



Politecnico di Torino
Laurea in Ingegneria Gestionale – Classe L8

Allocazione strategica delle aule universitarie

**Una piattaforma web per gestire ed ottimizzare
l'assegnazione secondo le richieste dei docenti**

Candidato:

Greta Colella
s294353

Supervisore:

Prof. Juan Pablo
Saenz Moreno

Anno Accademico 2024/2025

Allocazione strategica delle aule universitarie

Una piattaforma web per gestire ed ottimizzare
l'assegnazione secondo le richieste dei docenti

Indice

1 Proposta di progetto	4
1.1 Titolo della proposta	4
1.2 Descrizione del problema proposto	4
1.3 Rilevanza gestionale del problema	4
1.4 Dataset utilizzati	4
1.5 Algoritmi coinvolti	5
1.6 Funzionalità previste	5
2 Descrizione dettagliata del problema affrontato	6
2.1 Descrizione del problema	6
3 Descrizione del Dataset	8
3.1 Origine dei dati	8
3.2 Relazioni tra le entità	8
3.3 Utilizzo del dataset	9
4 Funzionamento dell'applicazione	11
4.1 Struttura generale	11
4.2 Ruoli e autenticazione	11
4.3 Flusso operativo dell'applicazione	12
4.4 Pagine principali dell'interfaccia	12
5 Algoritmo di assegnazione	13
5.1 Logica generale dell'algoritmo	13
5.2 Implementazione in codice	14
6 Interfaccia utente	18
6.1 Funzionalità per il docente	18
6.1.1 Creare una nuova richiesta	19
6.1.2 Pagina profilo	20
6.2 Funzionalità per l'amministratore	21
7 Risultati e discussione	23
8 Conclusioni	25

Riassunto

Questa tesi presenta lo sviluppo di una piattaforma web progettata per ottimizzare l'assegnazione delle aule universitarie in base alle richieste dei docenti. L'obiettivo principale è massimizzare il numero di richieste soddisfatte, evitando conflitti di sovrapposizione e assegnando l'aula più adatta per capienza e dotazioni. L'applicazione è sviluppata in Python utilizzando il framework Flask, con un database SQLite per la gestione dei dati.

L'algoritmo implementato segue una strategia che include: il calcolo delle alternative disponibili, un ordinamento intelligente delle richieste e una riassegnazione dinamica per risolvere conflitti. Il sistema distingue i ruoli di amministratore e docente: i docenti possono inserire e visualizzare le proprie richieste, mentre l'amministratore esegue l'algoritmo di assegnazione e visualizza le richieste di tutti i docenti.

I risultati mostrano la capacità del sistema di gestire numerose richieste in modo efficiente, proponendo una soluzione utile per la pianificazione didattica nelle strutture accademiche.

Capitolo 1

Proposta di progetto

1.1 Titolo della proposta

Allocazione strategica delle aule universitarie: una piattaforma web per gestire ed ottimizzare l’assegnazione secondo le richieste dei docenti.

1.2 Descrizione del problema proposto

L’idea dell’applicazione nasce dall’esigenza di una gestione più efficiente dell’allocazione delle aule per i corsi dei singoli semestri, che soddisfi il più possibile le richieste dei docenti. Ogni richiesta contiene: capienza minima, fasce orarie distribuite sui giorni della settimana, dotazioni aggiuntive dell’aula (prese, proiettore, pc). La complessità del problema risiede nel garantire un ottimale utilizzo delle risorse, evitando sovrapposizioni.

1.3 Rilevanza gestionale del problema

Nelle grandi università, come il Politecnico di Torino, l’allocazione delle aule rappresenta una criticità nella gestione operativa. Uno strumento automatizzato ed intelligente permette di:

- accelerare il processo di assegnazione;
- garantire la soluzione più adatta alle esigenze;
- migliorare la soddisfazione dei docenti;
- aumentare l’utilizzo efficiente degli spazi;

1.4 Dataset utilizzati

Il sistema si basa su un database SQLite che memorizza:

- le caratteristiche delle aule (capienza, dotazioni);
- gli slot orari disponibili (7 per giorno);

- le richieste dei docenti (capienza, giorno, slot, dotazioni);
- le assegnazioni effettuate.

1.5 Algoritmi coinvolti

L'assegnazione delle aule è governata da un algoritmo avanzato che segue questa logica a livelli:

1. Calcolo delle alternative possibili per ogni richiesta.
2. Ordinamento delle richieste in base alla criticità (minore numero di alternative).
3. Assegnazione diretta o, in caso di conflitto, tentativo di riassegnazione dinamica.
4. Tracciamento di avvenuta assegnazione o fallimento.

1.6 Funzionalità previste

L'applicazione offre le seguenti funzionalità principali:

- Login per docenti e amministratori.
- Inserimento richieste tramite interfaccia web intuitiva.
- Visualizzazione dello stato delle richieste (myaccount).
- Esecuzione dell'algoritmo di assegnazione (riservato all'admin).
- Consultazione degli esiti delle assegnazioni.

Capitolo 2

Descrizione dettagliata del problema affrontato

2.1 Descrizione del problema

L'allocazione efficiente degli spazi universitari rappresenta una componente fondamentale per una gestione organizzativa adeguata alle dimensioni e alla complessità di un grande ateneo. Prendendo come esempio il Politecnico di Torino, secondo i dati ufficiali relativi all'anno accademico 2024/2025, l'università ospita circa 38.800 studenti, così distribuiti:

- 21.300 iscritti alla laurea triennale
- 15.400 iscritti alla laurea magistrale
- 630 frequentanti Master di I e II livello e corsi di formazione permanente
- 1.450 dottorandi

Sempre per l'anno accademico 2024/2025, l'offerta formativa comprende:

- 25 corsi di Laurea di I livello (4 in Architettura/Design/Pianificazione e 21 in Ingegneria)
- 37 corsi di Laurea Magistrale di II livello (9 in Architettura/Design/Pianificazione e 28 in Ingegneria)
- 32 percorsi completamente in inglese
- 32 Master di I e II livello e corsi di formazione permanente
- 18 corsi di Dottorato di ricerca (di cui 5 in convenzione)

Di fronte a una tale ampiezza e varietà di corsi, l'organizzazione manuale o semi-automatica degli spazi risulta spesso inefficiente. In particolare, nonostante la disponibilità di numerose aule, è difficile trovare una soluzione ottimale che soddisfi quante più esigenze possibili.

Il problema è aggravato da diversi fattori:

- l'elevato numero di richieste ricevute ogni semestre;
- la scarsità relativa di aule con caratteristiche specifiche;
- l'eterogeneità delle esigenze didattiche;
- la necessità di equità e trasparenza nel processo decisionale.

È qui che entra in gioco la piattaforma presentata: si chiede ai professori di formulare richieste specificando le caratteristiche desiderate dell'aula (come prese di corrente, PC o proiettore), in modo da elaborarle in maniera intelligente e offrire agli studenti ambienti adeguati al tipo di didattica prevista dal corso.

Ad esempio, in un corso di Informatica gli studenti potrebbero aver bisogno di prese elettriche per i propri dispositivi; pertanto, un docente richiederà aule con questa dotazione.

Il sistema proposto affronta queste criticità tramite un algoritmo avanzato di assegnazione e riassegnazione dinamica, che consente di massimizzare il numero di richieste soddisfatte, evitando per quanto possibile sprechi di risorse (come l'utilizzo di aule troppo grandi per corsi con pochi studenti) e sovrapposizioni di slot.

Il progetto rappresenta quindi una soluzione concreta e scalabile per istituzioni accademiche di grandi dimensioni che desiderano digitalizzare e ottimizzare uno degli aspetti più delicati della pianificazione didattica.

Capitolo 3

Descrizione del Dataset

3.1 Origine dei dati

Il dataset è stato definito manualmente per modellare un tipico scenario universitario. Le aule sono differenziate per id, capienza e caratteristiche, mentre gli slot orari riflettono una giornata accademica standard (dalle 8:30 alle 19, 7 slot da 1h e 30min). Le richieste sono simulate sulla base di situazioni reali, con una varietà di capienze e preferenze distribuite su più giorni. Idealmente si potrebbe inserire ed adattare un database di aule reali di un ateneo, questo è stato creato solamente per lo scopo funzionale dell'applicazione.

3.2 Relazioni tra le entità

Il database `assegnaAule.db` è strutturato in modo relazionale. Ogni richiesta è collegata a un utente (docente) e potenzialmente a una assegnazione. Le assegnazioni collegano direttamente una richiesta a una specifica aula.

- **users → richiesta**

Relazione: 1 a N (uno a molti). Ogni utente (professore) può effettuare più richieste, ma ogni richiesta è associata a un solo professore. Chiave esterna: `richiesta.idProf` → `users.id`.

- **richiesta → assegnazione**

Relazione: 1 a 1 (uno a uno). Una richiesta può avere al massimo una sola assegnazione di aula. Se non viene assegnata, non compare nella tabella `assegnazione`. Chiave esterna: `assegnazione.id_richiesta` → `richiesta.id`.

- **aula → assegnazione**

Relazione: 1 a N (uno a molti). Una stessa aula può essere assegnata a più richieste (ma in slot e giorni diversi), mentre ogni assegnazione riguarda una sola aula. Chiave esterna: `assegnazione.id_aula` → `aula.id`.

- **slot (non relazionato direttamente)**

Gli slot non sono collegati direttamente con chiavi esterne, ma vengono riferenziati all'interno del campo `slots` della tabella `richiesta` come stringa di ID separati da virgola (es. "2,3"). Questo consente a ogni richiesta di occupare più fasce orarie in uno stesso giorno.

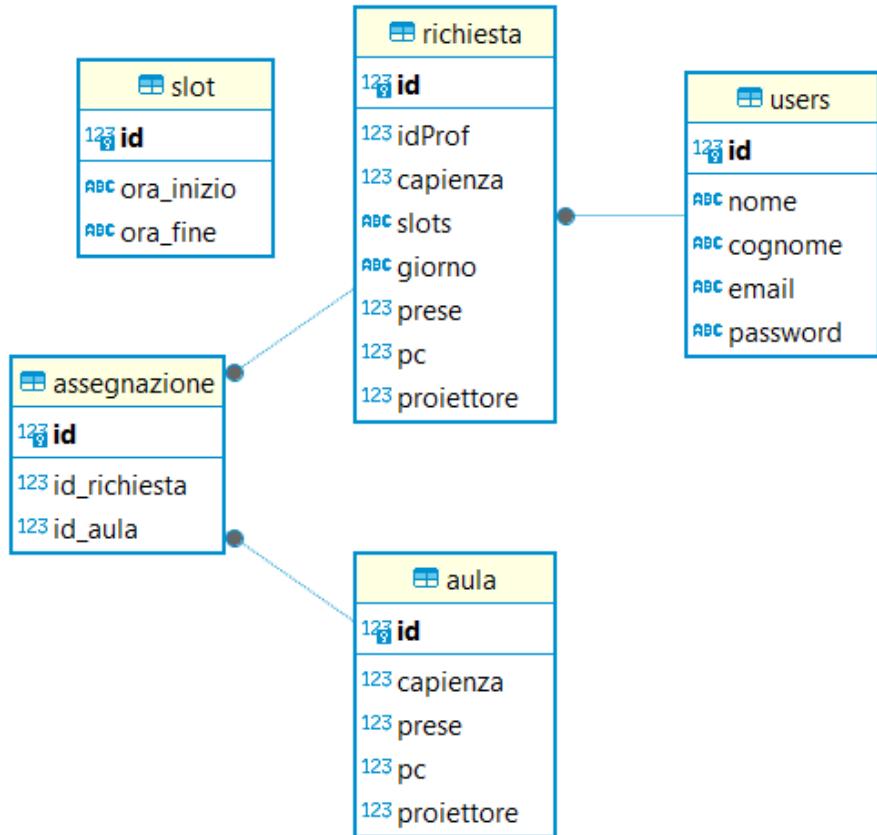


Figura 3.1: Diagramma delle relazioni tra entità del database assegnaAule.db

3.3 Utilizzo del dataset

il dataset viene utilizzato per:

- Verificare la disponibilità di aule compatibili: per ogni richiesta inserita da un docente, il sistema confronta le caratteristiche richieste (capienza, dotazioni) con quelle delle aule disponibili nel database, e verifica che non vi siano sovrapposizioni negli slot orari e nei giorni specificati.
- Identificare eventuali sovrapposizioni: tramite il controllo incrociato tra slot richiesti e slot già occupati da altre assegnazioni registrate nel dataset, il sistema evita conflitti e garantisce che ogni aula sia utilizzata da un solo docente per slot e giorno.
- Ottimizzare l'uso delle risorse: l'algoritmo seleziona l'aula più adatta con la minima capienza sufficiente, evitando lo spreco di spazi e favorendo un uso razionale delle strutture esistenti.
- Registrare gli esiti delle assegnazioni: una volta eseguita l'assegnazione, il database viene aggiornato salvando le informazioni nella tabella **assegnazione**, rendendo visibili le soluzioni calcolate. Questi dati sono poi consultabili dai docenti tramite l'interfaccia del profilo personale.

- Supportare modifiche dinamiche e futuri ricalcoli: poiché ogni elemento è tracciato in modo strutturato e relazionale, è possibile aggiornare o ricalcolare le assegnazioni in caso di nuove richieste, modifiche o cambiamenti imprevisti.

Capitolo 4

Funzionamento dell'applicazione

L'applicazione web è sviluppata con il micro-framework Flask in linguaggio Python. Essa si compone di diverse funzionalità accessibili tramite un'interfaccia semplice e responsiva, strutturata per adattarsi ai due principali ruoli utente: **docente** e **amministratore**.

4.1 Struttura generale

L'applicazione è organizzata in moduli funzionali che separano le responsabilità principali:

- **Cartella templates:** contiene i file HTML utilizzati per costruire le pagine dell'interfaccia utente. Ogni pagina è costruita con Jinja2, il motore di template predefinito di Flask, e si appoggia a Bootstrap per la formattazione.
- **Cartella static:** ospita i file CSS personalizzati e altri contenuti statici come immagini. Il file principale di stile `style.css` regola l'aspetto grafico dell'interfaccia.
- **File Python:** suddivisi tra moduli specifici:
 - I file DAO (Data Access Object), come `richieste_dao.py`, `aule_dao.py`, `slot_dao.py`, `users_dao.py`, `assegnazione_dao.py`, definiscono le query SQL per ciascuna entità e gestiscono l'accesso al database.
 - Il file `assegnazione_avanzata.py` contiene l'algoritmo che elabora e assegna le aule in base alle richieste.
 - Il file principale `app.py` gestisce la creazione dell'applicazione Flask, le route e collega l'interfaccia utente con la logica e il database.

4.2 Ruoli e autenticazione

- I **docenti** possono registrarsi, effettuare il login, inserire nuove richieste e visualizzare l'esito delle assegnazioni.
- L'**amministratore** ha accesso esclusivo alla funzione di assegnazione aule e alla visualizzazione completa di tutte le richieste.

L'autenticazione è gestita tramite il pacchetto **flask-login**, con gestione delle credenziali cifrate.

4.3 Flusso operativo dell'applicazione

1. L'utente accede all'applicazione (login o signup).
2. Il docente accede alla pagina di richiesta aula e seleziona giorno, slot, capienza e dotazioni.
3. Ogni richiesta viene salvata nel database in modo strutturato.
4. Quando l'amministratore avvia il processo di assegnazione, viene eseguito l'algoritmo avanzato (capitolo successivo).
5. Gli esiti vengono salvati nel database e resi disponibili per la consultazione.

4.4 Pagine principali dell'interfaccia

- **Home:** schermata iniziale con accesso e registrazione.
- **Richiesta:** modulo di inserimento richieste, visibile ai docenti loggati.
- **Profilo (myaccount):** panoramica delle richieste del docente con assegnazioni.
- **Assegna aule:** funzione disponibile solo per l'admin, per avviare il processo.
- **Visualizza assegnazioni:** riepilogo di tutte le assegnazioni effettuate.

Capitolo 5

Algoritmo di assegnazione

L'algoritmo di assegnazione è il cuore funzionale dell'applicazione. Il suo scopo è quello di massimizzare il numero di richieste soddisfatte, assegnando ad ognuna l'aula più adatta. Viene chiamato dall'amministratore, inizia raccogliendo tutte le richieste memorizzate fino a quel momento. L'idea è di farlo partire dopo un'ipotetica data di scadenza per l'invio delle richieste.

5.1 Logica generale dell'algoritmo

L'algoritmo è diviso in più fasi:

1. Costruzione dell'insieme di alternative valide:

Per ogni richiesta si individuano le aule che soddisfano i vincoli di capienza e disponibilità negli slot.

2. Ordinamento delle richieste in base alla criticità:

Le richieste vengono ordinate a partire da quelle con meno alternative disponibili. L'obiettivo è assegnare prima le richieste più difficili da soddisfare, riducendo il rischio di esclusione.

3. Ordinamento delle aule compatibili:

Le aule possibili per ciascuna richiesta vengono ordinate in modo da privilegiare quelle con capienza più vicina a quella richiesta.

4. Assegnazione basata sulle dotazioni:

Si assegna l'aula dove il match con le dotazioni richieste è massimo. Se nessuna aula soddisfa tutte le dotazioni, viene comunque assegnata l'opzione migliore possibile.

5. Risoluzione dei conflitti:

In caso di sovrapposizioni tra richieste con slot coincidenti, l'algoritmo verifica se è possibile “spostare” le richieste già assegnate verso aule alternative, allo scopo di liberare una soluzione per la richiesta corrente. Questo riassegnamento dinamico consente di aumentare la copertura totale.

6. Tracciamento degli esiti:

Per ogni richiesta viene registrato un esito:

- aula assegnata direttamente;
- aula assegnata dopo riassegnazione dinamica;
- fallimento (“Impossibile assegnare: capienza insufficiente o slot occupati”).

Efficienza e vantaggi

Questo algoritmo, pur non essendo basato su una tecnica di ottimizzazione formale (come programmazione lineare o backtracking completo), rappresenta un buon compromesso tra efficienza computazionale e qualità della soluzione. È sufficientemente veloce anche con decine di richieste e aule, e assicura un'elevata percentuale di soddisfazione.

Inoltre, il sistema è progettato per essere estensibile: l'algoritmo può essere ulteriormente potenziato, ad esempio, con meccanismi di priorità per determinati docenti o corsi, o con strategie predittive basate su dati storici.

5.2 Implementazione in codice

Di seguito è riportato il codice Python dell'algoritmo avanzato di assegnazione:

```
import aule_dao

giorni = ["Lunedì", "Martedì", "Mercoledì", "Giovedì", "Venerdì"]

def assegna_aule_avanzato(richieste, aule, slots):
    disponibilità = {}
    for aula in aule:
        for giorno in giorni:
            for slot in range(1, 8):
                disponibilità[(aula.id_aula, giorno, slot)] = True

    alternative_per_richiesta = {}
    for richiesta in richieste:
        alternative = 0
        for aula in aule:
            if aula.capienza >= richiesta.capienza_richiesta:
                if all(disponibilità[(aula.id_aula, richiesta.giorno, id_slot)] for id_slot in richiesta.slotIds):
                    alternative += 1
        alternative_per_richiesta[richiesta.id] = alternative

    richieste_ordinate = sorted(richieste, key=lambda r: (alternative_per_richiesta[r.id], r.capienza_richiesta))

    assegnazioni = {}
    motivazioni = {}

    for richiesta in richieste_ordinate:
        aula_trovata = None
        max_match = -1
        aule_ordinate = sorted(aule, key=lambda a: a.capienza)
```

```

        for aula in aule_ordinate:
            if aula.capienza >= richiesta.capienza_richiesta:
                if all(disponibilita[(aula.id_aula, richiesta.
                    giorno, id_slot)] for id_slot in richiesta.
                    slotIds):
                    aula_db = aule_dao.get_aula_by_id(aula.id_aula)
                    if not aula_db:
                        continue
                    match = (
                        (richiesta.prese and aula_db['prese']) +
                        (richiesta.pc and aula_db['pc']) +
                        (richiesta.proiettore and aula_db['
                            proiettore']))
                )
                if match > max_match:
                    aula_trovata = aula
                    max_match = match

        if aula_trovata:
            assegnazioni[richiesta.id] = aula_trovata.id_aula
            motivazioni[richiesta.id] = "Assegnazione diretta"
            for id_slot in richiesta.slotIds:
                disponibilita[(aula_trovata.id_aula, richiesta.
                    giorno, id_slot)] = False
        else:
            if tenta_riassegnazione(richiesta, assegnazioni,
                disponibilita, richieste, aule):
                motivazioni[richiesta.id] = "Assegnata dopo
                    riasssegnazione dinamica"
            else:
                assegnazioni[richiesta.id] = None
                motivazioni[richiesta.id] = "Impossibile assegnare"

        return assegnazioni, motivazioni

def tenta_riassegnazione(richiesta_corrente, assegnazioni,
    disponibilita, richieste, aule):
    aule_candidate = [aula for aula in aule if aula.capienza >=
        richiesta_corrente.capienza_richiesta]
    for aula in aule_candidate:
        slot_blocanti = [id_slot for id_slot in richiesta_corrente
            .slotIds
                if not disponibilita[(aula.id_aula,
                    richiesta_corrente.giorno, id_slot)]]
    if not slot_blocanti:
        continue

    richieste_blocanti = []
    for id_slot in slot_blocanti:
        for r in richieste:
            if (assegnazioni.get(r.id) == aula.id_aula and r.
                giorno == richiesta_corrente.giorno
                and id_slot in r.slotIds):
                richieste_blocanti.append(r)

    for richiesta_blocante in richieste_blocanti:
        nuova_aula = trova_nuova_aula(richiesta_blocante,

```

```

        disponibilita, aule, preferenze=(
            richiesta_bloccante.prese, richiesta_bloccante.pc,
            richiesta_bloccante.proiettore))
    if nuova_aula:
        vecchia_aula = assegnazioni[richiesta_bloccante.id]
        assegnazioni[richiesta_bloccante.id] = nuova_aula.
            id_aula
        for id_slot in richiesta_bloccante.slotIds:
            disponibilita[(vecchia_aula,
                richiesta_bloccante.giorno, id_slot)] = True
            disponibilita[(nuova_aula.id_aula,
                richiesta_bloccante.giorno, id_slot)] =
                False

        assegnazioni[richiesta_corrente.id] = aula.id_aula
        for id_slot in richiesta_corrente.slotIds:
            disponibilita[(aula.id_aula, richiesta_corrente
                .giorno, id_slot)] = False
    return True
return False

def trova_nuova_aula(richiesta, disponibilita, aule, preferenze=
None):
    prese_pref, pc_pref, proiettore_pref = preferenze if preferenze
    else (0, 0, 0)
    aule_ordinate = sorted(aule, key=lambda a: a.capienza)
    best_aula = None
    best_match = -1

    for aula in aule_ordinate:
        if aula.capienza >= richiesta.capienza_richiesta:
            if all(disponibilita[(aula.id_aula, richiesta.giorno,
                id_slot)] for id_slot in richiesta.slotIds):
                match = (
                    (prese_pref and aula.prese) +
                    (pc_pref and aula.pc) +
                    (proiettore_pref and aula.proiettore)
                )
                if match > best_match:
                    best_match = match
                    best_aula = aula
    return best_aula

```

Tabella delle variabili e funzioni principali

Nome	Tipo / Funzione	Descrizione
<code>richieste</code>	Lista	Oggetti che rappresentano le richieste dei docenti: includono capienza, giorno, slot e dotazioni.
<code>aula</code>	Lista	Oggetti con attributi come <code>id_aula</code> , <code>capienza</code> , <code>prese</code> , <code>pc</code> , <code>proiettore</code> .
<code>slots</code>	Lista	Rappresenta gli slot disponibili, da 1 a 7 per ogni giorno.
<code>disponibilita</code>	Dizionario	Mappa ogni tupla (aula, giorno, slot) a <code>True/False</code> per indicare se è libera.
<code>alternative_per_richiesta</code>	Dizionario	Conta quante aule compatibili ci sono per ciascuna richiesta. Serve per ordinare le richieste.
<code>assegnazioni</code>	Dizionario	Mappa <code>id</code> della richiesta → <code>id</code> dell'aula assegnata (o <code>None</code>).
<code>motivazioni</code>	Dizionario	Mappa <code>id</code> richiesta → motivazione dell'assegnazione o del fallimento.
<code>tenta_riassegnazione()</code>	Funzione	Cerca di liberare un'aula bloccata riassegnando la richiesta esistente altrove.
<code>trova_nuova_aula()</code>	Funzione	Cerca l'aula migliore per una richiesta data, tenendo conto di capienza e dotazioni.
<code>slot_bloccanti</code>	Lista	Slot richiesti dalla richiesta corrente ma già occupati in quell'aula.
<code>richieste_bloccanti</code>	Lista	Richieste che occupano gli <code>slot_bloccanti</code> e impediscono l'assegnazione corrente.

Tabella 5.1: Variabili e funzioni principali nell'algoritmo di assegnazione

Capitolo 6

Interfaccia utente

L’interfaccia utente dell’applicazione è progettata per essere intuitiva, responsiva e facilmente accessibile sia per i docenti sia per l’amministratore. Ogni tipo di utente ha a disposizione funzionalità mirate, accessibili tramite un menu di navigazione semplice e coerente, realizzato con HTML, Bootstrap e Jinja2.

Accesso alla piattaforma

All’apertura del sito, l’utente è accolto da una pagina di login/registrazione. I docenti possono registrarsi autonomamente tramite un modulo con nome, cognome, email e password. L’amministratore ha un account predefinito non registrabile via interfaccia.

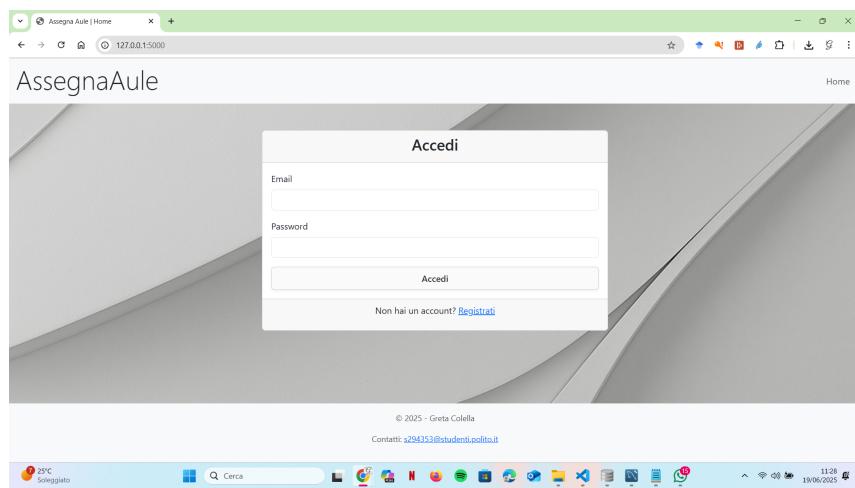


Figura 6.1: Homepage iniziale con pulsanti di login e registrazione

6.1 Funzionalità per il docente

Una volta autenticato, il docente accede alla schermata home:

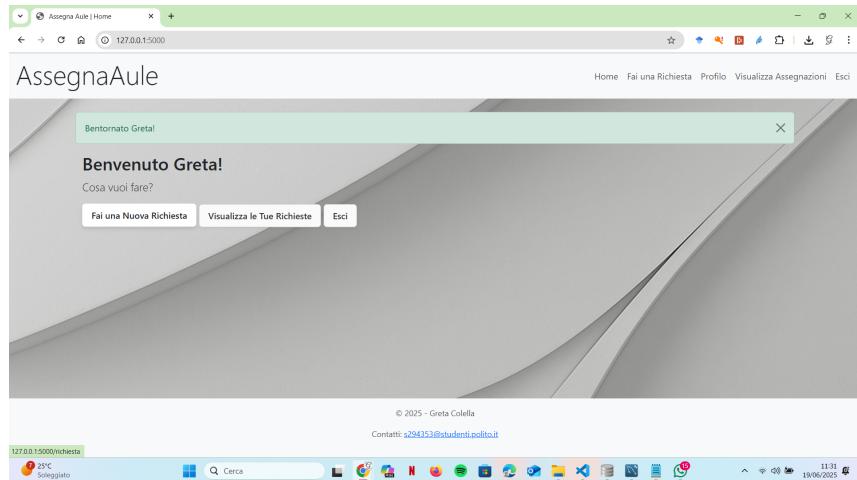


Figura 6.2: Form di inserimento richiesta aula

Qui trova una barra di navigazione, oltre ai button presenti nella pagina, da cui può reindirizzarsi nelle altre pagine per:

- Creare una richiesta di aula
- Visualizzare la sua pagina “Profilo”
- Visualizzare tutte le assegnazioni avvenute

6.1.1 Creare una nuova richiesta

- capienza minima necessaria;
- giorno della settimana;
- uno o più slot orari;
- dotazioni richieste (prese, PC, proiettore).

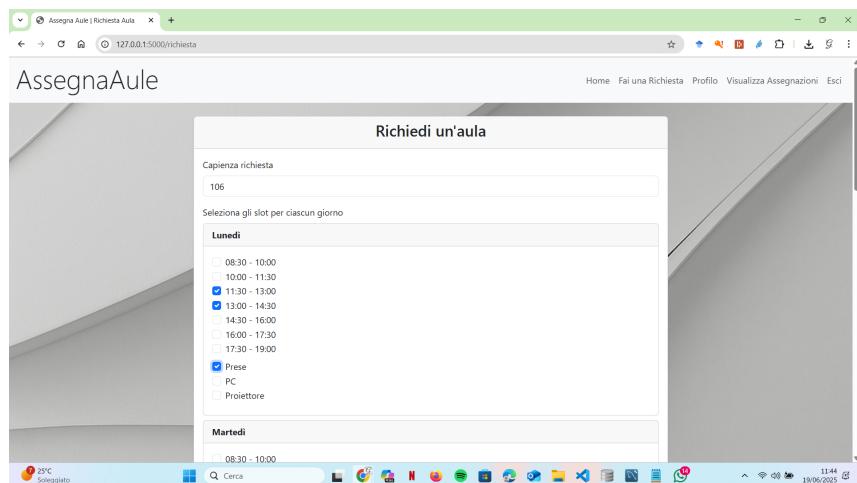


Figura 6.3: Form di inserimento richiesta aula

Quando si schiaccia il pulsante di invio, nel database viene registrata una richiesta diversa per ogni giorno della settimana dove è stato selezionato almeno uno slot.

6.1.2 Pagina profilo

In questa sezione il docente visualizza i propri dati, e tutte le richieste effettuate. Qualora l'assegnazione per tali richieste è avvenuta, visualizzerà l'id dell'aula e le sue dotazioni.

- ID richiesta;
- capienza;
- slot e giorno;
- aula assegnata (se presente);
- stato dell'assegnazione.

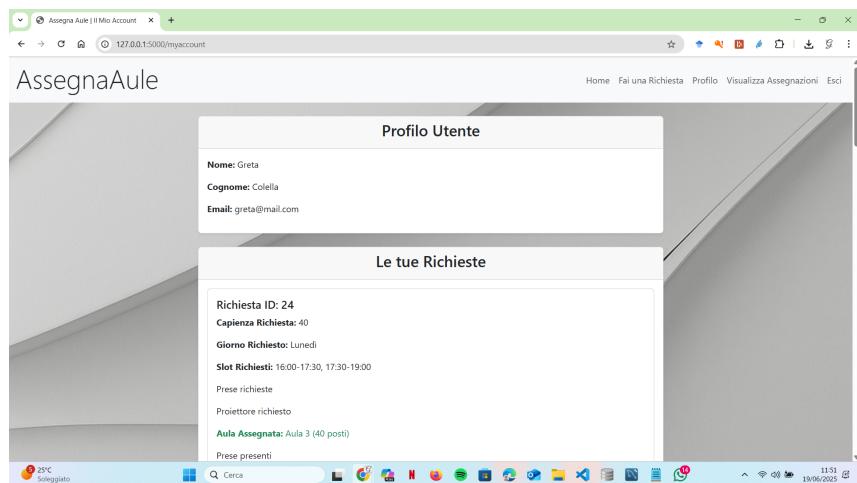


Figura 6.4: Pagina del profilo docente con le richieste visualizzate

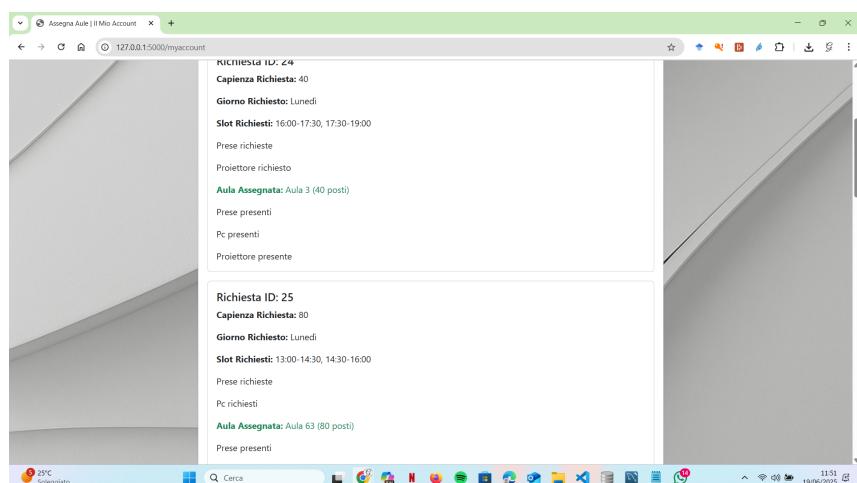


Figura 6.5: Pagina del profilo docente con le richieste visualizzate

6.2 Funzionalità per l'amministratore

L'amministratore, accedendo con il suo account, ha una visualizzazione diversa con privilegi esclusivi:

- Pulsante “Assegna Aule”: disponibile solo per l'admin, permette di avviare l'algoritmo avanzato. Il sistema processa tutte le richieste salvate e aggiorna il database con le assegnazioni trovate.
- Visualizzazione riepilogo: dopo l'esecuzione, il sistema mostra il risultato per ogni richiesta: aula assegnata oppure motivo del fallimento.
- L'admin può consultare tutte le richieste ricevute.

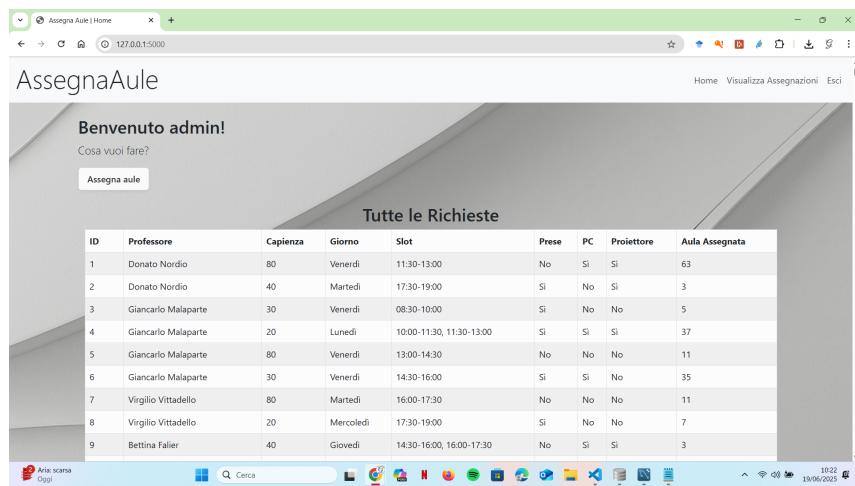


Figura 6.6: Pagina dell'amministratore con tutte le richieste e il pulsante per avviare l'assegnazione

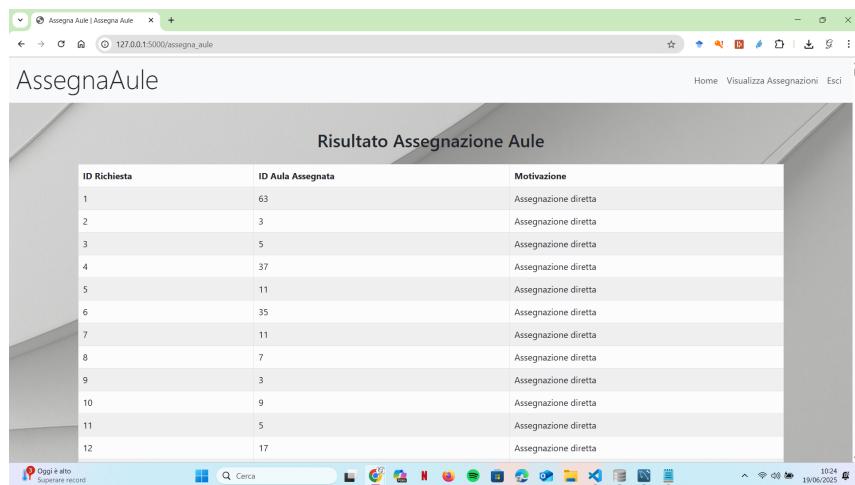
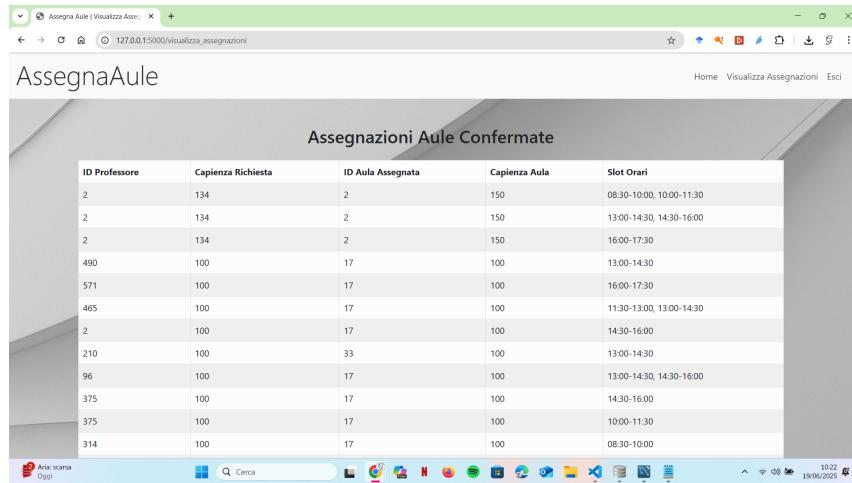


Figura 6.7: Vissualizzazione al termine dell'assegnazione

Visualizzazione delle aule assegnate

Sia i professori che l'amministratore hanno in comune questa pagina, in cui si mostrano tutte le assegnazioni avvenute:



The screenshot shows a web application window titled "AssegnaAule | Visualizza Assegnazioni". The main content area is titled "Assegnazioni Aule Confermate" and displays a table of assigned classrooms. The table has columns: ID Professore, Capienza Richiesta, ID Aula Assegnata, Capienza Aula, and Slot Orari. The data in the table is as follows:

ID Professore	Capienza Richiesta	ID Aula Assegnata	Capienza Aula	Slot Orari
2	134	2	150	08:30-10:00, 10:00-11:30
2	134	2	150	13:00-14:30, 14:30-16:00
2	134	2	150	16:00-17:30
490	100	17	100	13:00-14:30
571	100	17	100	16:00-17:30
465	100	17	100	11:30-13:00, 13:00-14:30
2	100	17	100	14:30-16:00
210	100	33	100	13:00-14:30
96	100	17	100	13:00-14:30, 14:30-16:00
375	100	17	100	14:30-16:00
375	100	17	100	10:00-11:30
314	100	17	100	08:30-10:00

Figura 6.8: Visualizzazione di tutte le assegnazioni riuscite

Aspetto grafico e usabilità

Tutte le pagine sono costruite con Bootstrap per garantire:

- layout responsive su dispositivi mobili e desktop;
- form puliti e ben allineati;
- pulsanti coerenti e accessibili;
- messaggi di conferma o errore per ogni azione effettuata.

Il foglio di stile personalizzato (`style.css`) è utilizzato per mantenere una coerenza visiva generale, senza appesantire il codice HTML.

Capitolo 7

Risultati e discussione

Per valutare la correttezza e l'efficacia dell'algoritmo di assegnazione delle aule, sono stati considerati tre criteri principali:

- **Correttezza:** ogni richiesta viene assegnata a un'aula che soddisfa i vincoli di capienza e dotazioni, evitando sovrapposizioni.
- **Efficienza:** l'algoritmo massimizza il numero di richieste soddisfatte.
- **Riassegnazione dinamica:** il sistema è in grado di riorganizzare assegnazioni precedenti per favorire richieste più critiche.

Setup sperimentale

Per valutare le prestazioni, sono stati generati i seguenti dati:

- **600 professori** con profili realistici;
- ~ 1300 **richieste** di aule, distribuite casualmente in tutti i giorni e fasce orarie;
- **75 aule** con capienze variabili (20–200 posti) e caratteristiche eterogenee (prese, PC, proiettore);
- **7 slot orari** per ciascun giorno lavorativo, per un totale di 35 combinazioni settimana \times aula.

Sono stati condotti test progressivi, con **carichi crescenti** di richieste simili tra loro, fino a raggiungere situazioni di **saturazione completa degli slot** in determinate fasce orarie.

Risultati principali

- Il sistema ha correttamente assegnato la quasi totalità delle richieste, con una percentuale di successo iniziale superiore al **95%**.
- La **verifica SQL** ha confermato l'assenza di **sovraposizioni di slot** su una stessa aula, segno del corretto funzionamento delle logiche di disponibilità.

Listing 7.1: Verifica assenza sovrapposizioni

```
SELECT a1.id_aula, r1.giorno, s1.id AS slot_id, COUNT
      (*) AS occupazioni
  FROM assegnazione AS a1
  JOIN richiesta AS r1 ON a1.id_richiesta = r1.id
  JOIN (
    SELECT id, ora_inizio, ora_fine FROM slot
  ) AS s1 ON instr(r1.slots, CAST(s1.id AS TEXT)) > 0
 GROUP BY a1.id_aula, r1.giorno, slot_id
 HAVING occupazioni > 1;
```

- In presenza di richieste critiche, l'algoritmo ha attivato correttamente la **funzione di riassegnazione dinamica**, liberando slot occupati da richieste meno critiche.
- In mancanza di alternative compatibili, l'algoritmo ha restituito il fallimento dell'assegnazione. Questa situazione si è presentata solo con uno stress-test dell'algoritmo molto elevato, ovvero aggiungendo molte richieste identiche tra loro, in maniera anche poco realistica.

Analisi critica

- Il sistema mostra una buona **scalabilità**, riuscendo a gestire centinaia di richieste e decine di aule in maniera efficiente.
- La logica di **ordinamento per priorità** (numero di alternative) si è dimostrata efficace nel massimizzare la soddisfazione delle richieste “più difficili”.
- Il limite principale risiede nel fatto che, in scenari di saturazione estrema, **nessuna riassegnazione sarà possibile**, e alcune richieste resteranno inevase: è un limite strutturale dovuto alla disponibilità fisica delle risorse.

Capitolo 8

Conclusioni

Il presente progetto rappresenta un primo passo verso la digitalizzazione e l'ottimizzazione del processo di assegnazione delle aule universitarie. Pur trattandosi di una piattaforma funzionante e testata con successo in scenari realistici, essa deve essere considerata come un **punto di partenza** e non di arrivo. Le potenzialità evolutive e le possibili migliorie sono infatti numerose.

In primo luogo, in un contesto reale come quello di un ateneo, la piattaforma dovrebbe essere **integrata all'interno di un sistema gestionale già esistente**, nel quale l'assegnazione delle aule può avvenire ancora in forma manuale o semi-automatica. Un'integrazione progressiva consentirebbe di sfruttare i dati già presenti nel sistema dell'università e al contempo facilitare l'adozione della piattaforma da parte del personale amministrativo e docente.

Attualmente, la logica di assegnazione è progettata per **allocazioni ricorrenti settimanali**, come quelle tipiche dei calendari delle lezioni di un semestre. Tuttavia, molte necessità organizzative di un ateneo riguardano **richieste occasionali e non periodiche**, come esami, seminari, eventi o recuperi straordinari. Per gestire questi casi, sarebbe opportuno sviluppare un modulo dedicato all'inserimento manuale di richieste “una tantum”, da parte dell'amministrazione o del docente stesso, attraverso un'interfaccia specifica. Queste richieste dovrebbero essere trattate secondo logiche differenti e indipendenti dal calendario settimanale fisso.

Un ulteriore sviluppo possibile riguarda la **gestione avanzata delle dotazioni delle aule**. In futuro, si potrebbero prevedere attributi aggiuntivi per ciascuna aula (ad esempio accessibilità, impianti audio, lavagne elettroniche, ecc.) che il docente possa specificare come requisiti in fase di richiesta, rendendo il sistema ancora più flessibile e aderente alla realtà.

In termini di logica algoritmica, un possibile miglioramento consisterebbe nell'introdurre un meccanismo che, in assenza di slot disponibili esattamente corrispondenti a quelli richiesti, **proponga soluzioni alternative "vicine"**: ad esempio slot contigui o orari leggermente diversi, purché compatibili con la capienza e le dotazioni richieste.

Infine, si potrebbe aggiungere una funzionalità che consenta all'amministratore

di **modificare manualmente un’assegnazione**, qualora si ritenga più opportuno attribuire un’aula diversa rispetto a quella proposta automaticamente. In questo scenario, l’algoritmo potrebbe fornire una lista di **alternative disponibili**, che l’amministratore può valutare prima di confermare o sostituire un’assegnazione esistente.

In sintesi, la piattaforma sviluppata costituisce una base solida e flessibile, pronta per essere ampliata, adattata e potenziata al fine di supportare in maniera sempre più intelligente ed efficiente la complessa organizzazione logistica delle università moderne.

