappresenta un giorno della settimana accademica. Attraverso questa barra, l'utente può selezionare il giorno desiderato su cui applicare i filtri impostati. La parte centrale della schermata è occupata dalla tabella di visualizzazione dell'orario, organizzata secondo una griglia in cui le colonne corrispondono alle aule e le righe agli slot orari, con intervalli di un'ora. Le lezioni sono rappresentate da riquadri colorati, posizionati in base

all'orario e all'aula assegnata. Ogni riquadro contiene le informazioni principali relative alla lezione, tra cui il docente, l'eventuale mutuazione, il corso di laurea e il curriculum di riferimento. All'interno di ciascuna cella occupata da una lezione l'utente ha la possibilità di eseguire due operazioni: eliminarla, attraverso il pulsante "Elimina", oppure modificarne la collocazione tramite il pulsante "Sposta". In quest'ultimo caso, l'applicazione reindirizza automaticamente alla schermata dedicata alla modifica della lezione. È rilevante sottolineare che le funzioni di eliminazione e spostamento sono abilitate esclusivamente per gli utenti dotati dei privilegi di tipo admin o super-admin.

4.1.3 Modifica Lezione

Di seguito è riportato, nella *Figura 19*, il mockup della pagina per la modifica e la visualizzazione dei dettagli di una specifica lezione.

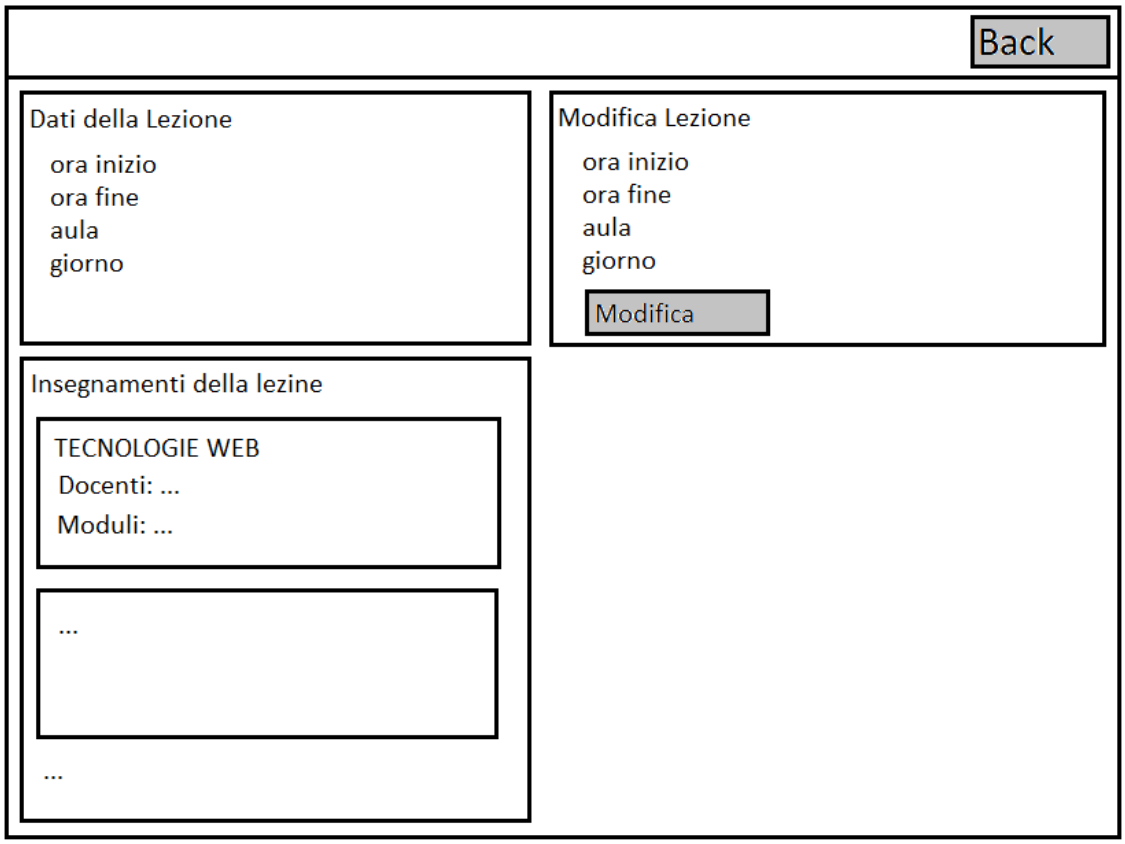

Figura 19: Mockup della pagina per la visualizzazione e la modifica di una Lezione

La schermata dedicata alla modifica di una lezione è articolata in tre sezioni principali. La prima sezione presenta i dati generali della lezione, quali l'orario, il giorno e l'aula assegnata. La seconda sezione è invece focalizzata sugli insegnamenti collegati alla lezione selezionata: per ciascuno di essi vengono visualizzate le relative informazioni, compresi i docenti responsabili e i moduli didattici associati. La terza e ultima sezione è costituita da un form che consente all'utente di apportare modifiche ai parametri principali della lezione, con la possibilità di aggiornare il giorno, l'aula e l'orario. Al termine della compilazione, l'utente può confermare le modifiche attraverso l'apposito pulsante. In assenza di errori, il sistema procede con il salvataggio dei dati e reindirizza l'utente alla schermata precedente. Qualora invece vengano rilevate incongruenze o errori nella compilazione, questi ultimi verranno segnalati in modo esplicito all'interno dell'interfaccia, consentendo all'utente di intervenire per correggerli.

4.1.3 Impostazioni

Di seguito è riportato, nella *Figura 20*, il mockup della pagina di impostazioni.

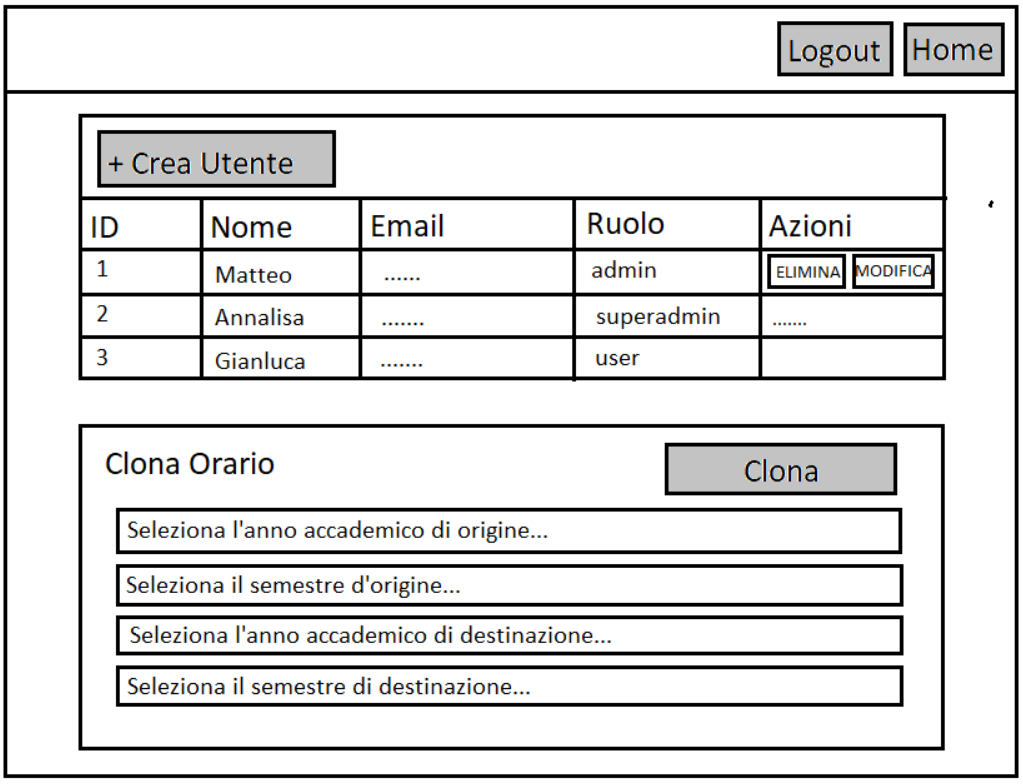

Figura 20: Mockup della pagina di impostazioni

Tutti gli utenti abilitati all'accesso al sistema hanno la possibilità di visualizzare l'interfaccia dedicata alle impostazioni. Le funzionalità offerte in questa sezione dipendono dal ruolo attribuito a ciascun utente. In particolare, solo gli utenti con privilegi di tipo admin o super-admin possono accedere alla sezione denominata "Clona Orario", riservata ad attività di gestione avanzata. Per procedere con la clonazione di un orario, è necessario compilare il form che richiede l'inserimento dell'anno accademico e del semestre di origine, oltre all'anno accademico e al semestre di destinazione.

Gli utenti con ruolo admin e user possono accedere unicamente alle informazioni relative al proprio profilo e hanno la possibilità di modificarle attraverso un'apposita finestra modale, accessibile tramite il pulsante "Modifica". In tale finestra è possibile aggiornare esclusivamente il nome utente e la password. Per questi ruoli non è prevista alcuna funzionalità legata alla creazione o eliminazione di altri utenti e di conseguenza l'interfaccia non presenta i comandi relativi a tali operazioni.

Al contrario, gli utenti super-admin hanno accesso completo alle funzionalità di gestione degli account. Possono visualizzare l'intero elenco degli utenti registrati, crearne di nuovi, modificarne i dati o procedere alla loro eliminazione. La creazione di un nuovo utente avviene tramite l'apposito bottone "Crea Utente", che apre una finestra modale nella quale è possibile inserire tutte le informazioni necessarie, compresa la definizione del ruolo.

4.2 Sviluppo Web App

In questa fase del progetto viene sviluppata l'applicazione web dedicata alla gestione e programmazione delle lezioni, seguendo i requisiti funzionali, le modalità operative, la struttura del database e il mockup descritti nei capitoli precedenti. Per la realizzazione è stato scelto PHP come linguaggio di programmazione lato backend, avvalendosi del framework Laravel, che rappresenta una delle soluzioni più diffuse e consolidate nell'ambito dello sviluppo web moderno.

Laravel offre un sistema integrato per la costruzione dell'interfaccia utente attraverso Blade, un sistema di template che consente di combinare codice HTML e PHP in modo ordinato ed efficiente. L'utilizzo di JavaScript permette inoltre di introdurre interattività e dinamicità nelle pagine, migliorando l'esperienza dell'utente finale.

Per quanto riguarda l'interazione con il database, vengono configurati i modelli Eloquent messi a disposizione dal framework. Questo ORM (Object-Relational Mapping) semplifica notevolmente la gestione dei dati, permettendo di interagire con le tabelle attraverso oggetti e metodi orientati agli oggetti, senza dover scrivere query SQL complesse.

Laravel si basa sul pattern architetturale MVC (Model-View-Controller), che consente di mantenere una chiara separazione tra logica applicativa, interfaccia utente e gestione dei dati. Questo approccio facilita la scalabilità del progetto e ne migliora la manutenibilità nel tempo. Nelle sezioni successive verranno approfonditi alcuni aspetti tecnici legati al funzionamento del framework e all'implementazione del codice PHP.

4.2.1 Framework

Laravel è un framework open source per lo sviluppo di applicazioni web basato sul linguaggio PHP, progettato per offrire una struttura solida e ben organizzata che faciliti la scrittura di codice chiaro e manutenibile. Si fonda sull'architettura Model-View-Controller (MVC), che consente di mantenere una netta separazione tra la logica applicativa, la gestione dei dati e l'interfaccia utente. All'interno di questo modello, i controller coordinano il flusso delle informazioni tra i modelli, che rappresentano le entità e l'accesso al database, e le view, responsabili della visualizzazione dei contenuti.

Il framework mette a disposizione una serie di strumenti integrati che semplificano operazioni comuni, come la gestione delle rotte, l'autenticazione, la validazione dei dati e l'invio di notifiche. L'ORM Eloquent è particolarmente rilevante: consente un'interazione efficace con il database attraverso un approccio orientato agli oggetti. L'insieme di queste caratteristiche rende Laravel una soluzione efficace per lo sviluppo di applicazioni web scalabili, sicure e facilmente estendibili. Di seguito è riportato, nella *Figura 21*, un diagramma che mostra l'interazione tra i vari componenti del pattern MVC.

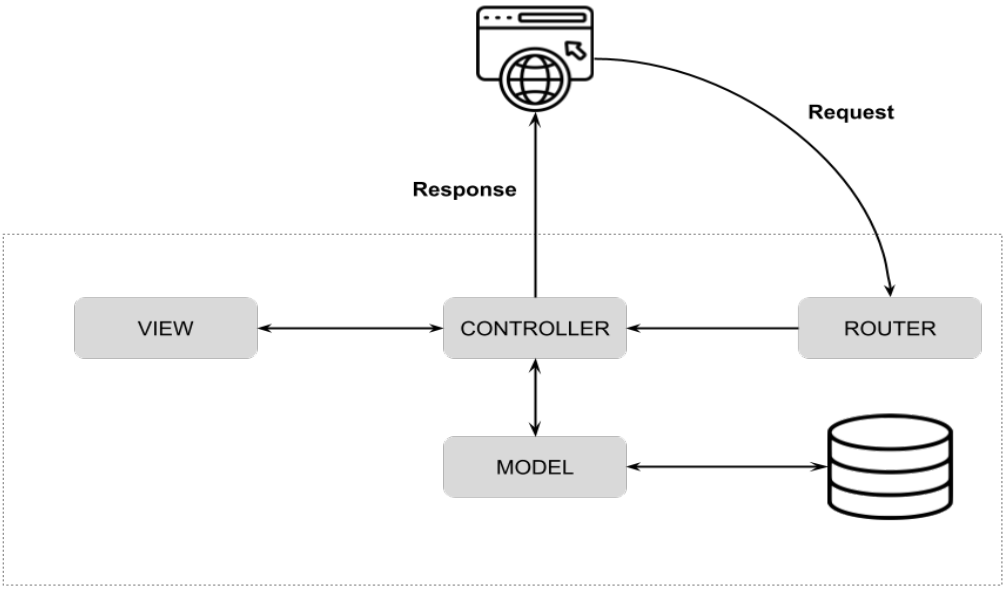

Figura 21: Diagramma pattern MVC

4.2.2 ORM

Eloquent è l'Object-Relational Mapping (ORM) integrato in Laravel, progettato per semplificare l'interazione con il database attraverso un approccio orientato agli oggetti. Ogni tabella del database viene rappresentata da una classe modello e ogni record corrisponde a un'istanza di quella classe, consentendo di eseguire operazioni come inserimenti, aggiornamenti, cancellazioni e query in modo intuitivo e leggibile. Le relazioni tra tabelle, come quelle di tipo uno-a-molti o molti-a-molti, sono facilmente definibili attraverso metodi specifici all'interno dei modelli, senza dover scrivere query SQL esplicite. Segue un esempio semplificato della definizione della classe Modulo:

```php
<?php

class Modulo extends Model
{
   protected $table = "modulo";
   protected $primaryKey="cod_modulo";
   public $timestamps = false;

   protected $fillable = [
      'cod_modulo',
      'nome',
      'descrizione',
```

```
        'ore_frontali',
        'ore_ese',
       'ore_lab',
        'moltiplicatore',
       'cod_docente',
    ];

    public function Docente(): BelongsTo
    {
        return $this->belongsTo(Docente::class, "cod_docente", "cod_docente");
     }
}
```

In Eloquent, le classi che estendono la classe base Model vengono interpretate come rappresentazioni delle tabelle presenti nel database. Il framework è in grado di dedurre automaticamente il nome della tabella associata a partire dal nome della classe, utilizzando una convenzione al plurale in minuscolo (ad esempio, la classe Modulo sarà associata alla tabella Modulo). Tuttavia, è possibile specificare manualmente il nome della tabella attraverso la proprietà $table, qualora si voglia disattendere la convenzione predefinita.

Ogni modello definisce un array $fillable che indica esplicitamente i campi della tabella che possono essere valorizzati tramite assegnazione di massa, come nel caso della creazione o dell'aggiornamento di un record. Anche la chiave primaria viene per impostazione predefinita identificata da Eloquent come un campo denominato id, ma può essere personalizzata mediante la proprietà $primaryKey.

Un aspetto particolarmente rilevante riguarda la definizione delle relazioni tra modelli. Nel caso in esame, un modulo è associato a un docente, mentre un docente può essere associato a più moduli: si tratta quindi di una relazione uno-a-molti. Eloquent mette a disposizione metodi specifici per la mappatura delle relazioni, tra cui belongsTo, hasMany e belongsToMany, che consentono di rappresentare correttamente i vari tipi di associazioni tra entità.

Nella classe Modulo, ad esempio, è definito un metodo docente che restituisce un valore di tipo BelongsTo. Questo permette a Eloquent di interpretare correttamente il legame logico tra la tabella dei moduli e quella dei docenti, riconoscendo che ciascun modulo appartiene a un singolo docente e che quest'ultimo può a sua volta essere collegato a più moduli. Segue un esempio di utilizzo della classe "Modulo" per operazioni CRUD:

```php
//SeLezione con clausola di where
$moduli=Modulo::where("ore_lab",">",0)->get();
//creazione
$nuovo_modulo=Modulo::create([
   'nome'=>'modulo_di_test',
   'descrizione'=>'test',
   'ore_frontali'=>30,
   'ore_ese'=>10,
   'ore_lab'=>5,
   'moltiplicatore'=>0,
   'cod_docente'=>7,
]);
//modifica
$nuovo_modulo->ore_lab=0;
$nuovo_modulo->save();
//utilizzo di associazioni
$docente=$nuovo_modulo->docente;
//eliminazione
$nuovo_modulo->delete();
```

4.2.3 Classi Rilevanti

Tra le componenti centrali dell'applicazione rivestono un ruolo fondamentale quelle deputate alla visualizzazione dell'orario e alla verifica dei vincoli. In particolare, per la gestione dell'orario, tre classi si distinguono per importanza: HomeController, LezioneResource e OrarioResource.

La classe HomeController, che estende la classe base Controller di Laravel, ha il compito di intercettare le richieste provenienti dal client, eseguire le opportune elaborazioni dei dati e restituire una vista contenente tutte le informazioni necessarie per la corretta visualizzazione lato utente.

Poiché ogni lezione è associata a numerosi dati correlati (come l'insegnamento, la classe, il docente, l'aula, e altri elementi) si è reso necessario l'impiego delle JsonResource messe a disposizione da Laravel. Tali classi consentono di trasformare oggetti del modello in rappresentazioni JSON personalizzate. A tal fine, è stata implementata la classe LezioneResource, che estende appunto JsonResource, con l'obiettivo di ricevere in ingresso un oggetto di tipo lezione e restituire una struttura JSON contenente tutte le informazioni associate in modo aggregato e coerente.

La classe OrarioResource segue un'impostazione analoga, estendendo anch'essa JsonResource, ma con un ulteriore livello di complessità. Essa accetta nel costruttore diversi parametri rappresentanti i filtri di ricerca e si occupa di

aggregare le lezioni in base alle aule corrispondenti, secondo i criteri specificati. In questo modo, è possibile ottenere una rappresentazione dell'orario dinamicamente filtrata.

Di seguito sono mostrate le 3 classi:

- LezioneResource
- OrarioResource
- HomeController

```php
class LezioneResource extends JsonResource
{
        protected LezioneService $service;
        public function __construct(Lezione $Lezione){
                $this->service=new LezioneServiceImpl($Lezione);
        }
        public function toArray(Request $request): array{
                $insegnamenti_collection=new Collection();
                $classi_collection=new Collection();
                foreach($this->insegnamenti as $Insegnamento){
                        $resouce=new InsegnamentoResource($Insegnamento);
                        $insegnamenti_collection->push($resouce->resolve());
                }
                foreach($this->classi as $classe){
                        $resource=new ClasseResource($classe);
                        $classi_collection->push($resource->resolve());
                }
                return array_merge($this->Lezione->toArray(),[
                        'insegnamenti'=>$insegnamenti_collection,
                        'aule'=>$this->service->getAule(),
                        'classi'=>$classi_collection,
                        'durata'=>$this->service->getDurata(),
                        'ora_inizio'=>$this->service->getOraInizio(),
                        'ora_fine'=>$this->service->getOraFine()
                ]);
        }
}
```

```
class OrarioResource extends JsonResource
{
        protected Collection $aule;
        protected int $aa;
        protected int $semestre;
        protected int $giorno;
        protected Collection $curricula;
        protected Collection $docenti;
        protected Collection $cdl;
        protected Collection $anni;
        protected bool $mostra_aule_vuote;

        public function __construct(Collection $aule, int $aa, int $semestre, int $giorno,
                Collection $cdl, Collection $curricula, Collection $anni, Collection
                        $docenti, bool $mostra_aule_vuote){
                $this->aule = $aule;
                $this->aa = $aa;
                $this->semestre = $semestre;
                $this->giorno = $giorno;
                $this->cdl = $cdl;
                $this->curricula = $curricula;
                $this->anni = $anni;
                $this->docenti = $docenti;
                $this->mostra_aule_vuote = $mostra_aule_vuote;
        }

        public function toArray(Request $request): array{
                $response = collect();
                foreach ($this->aule as $aula) {
                        $aulaService = new AulaServiceImpl($aula);
                        $lezioni = collect($aulaService->getLezioni($this->aa,
                                $this->semestre, $this->giorno));
                        $filterManager = new LezioneFilterManager();
                        $filterManager
                                ->addFilter(new CdlLezioneFilter($this->cdl))
                                ->addFilter(new CurriculumLezioneFilter($this->curricula))
                                ->addFilter(new AnnoLezioneFilter($this->anni))
                                ->addFilter(new DocenteLezioneFilter($this->docenti));
                        $lezioni_filtered      =      $filterManager->applyFilters($lezioni)-
>sortBy('orario_inzio')->values();
                        $lezioniJson = $lezioni_filtered->map(function ($lezione) {
                                return (new LezioneResource($lezione))->resolve();
                        });
                        if ($this->mostra_aule_vuote || $lezioniJson->isNotEmpty()) {
                                $response->push([
                                        'aula' => $aula,
                                        'lezioni' => $lezioniJson,
                                ]);
                        }
                }
                return $response->toArray();
        }
}
```

```php
class HomeController extends Controller
{
        public function index(Request $request){
                $request->validate([
                        'aa' => 'integer|nullable',
                        'giorno' => 'required|integer',
                        'semestre' => 'required|integer',
                        'aule' => 'nullable',
                        'docenti' => 'nullable',
                        'anni' => 'nullable',
                        'cdl' => 'nullable',
                        'curricula' => 'nullable',
                        'mostra_aule_vuote' => 'nullable'
                ]);
                $aa = $request->input('aa', AnnoAccademico::max('aa'));

                $aule = $request->has('aule') ? collect(json_decode(urldecode($request->aule)))->map(fn($a) => Aula::where('numero', $a->numero)->where('cod_sede', $a->cod_sede)->first())->filter() : Aula::all();

                $docenti = $request->has('docenti') ? collect(json_decode(urldecode($request->docenti))) : new Collection();

                $cdls = $request->has('cdl') ? collect(json_decode(urldecode($request->cdl))) : new Collection();

                $curricula = $request->has('curricula') ? collect(json_decode(urldecode($request->curricula))) : new Collection();

                $anni = $request->has('anni') ? collect(json_decode(urldecode($request->anni))) : new Collection();

                $mostraAuleVuote = $request->boolean('mostra_aule_vuote');

                $response=new OrarioResource($aule,$aa,$request->semestre,$request->giorno,
                        $cdls,$curricula,$anni,$docenti,$mostraAuleVuote);

                return view('index',[
                        'lezioniaule'=>$response->resolve(),
                        'anniaccademici'=>AnnoAccademico::all(),
                        'curricula'=>Curriculum::all(),
                        'docenti'=>Docente::select('*')->orderBy('nome')->get(),
                        'aule'=>Aula::all(),
                ]);
        }
}
```

Come si può osservare, la classe HomeController gestisce tutte le richieste HTTP relative all'URL della home page ("/"). Al suo interno vengono decodificati e interpretati tutti i filtri opzionali eventualmente inclusi nella richiesta, mentre i parametri obbligatori (come il semestre e il giorno) vengono sempre inizializzati. Tutti gli altri filtri, come anno accademico, aule, docenti, corsi di studio, curricula e anni di corso, sono invece facoltativi e vengono elaborati solo se presenti.

Una volta raccolti e validati i parametri, il controller crea un'istanza della classe OrarioResource, passando ad essa tutti i filtri ricevuti. La resource ha il compito di aggregare ed elaborare i dati relativi alle lezioni, restituendo una struttura completa e coerente che sarà poi trasmessa alla vista. In questo modo, il frontend riceve tutte le informazioni necessarie per la corretta visualizzazione dell'orario, filtrato secondo i criteri selezionati dall'utente.

Un insieme di classi assume un ruolo fondamentale nella gestione e nella verifica dei vari vincoli del sistema. Per rappresentare il concetto di "vincolo" in modo modulare e scalabile, è stata adottata l'architettura basata sul pattern "Chain Of Responsibility". Il diagramma UML sottostante illustra l'organizzazione e l'interazione delle componenti che, attraverso la delega progressiva, implementano la logica di validazione dei vincoli. Di seguito è riportato, nella *Figura 22*, il digramma delle classi che descrive il pattern "Chain of Responsibility".

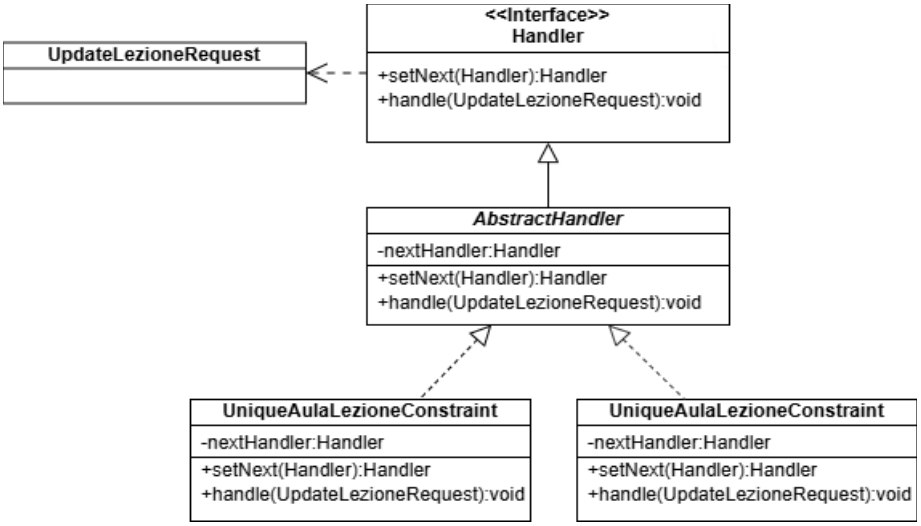

Figura 22: Diagramma UML del pattern "Chain Of Responsibility"

Come si può osservare dal diagramma UML, esiste una relazione di tipo dipendenza tra l'interfaccia Handler e la classe UpdateLezioneRequest. Tale relazione è giustificata dal fatto che ogni Handler riceve, come parametro del metodo handle, un'istanza della classe UpdateLezioneRequest. In questo modo, ciascuna implementazione del metodo handle può operare direttamente sulla richiesta di aggiornamento di una lezione, rappresentata secondo le convenzioni del framework Laravel. Questa struttura consente l'applicazione del pattern Chain of Responsibility, in cui ogni Handler incarna uno specifico vincolo o controllo previsto dal sistema. La richiesta di modifica attraversa dunque una catena di oggetti, ognuno dei quali ha la responsabilità di validare o elaborare un particolare aspetto della richiesta stessa. Segue un esempio di classe che estende AbstractHandler, implementando una specifica regola nella catena di responsabilità.

```
class UniqueAulaLezioneConstraint extends AbstractHandler{

        public function handle(UpdateLezioneRequest $req): void
        {
                $aule=new Collection();
                $lezione=Lezione::findOrFail($req->id_Lezione);
                $ora_inizio_time=Carbon::createFromTime($req->ora_inzio,0,0)-
>format('H:i:s');
                $ora_fine_time=Carbon::createFromTime($req->ora_fine,0,0)->format('H:i:s');
                $aule_json=json_decode($req->aule);

                foreach($aule_json as $aula_json){
                        if(Aula::where("numero",$aula_json->numero)
                        ->where("cod_sede",$aula_json->cod_Sede)
                        ->exists()){
                                $aule->push(Aula::where("numero",$aula_json->numero)
                                ->where("cod_sede",$aula_json->cod_Sede)
                                ->first());
                        }else{
                                throw new NotFound("Aula con numero=".$aula_json->numero." e
cod_Sede=".$aula_json->cod_Sede." non è stata trovata nel database");
                        }
                }

                if(Lezione::join('LezioniAule', 'Lezione.id', '=', 'LezioniAule.id')
                ->where("Lezione.orario_inizio","<",$ora_fine_time)
                ->where("Lezione.orario_fine",">",$ora_inizio_time)
                ->where('Lezione.giorno', '=', $req->giorno)
                ->where('Lezione.semestre', '=', $lezione->semestre)
                ->where('Lezione.aa', '=', $lezione->aa)
                ->whereNotIn("Lezione.id",[$lezione->id])
                ->whereIn("LezioniAule.numero",$aule->map(function($aula){return      $aula-
>numero;}))
```

```
                ->whereIn("LezioniAule.cod_sede",$aule->map(function($aula){return $aula->cod_sede;}))
                ->exists()){
                        throw new ConstraintFailedException("Ci sono già delle lezioni nell'orario stabilitio nelle aule: ".$aule->map(function($aula){return $aula->numero;})->__tostring());
                }
                parent::handle($req);
        }
}
```

Nell'esempio è riportato il codice della classe incaricata di verificare il vincolo relativo alla disponibilità delle aule. In particolare, il sistema non consente la sovrapposizione di due o più lezioni nello stesso orario e nella medesima aula, all'interno di un determinato semestre e anno accademico.

La classe ha il compito di estrarre dalla richiesta, ricevuta dal precedente handler nella catena, i dati necessari per effettuare il controllo. Viene quindi verificato che non risultino già programmate, per lo stesso intervallo temporale e nella stessa aula, altre lezioni in conflitto con quella contenuta nella richiesta. Qualora il vincolo non venga rispettato, viene sollevata un'eccezione di tipo ConstraintFailedException. In tal caso, l'eccezione impedisce il passaggio al successivo handler, interrompendo l'esecuzione della catena. L'errore verrà infine intercettato dal controller, che restituirà un messaggio di notifica al frontend.

4.3 Test

Per testare l'applicazione in modo accurato, si è scelto di concentrarsi sulle funzionalità più dettagliate, in particolare su quelle che riflettono le operazioni svolte quotidianamente dalla segreteria. A tal fine, è stato predisposto uno script PHP che ha permesso di importare nel database l'intero orario delle lezioni, nonché la struttura completa dei corsi di studio e dei relativi curricula per l'anno accademico 2024/25. Questo ha consentito di ricreare un contesto realistico e coerente, utile per effettuare verifiche approfondite.

Successivamente, attraverso il confronto con il foglio Excel ufficiale fornito dalla segreteria, è stato possibile validare l'accuratezza dei filtri di ricerca e della visualizzazione dei dati nel sistema, confermandone il corretto comportamento rispetto alle informazioni ufficiali.

Per quanto riguarda invece la verifica dei vincoli applicativi, sono state effettuate modifiche mirate su specifiche lezioni già presenti a sistema, con l'obiettivo di forzare situazioni di conflitto e osservare la reazione dell'applicazione. In tutti i casi testati, l'interfaccia ha correttamente restituito i messaggi di errore previsti, dimostrando l'efficacia delle logiche di controllo implementate e il corretto funzionamento della catena di responsabilità. Segue una panoramica delle schermate principali dell'interfaccia dell'applicazione.

Schermata Login

La schermata di login presenta due campi di input testuali, nei quali l'utente deve inserire la propria e-mail e password. Se le credenziali sono corrette e l'autenticazione ha esito positivo, l'utente verrà reindirizzato alla pagina Home dell'applicazione. In caso contrario, verrà visualizzato un messaggio di errore. Di seguito, in *Figura 23*, è mostrata la schermata di login dell'applicazione web.

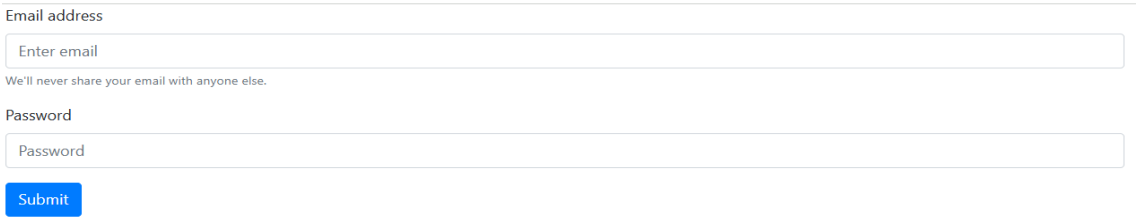

Figura 23: Schermata di Login

Schermata Home

Di seguito è riportata, nella *Figura 24*, la schermata di home page dell'applicazione web

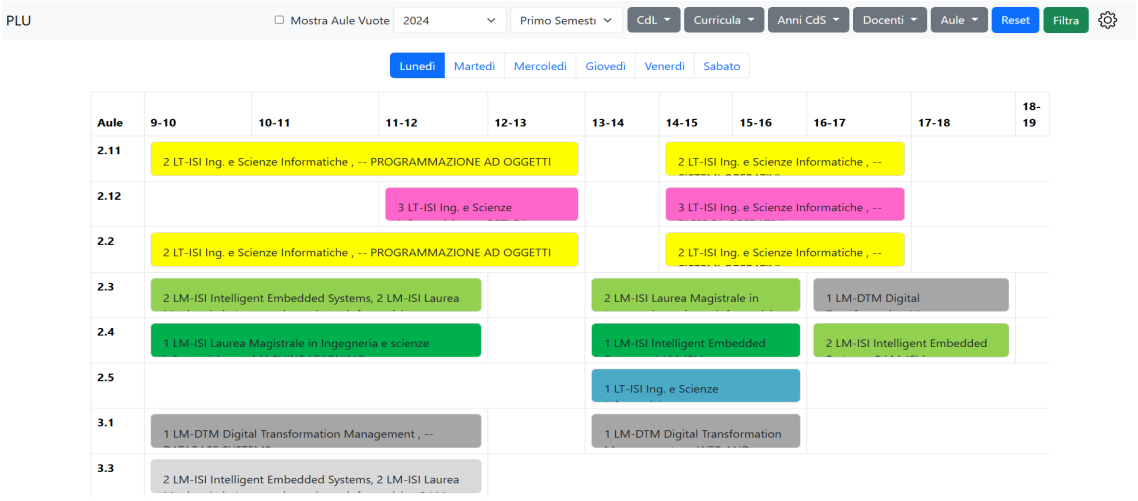

Figura 24: Schermata Home (visualizzazione orario)

La schermata Home dell'applicazione presenta, all'interno della parte superiore della pagina, un insieme di controlli (combo box e select box) che consentono di effettuare filtri di ricerca combinati sui seguenti parametri: anno accademico, semestre, curriculum, corso di laurea (CdL), docenti, aule e anni di corso. Il pulsante "Reset" consente di azzerare tutti i filtri selezionati. Il pulsante "Filtra", invece, invia i dati di ricerca tramite form, aggiornando la visualizzazione delle lezioni filtrate secondo i criteri impostati, all'interno della stessa pagina. I filtri selezionati vengono mantenuti anche dopo la ricerca grazie all'utilizzo di JavaScript e dei parametri in query string, che permettono di conservare lo stato delle selezioni. La griglia contenente i risultati ha un'altezza massima predefinita; al superamento di tale altezza, viene abilitata una scrollbar verticale, mantenendo fissa la prima riga dell'intestazione (sticky header) per facilitarne la consultazione. La checkbox "Aule vuote", se selezionata, include nella visualizzazione anche le aule che non presentano lezioni pianificate. Ogni lezione è rappresentata da una card colorata, il cui colore varia in base alla combinazione tra corso di laurea e curriculum.

Cliccando su una lezione, la card si espande mostrando nel dettaglio tutte le informazioni relative alla lezione stessa. Nella *Figura 25* è illustrato il contenuto di una lezione visualizzato in modalità espansa. Infine, cliccando sul pulsante "Elimina", viene chiesto all'utente di confermare l'operazione prima di procedere con l'eliminazione della lezione. Il pulsante "Sposta", invece, reindirizza l'utente alla schermata di modifica della lezione, mostrata in *Figura 27*.

Figura 25: Schermata dettaglio Lezione

Schermata Impostazioni

Di seguito è riportata, nella *Figura 26*, la schermata di impostazioni dell'applicazione web.

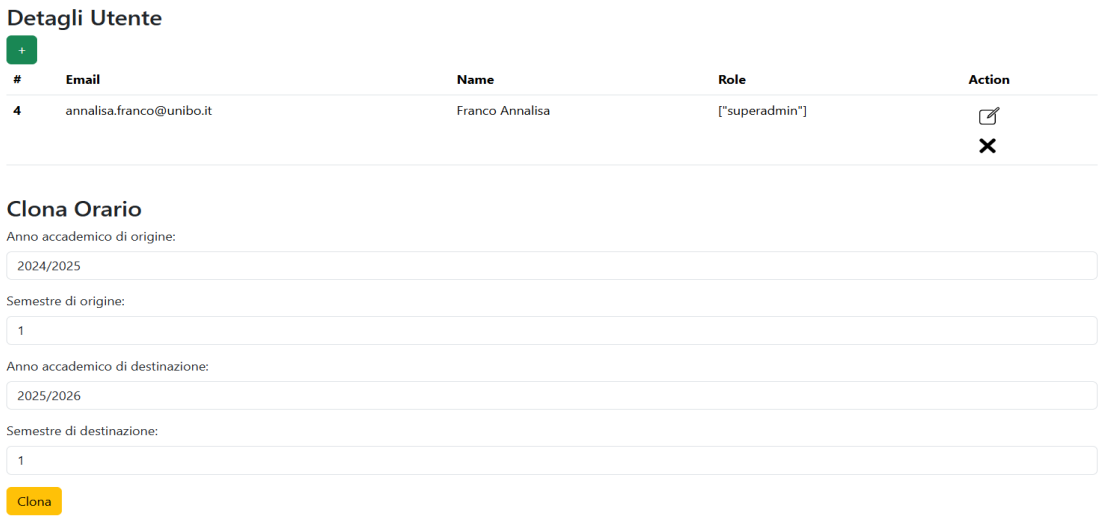

Figura 26: Schermata Impostazioni

La schermata Impostazioni dell'applicazione è composta da due aree principali. La prima riguarda la gestione degli utenti e si comporta in modo diverso a seconda del ruolo dell'utente autenticato. Gli utenti con ruolo super-admin hanno il pieno controllo sull'intero sistema e possono gestire tutti gli account registrati. Gli utenti con ruolo admin o user, invece, possono modificare solamente il proprio profilo, limitandosi a cambiare username e password.

Quando un super-admin desidera aggiungere un nuovo utente, può cliccare sull'icona "+", che apre un modale in cui vengono richiesti alcuni dati: username, email, password e ruolo. La modifica di un utente già esistente avviene selezionando l'apposita icona nella colonna "Azioni"; si apre quindi un altro modale che consente di aggiornare username, password e ruolo dell'utente selezionato. Sempre nella stessa colonna è presente un'icona con una "X" che, se cliccata, consente l'eliminazione dell'utente dopo una conferma tramite finestra di dialogo.

La seconda area della schermata, denominata Clona Orario, è accessibile esclusivamente agli utenti con ruolo admin o super-admin. Questa funzionalità consente di copiare l'orario delle lezioni da un anno accademico e semestre di origine a un altro anno accademico e semestre di destinazione. Per farlo, è sufficiente inserire le informazioni richieste e premere il pulsante "Clona". Se si

verificano errori durante l'operazione, questi vengono mostrati direttamente nella stessa pagina. In caso contrario, l'utente viene automaticamente reindirizzato alla schermata Home.

Schermata Modifica Lezione

Di seguito è riportata, nella Figura 27, la schermata di modifica lezione dell'applicazione web.

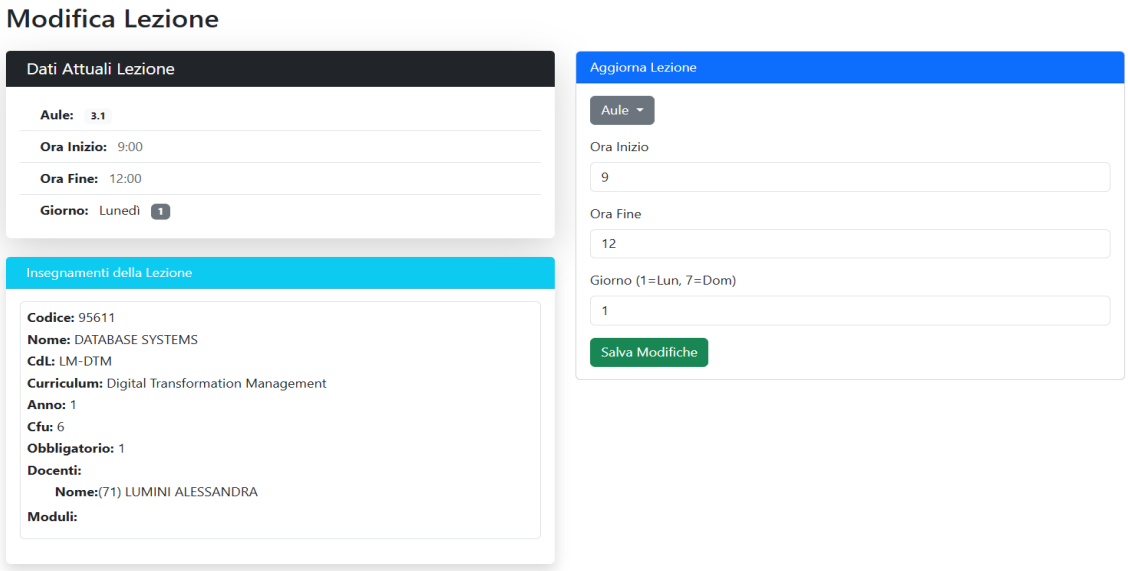

Figura 27: Schermata Modifica Lezione

La schermata di modifica lezione è strutturata in tre sezioni principali. La prima, denominata "Dati Attuali Lezione", mostra le informazioni principali della lezione nello stato attuale, fornendo all'utente una panoramica dei dati già registrati. La seconda sezione, "Insegnamenti della Lezione", elenca tutti gli insegnamenti associati alla lezione, con i relativi docenti e i moduli collegati. Infine, la sezione "Aggiorna Lezione" consente all'utente di modificare alcuni parametri della lezione tramite un apposito form.

Tra i dati aggiornabili vi sono le aule, che possono essere una o al massimo due. Nel caso in cui vengano selezionate due aule, la lezione verrà automaticamente considerata come "sdoppiata", ovvero svolta contemporaneamente in entrambe le aule. Come descritto nella fase 2.2. dell'analisi, questa configurazione è prevista per situazioni in cui è necessario dividere fisicamente gli studenti pur mantenendo la lezione in contemporanea. Oltre alle aule, è possibile modificare anche l'ora di inizio, l'ora di fine e il giorno della settimana in cui si tiene la lezione.

Una volta apportate le modifiche desiderate, l'utente può cliccare sul pulsante "Salva modifiche". A questo punto, il sistema esegue una serie di controlli sui vincoli definiti, applicando la logica implementata attraverso il pattern Chain of Responsibility, descritto nel capitolo 4.2.3. Se uno o più vincoli non risultano soddisfatti, l'operazione viene interrotta e nella stessa pagina viene mostrato un messaggio di errore con tutti i dettagli utili alla comprensione del problema. Al contrario, se tutti i vincoli vengono rispettati, la modifica viene salvata correttamente e l'utente viene automaticamente reindirizzato alla pagina precedente.

4.4 Deployment

La fase di deploy di un'applicazione web consiste nell'insieme delle operazioni necessarie per rendere l'applicazione accessibile e funzionante in un ambiente di produzione o di test. Questa fase include la configurazione del server, l'installazione delle dipendenze, la gestione del database e l'avvio dei servizi necessari affinché l'applicazione sia raggiungibile dagli utenti finali: un deploy efficace garantisce stabilità, sicurezza e scalabilità dell'applicazione, facilitando eventuali aggiornamenti futuri.

Docker è una piattaforma che permette di creare, distribuire e gestire applicazioni all'interno di container isolati, ovvero ambienti leggeri e portabili che includono tutto il necessario per eseguire il software, indipendentemente dal sistema operativo sottostante. Il grande vantaggio offerto da Docker risiede nella sua capacità di assicurare coerenza tra ambienti di sviluppo, test e produzione, eliminando i classici problemi di incompatibilità. Inoltre, grazie ai container, è possibile orchestrare facilmente più componenti di un'applicazione in modo modulare, semplificando la manutenzione e la scalabilità.

Nel caso specifico della web app per la programmazione delle lezioni dell'Università di Bologna, l'architettura è articolata in due container distinti. Il primo container ospita l'applicazione Laravel ed è basato su Apache, che funge da server HTTP e gestisce sia le richieste web che l'esecuzione del codice PHP tramite mod_php. Il secondo container esegue MySQL, il database relazionale a cui l'applicazione Laravel si connette per memorizzare e recuperare i dati relativi alle lezioni, ai corsi di studio e agli utenti. Questa separazione in container consente di isolare i diversi ruoli, facilitando la gestione, il deploy e l'eventuale scalabilità dei singoli componenti.

La scelta di utilizzare Apache è dettata dalla sua diffusione, stabilità e dalla facilità d'integrazione con PHP tramite moduli dedicati. Questa configurazione permette inoltre di eseguire i comandi di Laravel Artisan direttamente all'interno del container dell'applicazione, semplificando così le operazioni di manutenzione e sviluppo. Di seguito è riportato il contenuto del file docker-compose.yml utilizzato per il deployment dell'applicazione.

```yaml
services:
  app:
    build:
     context: ./docker/apache/
    dockerfile: Dockerfile
    container_name: pol_app
    working_dir: /var/www/html
    volumes:
      - ./:/var/www/html
    ports:
      - "8000:80"
    depends_on:
      - mysql
    networks:
      - laravel

   mysql:
    image: mysql:8.0
    container_name: pol_db
    restart: unless-stopped
    environment:
      MYSQL_DATABASE: laravel
      MYSQL_ROOT_PASSWORD: root
      MYSQL_USER: laravel
      MYSQL_PASSWORD: laravel
    volumes:
      - dbdata:/var/lib/mysql
      - ./docker/dbdump/unibo_v12.sql:/docker-entrypoint-initdb.d/dump.sql:ro
    expose:
      - 3306
    networks:
      - laravel

volumes:
  dbdata:

networks:
  laravel:
    driver: bridge
```

Capitolo 5

5 Conclusioni

Il lavoro svolto in questa tesi ha portato alla progettazione e allo sviluppo di una Web App pensata per semplificare e ottimizzare la gestione e la programmazione delle lezioni dell'università di Bologna. A partire dall'analisi dei requisiti funzionali e non funzionali, è stato possibile definire un'architettura scalabile e modulare basata su tecnologie web moderne che consentono agli utenti di operare in maniera intuitiva e fluida.

Durante lo sviluppo sono state affrontate e superate diverse sfide tecniche, in particolare le due più importanti sono state la realizzazione della base di dati in modo che fosse più generale possibile e la gestione dei conflitti tra lezioni sovrapposte, compresa la verifica dei vincoli. L'adozione di strumenti come Laravel e Docker ha permesso di garantire buone prestazioni, manutenibilità del codice e facilità di distribuzione.

Il progetto costituisce una solida base per l'evoluzione verso un'applicazione completa, grazie a un'architettura pensata per essere estendibile e facilmente manutenibile. Tra i possibili sviluppi futuri si possono considerare:

- Integrazione con sistemi di autenticazione avanzati, come il protocollo Kerberos, per consentire l'accesso tramite le credenziali locali del dominio UNIBO.
- Generazione automatica dell'orario delle lezioni tramite un algoritmo di ottimizzazione, in grado di bilanciare vincoli logistici, didattici e di disponibilità.
- Possibilità di effettuare operazioni CRUD (Creazione, Lettura, Aggiornamento, Eliminazione) sui curricula dei Corsi di Laurea, con la definizione dettagliata di insegnamenti e moduli.
- Integrazione con altri sistemi UNIBO, ad esempio tramite API che permettono di sincronizzare le informazioni con i pannelli di visualizzazione degli orari presenti nelle diverse sedi.

In conclusione, questa esperienza ha permesso di consolidare le competenze acquisite nel corso del percorso di studi, sia in ambito teorico che pratico, offrendo una concreta applicazione delle tecniche di progettazione e sviluppo apprese. Inoltre, ha evidenziato l'importanza di una visione sistemica e trasversale nello sviluppo di soluzioni informatiche rivolte a esigenze reali.

Bibliografia

- Dipartimento per la trasformazione digitale (Presidenza del Consiglio dei Ministri). (2022, 12 ottobre). Italia digitale 2026. Innovazione.gov.it. https://innovazione.gov.it/italia-digitale-2026/
- Taylor, O. (2023). Laravel documentation. Laravel. https://laravel.com/docs
- Hidar, B. (2022). Building MVC Applications in PHP Laravel: Part 1, Code Magazine. https://www.codemag.com/Article/2205071/Building-MVC-Applications-in-PHP-Laravel-Part-1
- Docker, Inc. (s.d.). Compose overview. Docker Documentation. https://docs.docker.com/compose/
- PHP Group. (s.d.). PHP Manual. PHP.net. https://www.php.net/docs.php
- Oracle Corporation. (s.d.). MySQL Documentation. MySQL. https://dev.mysql.com/doc/
- Refactoring Guru. (s.d.). Chain of Responsibility. Refactoring Guru. https://refactoring.guru/design-patterns/chain-of-responsibility

Ringraziamenti

Un ringraziamento profondo e sincero va alla mia famiglia, che ha sempre creduto in me, anche quando io stesso ho fatto fatica a farlo. La forza che mi avete trasmesso in ogni momento è stata fondamentale per arrivare fin qui. Questa tesi è anche vostra.

Un grazie speciale va ai miei coinquilini, Dario e Alex. In questo periodo siete stati molto più che semplici compagni di casa: siete stati una seconda famiglia. Grazie per il sostegno, le risate, le notti insonni condivise e per non avermi mai fatto sentire solo, nemmeno nei momenti più difficili.

Un ringraziamento profondo va anche a mio nonno Angelo, che ha saputo trasmettermi, con semplicità e passione, un modo di guardare alle cose con spirito pratico e ingegneristico, partendo dalla campagna e dai lavori manuali.

Infine, desidero ringraziare la prof.ssa Franco e il prof. Ferrara per la fiducia e la pazienza che hanno avuto nel seguirmi durante la realizzazione di questo progetto. Senza il loro supporto, nulla di tutto questo sarebbe stato possibile.

Bianchi Gianluca

01/07/2025