Università
Ca'Foscari
Venezia

GRUPPO

NOME	MATRICOLA
Jinpeng Zhang	886854
Rebecca Frisoni	885771
Martina Ragusa	885113

A.A. 2022/2023

WebApp per la creazione e gestione di prove ed esami

Indice

Introduzione

Il gruppo ha scelto di sviluppare il progetto di una WebApp per la gestione degli esami universitari. In questo documento verrà illustrata la struttura del progetto e come è stato sviluppato. Inizialmente verranno descritte le funzionalità principali fornite dall'applicazione. Successivamente verrà spiegato come è stata effettuata la modellazione concettuale e logica della base di dati e verrà illustrata una selezione di query importanti del progetto. Verranno descritte le strutture del back-end e del front-end e, per finire, verranno specificate le scelte progettuali che sono state utilizzate.

Contesto dell'applicazione

L'applicativo che abbiamo sviluppato consente la gestione degli esami universitari: esso è uno strumento avanzato e intuitivo pensato per agevolare i docenti nell'organizzazione degli esami e nella registrazione dei risultati accademici dei propri studenti. Questo sistema è stato progettato per semplificare il processo di gestione degli esami, permettendo ai docenti di crearne di nuovi in modo facile e veloce, e di visionare la situazione riguardante un determinato esame di ciascuno studente.

Con la nostra applicazione di gestione degli esami universitari, aspiriamo a fornire un ambiente efficiente e trasparente per facilitare il percorso degli studenti verso il successo accademico e supportare i docenti nella gestione di un processo essenziale per il mondo universitario.

Funzionalità principali

L'applicativo sviluppato permette di gestire le funzioni principali riguardanti gli esami universitari. In particolare, di seguito vengono elencate tali funzionalità:

1. Creazione di esami: i docenti hanno il potere di creare nuovi esami, specificando il numero di prove richieste per ciascun esame e altre informazioni rilevanti.
2. Gestione delle prove: Gli studenti possono sostenere una delle prove disponibili in uno degli appelli messi a disposizione. A seguito del superamento di tale prova verranno registrate le informazioni necessarie, tra cui la data di superamento e il voto conseguito.
3. Scadenza delle prove: Ogni prova ha una data di scadenza, oltre la quale la votazione ottenuta dallo studente non sarà più valida.
4. Visualizzazione dello stato degli studenti: Il sistema offre una panoramica dello stato di ciascuno studente, mostrando le prove valide sostenute, lo storico di tutte le prove sostenute e quelle che devono ancora essere superate per registrare l'esame.
5. Visualizzazione degli appelli: È possibile ottenere l'elenco degli appelli e degli studenti che hanno superato le diverse prove.
6. Verifica dell'idoneità all'esame: Gli studenti possono verificare se sono in condizione di registrare l'esame, visualizzando le prove valide già sostenute e quelle ancora necessarie per completare il percorso accademico.

Progettazione della base di dati

La progettazione della base di dati per l'applicazione di gestione degli esami universitari è stata svolta con attenzione per garantire l'efficienza, l'integrità e la coerenza dei dati all'interno del sistema. La base di dati gioca un ruolo fondamentale nell'archiviazione e nella gestione dei dati relativi agli esami, alle prove, agli studenti e agli appelli. Di seguito sono presentati i principi guida e gli elementi chiave della progettazione della base di dati:

1. **Analisi dei requisiti**
   La progettazione della base di dati è stata avviata da un'analisi approfondita dei requisiti dell'applicazione. Sono stati identificati i principali attori coinvolti, come docenti e studenti, nonché le entità principali, come Docente, Studente, Esame e Prova. Inoltre, sono state individuate le relazioni tra le diverse entità, come la creazione degli esami da parte dei Docenti.
2. **Normalizzazione dei dati**
   Per garantire la ridondanza dei dati e l'integrità referenziale, sono stati applicati i principi di normalizzazione delle tabelle. Le tabelle sono state organizzate fino alla forma normale desiderata, evitando dipendenze funzionali ridondanti e assicurando che ciascun attributo sia strettamente legato alla chiave primaria della tabella.
3. **Definizione delle tabelle e degli attributi**
   Le tabelle sono state definite in modo coerente e conforme alla struttura delle entità coinvolte nell'applicazione. Ogni tabella ha una chiave primaria univoca per identificare in modo inequivocabile ogni record. Sono stati inoltre definiti gli attributi necessari per ciascuna entità, garantendo la registrazione accurata delle informazioni.

Progettazione concettuale

Abbiamo utilizzato il modello concettuale per modellare i dati della nostra base di dati. Come primo step abbiamo individuato le entità principali coinvolte nel dominio di interesse: Studenti, Esami, Prove e Docenti.

La tabella Studente presenta i seguenti attributi:

- idS, il quale rappresenta il numero di matricola, e che costituisce la chiave primaria
- nome
- cognome

La tabella Esame contiene le informazioni relative agli esami e presenta quattro attributi:

- idE, di tipo stringa, che identifica l’esame
- nome, di tipo stringa
- anno_accademico a cui appartiene, di tipo stringa
- cfu, di tipo int, che specifica il numero di crediti assegnati a tale esame

La tabella Docente contiene le informazioni relative ai docenti, specificate dagli attributi:

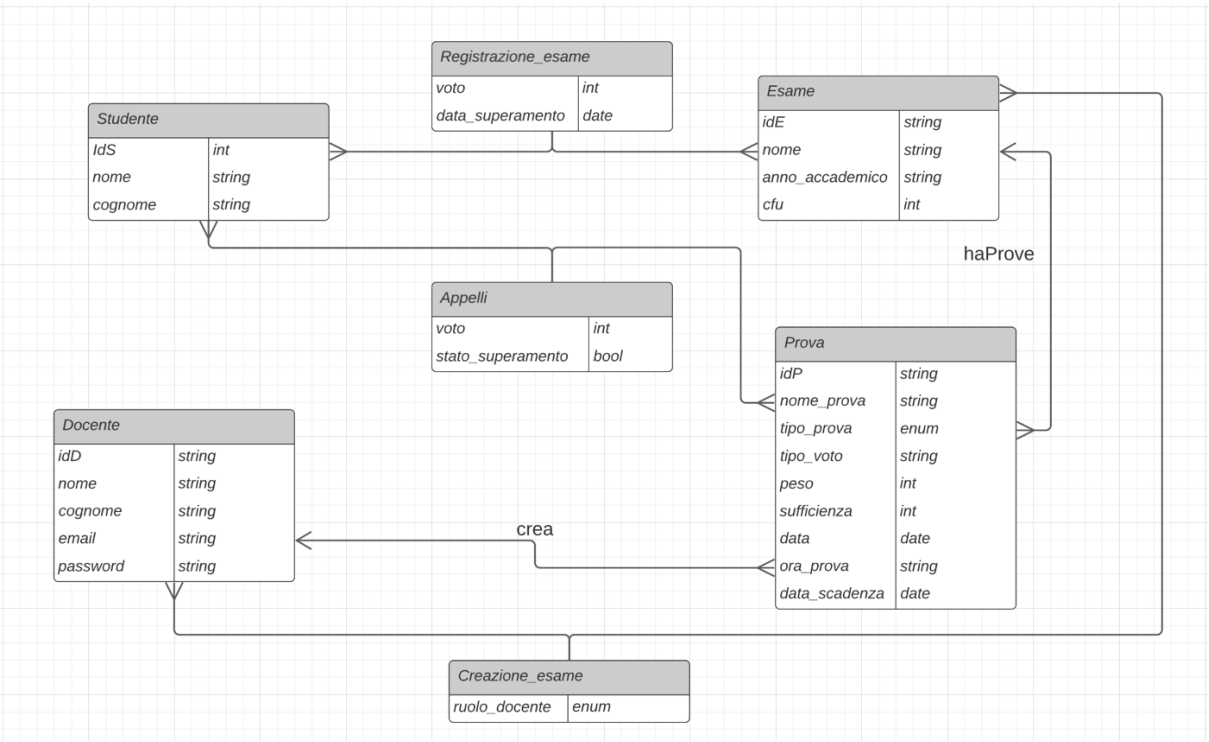
- idD, identificativo del docente
- nome (stringa)
- cognome (stringa)

- e-mail (stringa)
- password (stringa)

La tabella Prova presenta i seguenti attributi:

- idP, codice identificativo della prova, di tipo stringa;
- nome_prova (stringa)
- tipo_prova, di tipo enum, che identifica il tipo della prova
- tipo_voto (stringa)
- peso, che specifica il peso della prova in questione
- sufficienza (int)
- data (date)
- ora_prova (stringa)
- data_scadenza (date)

La classe Studenti è messa in relazione con la classe Esami attraverso una relazione ‘molti a molti’ “Registrazione_esame”, la quale contiene informazioni sul voto (di tipo int) e la data di superamento (Date). Studenti possiede un’altra relazione ‘molti a molti’ “Appelli” che contiene informazioni riguardanti un appello dell’esame. Essa ha come attributi: il voto (int), lo stato di superamento (Boolean). La classe Esami è, a sua volta, messa in relazione con la classe Docenti attraverso la relazione ‘molti a molti’ denominata “Creazione_esame”, la quale possiede l’attributo “ruolo_docente”. Essa è inoltre messa in relazione con la classe Prove con una ‘uno a molti’, denominata “haProve”. La classe Prove è, a sua volta, messa in relazione con la classe Docenti con una relazione ‘molti a uno’ denominata “Crea”.



Progettazione logica

Successivamente, partendo dalla progettazione concettuale, abbiamo proseguito con la progettazione logica traducendo il nostro schema ad oggetti in uno schema relazionale.

Siamo partiti trasformando le relazioni ‘molti a uno’ e ‘uno a molti’ aggiungendo le chiavi esterne necessarie. Come ultimo step abbiamo poi proseguito trasformando le relazioni ‘molti a molti’.

Di seguito vengono descritte le varie modifiche che hanno subito le tabelle nella fase di progettazione logica.

Alla tabella Prova sono state aggiunte le due chiavi esterne:

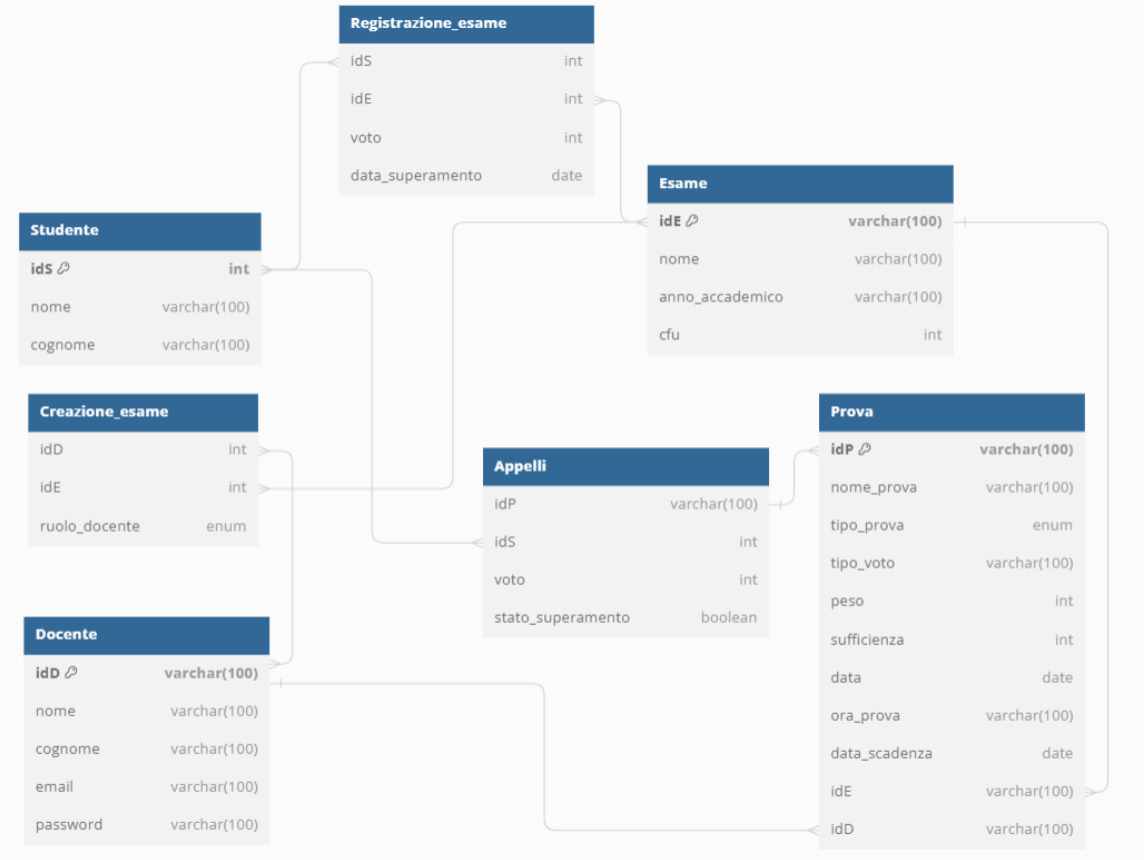
- idE, di tipo stringa, che è stato aggiunto a seguito della trasformazione della relazione con la tabella Esame
- idD, anch’esso di tipo stringa, che punta alla tabella Docente.

La relazione Registrazione_esame è stata trasformata in una tabella, al cui interno possiamo trovare:

- idS, che punta alla tabella Studente
- idE, che punta alla tabella Esame
- voto
- data_superamento

La seconda relazione ‘molti a molti’ è costituita da “Appelli”, la quale è stata a sua volta trasformata in una tabella, nella quale possiamo trovare:

- idP, puntatore alla tabella Prova
- idS, puntatore alla tabella Studente
- voto
- stato_superamento



Durante la progettazione logica siamo riusciti a identificare quali tabelle avrebbero avuto bisogno di una raccolta di informazioni obsolete, da ognuna di queste tabelle abbiamo poi creato una contro parte con il suffisso "_audit" per riuscire ad immagazzinare i dati che non hanno più alcun tipo di relazione con il nostro schema corrente ma che possono pur sempre essere visualizzati ed utilizzati nel caso fossero necessari.

Progettazione fisica

Una volta completate con successo sia la progettazione concettuale che la progettazione logica del nostro database, siamo stati in grado di ottenere una visione chiara e completa del sistema così da procedere con la progettazione fisica del database.

Inizialmente, abbiamo scelto di utilizzare il DBMS SQLite per diversi motivi che si sono dimostrati molto validi per le nostre esigenze specifiche. In primo luogo, l'aspetto della leggerezza ha giocato un ruolo fondamentale nella decisione. SQLite offre un database senza server, incorporato all'interno delle nostre applicazioni, il che ha comportato un notevole vantaggio in termini di efficienza e risorse di sistema. Le prestazioni di questo DBMS sono state sorprendenti, grazie alla sua natura leggera, alla mancanza di configurazioni complesse e alle transazioni locali che hanno reso le operazioni di lettura e scrittura molto rapide ed efficienti. Questo ci ha consentito di ottenere elevate prestazioni del database anche in situazioni in cui era richiesta una rapida elaborazione dei dati.

Un altro aspetto cruciale nella nostra scelta è stato il supporto multipiattaforma offerto da SQLite. La possibilità di utilizzare il database su diverse piattaforme ci ha garantito una maggiore flessibilità nell'implementazione delle nostre applicazioni, permettendo le

componenti del gruppo di lavorare facilmente sull'app data la presenza di sistemi operativi diversi.

Tuttavia, verso la fase finale della progettazione, abbiamo iniziato a notare alcune limitazioni e carenze associate all'utilizzo di SQLite come DBMS principale. In particolare, abbiamo riconosciuto che mancavano alcune delle funzionalità più avanzate necessarie per un'amministrazione dei dati sofisticata e per gestire progetti di maggiore complessità. La scalabilità di SQLite si è rivelata una sfida, poiché iniziavamo a lavorare con un sistema più grande e complesso, si è sentita la necessità di un DBMS in grado di affrontare carichi di lavoro più impegnativi e supportare un numero maggiore di utenti concorrenti.

Dopo un'attenta valutazione delle nostre esigenze, abbiamo deciso di migrare il nostro database e tutte le informazioni ad esso associate a PostgreSQL, un sistema di gestione di database relazionale (RDBMS) che ci ha fornito una vasta gamma di funzionalità avanzate. Questa decisione è stata guidata dal desiderio di sfruttare al meglio le potenzialità offerte da PostgreSQL, tra cui l'utilizzo di trigger per gestire automaticamente gli eventi di database, le funzioni definite dall'utente per personalizzare l'elaborazione dei dati e la scalabilità per gestire progetti complessi con un numero crescente di utenti e dati.

La migrazione al nuovo RDBMS ha rappresentato una sfida tecnica, ma ci ha fornito la soluzione più adatta alle nostre esigenze e ci ha permesso di raggiungere una maggiore efficienza e flessibilità nella gestione del database.

Ruoli

Per quanto riguarda la gestione dei ruoli e le politiche di amministrazione per il nostro progetto, al fianco del ruolo postgres, già presente di default all'interno del database, è stato definito il ruolo Docente.

Il ruolo "postgres" costituirà il superutente, ovvero colui che avrà il compito di amministrare il sistema. Avrà quindi accesso a tutte le tabelle, su cui potrà effettuare tutte le operazioni disponibili.

Avremo poi, come specificato sopra, il ruolo di Docente.

```
1. CREATE ROLE Docente NOSUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT LOGIN
2. NOREPLICATION NOBYPASSRLS PASSWORD '****';
3. GRANT SELECT, INSERT, DELETE ON TABLE Creazione_esame TO Docente;
4. GRANT SELECT, INSERT, DELETE ON TABLE Esame TO Docente;
5. GRANT SELECT, INSERT, DELETE ON TABLE Prova TO Docente;
6. GRANT SELECT, INSERT, DELETE ON TABLE Appelli TO Docente;
7. GRANT SELECT, INSERT, DELETE ON TABLE Registrazione_esame TO Docente;
```

Il ruolo Docente, come suggerisce dal nome, identificherà i docenti. Esso consente l'accesso alle tabelle relative a tutto ciò che riguarda gli esami, ovvero Esame, Prova, Appelli, Registrazione_esame e Creazione_esame. Su queste tabelle, i docenti potranno effettuare operazioni di SELECT, INSERT e DELETE, che gli consentiranno dunque di aggiungere esami, modificare e registrare i voti.

Inoltre, essendo la nostra WebApp basata su un server ed un database locale non abbiamo sentito la necessità di creare ulteriori ruoli aggiuntivi per altri eventuali utenti, cosa che sarebbe stata molto gradita se il server avesse permesso connessioni da remoto.

Principali scelte progettuali

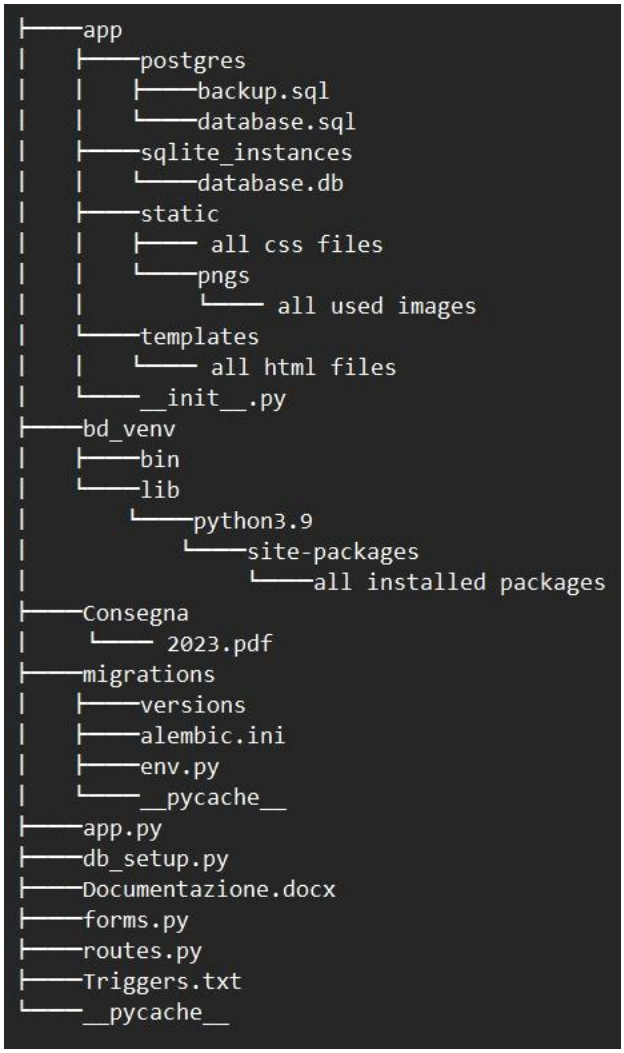
In questa sezione vengono descritte le principali scelte progettuali che sono state prese dal gruppo, distinguendo tra i linguaggi di programmazione utilizzati e i frameworks adoperati.
Per quanto riguarda i linguaggi di programmazione sono stati utilizzati Python, JavaScript, HTML, CSS, SCSS, SQL.
Inoltre, sono stati utilizzati diversi framework, tra i quali Flask, Bootstrap e Ajax.

Nella fase di progettazione del progetto, sono stati selezionati, sulla base di ciò che ci era stato presentato dal professore, i linguaggi di programmazione che avremmo utilizzato. Abbiamo utilizzato JavaScript, linguaggio di scripting, per il lato client per sviluppare le funzionalità interattive e dinamiche.
Per il lato server, invece, abbiamo utilizzato Python, linguaggio ad oggetti, sempre sulla base delle informazioni che ci erano state fornite dal professore. Esso è infatti di facile lettura e comprensione, che ci ha permesso una maggiore facilità nell'utilizzo. Inoltre, offre un'ampia varietà di framework e librerie per facilitare lo sviluppo web, come Flask, che abbiamo deciso di utilizzare.
Flask è un micro-framework basato sul linguaggio Python che, nonostante presenti solo le funzionalità di base per lo sviluppo web, per il nostro progetto, si è rivelato perfetto per sviluppare le funzionalità server-side in modo efficiente.
Per quanto riguarda il front-end abbiamo utilizzato una combinazione di HTML, CSS e JavaScript. HTML è stato adoperato per la struttura principale del contenuto; per progettare e definire l'aspetto e lo stile delle pagine web abbiamo utilizzato CSS; infine, per fornire interattività e dinamicità all'interfaccia utente è stato utilizzato JavaScript.
A fianco di CSS è stato utilizzato anche SCSS (Sassy CSS), un'estensione di sintassi per CSS che ci ha offerto funzionalità più avanzate.
In combinazione ai tre linguaggi nominati sopra, è stato utilizzato anche il framework Bootstrap che ci ha messo a disposizione varie componenti predefinite, facilitando il processo di creazione di una migliore interfaccia utente. Inizialmente si era pensato di usare anche il framework W3, consultando la documentazione online, però non forniva alla nostra WebApp l'aspetto desiderato. Siccome più membri del gruppo stavano lavorando alla parte del front-end contemporaneamente, e da una parte si era cominciato a usare Bootstrap, allora si decise di uniformare tutte le pagine a quel framework specifico. In conclusione, si capì che W3 era minimalista e aveva features basiche, ma ha dato decisamente uno spunto per il design della WebApp.
Al fianco di JavaScript, affinché l'interfaccia fosse più interattiva, abbiamo utilizzato Ajax (Asynchronous JavaScript And XML.). Ajax permette alle pagine web di essere aggiornate in modo asincrono, scambiando dati con il web server. Questo consente di modificare in parte una pagina web senza che essa venga ricaricata nella sua interezza e dunque permettendo all'interazione dell'utente di non essere interrotta.
Per la gestione del database, è stato utilizzato il linguaggio SQL (Structured Query Language) per definire e manipolare i dati. L'abbiamo impiegato per creare e manipolare le tabelle del nostro database, definendo la struttura dei dati e gestendo le operazioni di lettura, scrittura, modifica e cancellazione.

A sua volta, il supporto di Flask per l'integrazione con un database relazionale ci ha consentito di creare e gestire le tabelle, eseguire query e garantire l'integrità e la coerenza dei dati.

Struttura del back-end

Nella figura sottostante, è rappresentata la struttura completa del progetto, illustrando il modo in cui abbiamo organizzato e gestito diversi file, inclusi i componenti backend e frontend. Il progetto è composto da vari file, ognuno con un ruolo specifico nel sistema.



Segue una breve descrizione dei file e delle directories più importanti:

__init__.py: è un file specializzato che viene riconosciuto come un pacchetto contenente le risorse necessarie per eseguire l'applicazione. Tale file contiene le informazioni per connettersi al database che è basato su un server locale.

routes.py: è un file chiave del progetto, poiché contiene tutte le funzioni routes necessarie per il corretto funzionamento dell'applicazione web. Le routes rappresentano le URL a cui gli utenti possono accedere per interagire con diverse pagine del progetto.

Questo file è responsabile della gestione delle richieste degli utenti e della conseguente associazione alle funzioni appropriate; ad esempio, contiene le funzioni che gestiscono la visualizzazione degli studenti iscritti ad un determinato esame, la lista degli esami creati dai docenti, la lista delle prove sostenute da uno studente ecc…
Le funzionalità principali dell'applicazione sono implementate attraverso queste routes, garantendo un'esperienza utente intuitiva e fluida, tale file contiene tutte le funzioni routes che sono state definite per la corretta esecuzione del programma.

static/: questa cartella contiene tutti i file statici dell'applicazione, come fogli di stile CSS, script JavaScript e risorse, inclusi tutti gli elementi grafici come immagini. Poiché questi file di solito rimangono invariati durante la navigazione dell'utente, vengono definiti "statici". Utilizzando questa cartella appositamente designata, l'applicazione può organizzare e fornire facilmente questi file ai client senza doverli rigenerare ad ogni richiesta.

templates/: dentro questa cartella risiedono i template HTML che costituiscono il cuore delle pagine dinamiche dell'applicazione. I template fungono da stampi o modelli, consentendo all'applicazione di riutilizzare la struttura comune per diverse pagine e popolarla con dati specifici in modo dinamico.

forms/: questa cartella contiene tutti i moduli HTML che definiscono i form dell'applicazione, consentendo agli utenti di inserire dati e inviarli al server per l'elaborazione. Ogni modulo è costituito da campi di input, caselle di controllo, menu a tendina e altri elementi affinché gli utenti possano interagire con l'applicazione. Utilizzando queste tipo di strutture, l'applicazione può acquisire informazioni dagli utenti in modo strutturato e coerente. Questi form possono essere gestiti dalla logica del backend per eseguire azioni specifiche, come l'inserimento dei dati nel database o l'elaborazione delle richieste dell'utente.

Interazione con il database

Per interagire con il database PostgreSQL, abbiamo utilizzato il modulo SQLAlchemy, che è un'interfaccia ORM (Object-Relational Mapping) per Python. SQLAlchemy ci ha fornito uno strumento potente per rappresentare le tabelle del database come classi Python e le righe come oggetti. Questa astrazione ci ha permesso di manipolare i dati del database utilizzando oggetti e metodi Python, semplificando notevolmente le operazioni di accesso e modifica dei dati. L'uso dell'ORM ci ha fornito un'interfaccia di alto livello per eseguire query al database senza dover scrivere direttamente SQL. Per esempio nella query seguente che riguarda la ricerca degli studenti tramite la barra di ricerca la sintassi risulta diversa da SQL:

```
search_query_with_wildcard = f"%{search_query}%"

# Query students based on matricola (idS), Nome, Cognome, Nome + Cognome, or Cognome + Nome
query = db.session.query(Studente).filter(
    or_(
        cast(Studente.idS, String).ilike(search_query_with_wildcard),
        cast(Studente.nome, String).ilike(search_query_with_wildcard),
        cast(Studente.cognome, String).ilike(search_query_with_wildcard),
        cast(Studente.nome + ' ' + Studente.cognome, String).ilike(search_query_with_wildcard),
        cast(Studente.cognome + ' ' + Studente.nome, String).ilike(search_query_with_wildcard)
    )
).all()
```

Inoltre, l'utilizzo di SQLAlchemy ha reso il nostro codice più indipendente dal tipo di database sottostante. Quindi, se dovessimo in futuro cambiare il DBMS da PostgreSQL a un altro sistema, dovremmo apportare solo poche modifiche nella configurazione di SQLAlchemy, mantenendo inalterata gran parte della logica di accesso ai dati.

Query principali

Di seguito verranno commentate alcune delle query principali, in particolare quella per visualizzare la lista degli studenti che hanno superato un esame in particolare, quella per visualizzare la lista degli esami che sono stati superati da uno studente e quella per la visualizzazione degli studenti che hanno superato tutte le prove che compongono un esame.

Per quanto riguarda la prima query, necessaria per avere una visione di tutti gli studenti che hanno superato un determinato esame, la tabella degli Studenti viene messa in relazione con la tabella Appelli attraverso un'operazione di join. Il risultato dell'operazione di giunzione viene filtrato selezionando solamente gli studenti che compaiono all'interno della tabella Appelli.

```
lista_studenti = (
    db.session.query(Studente)
    .join(Appelli)
    .filter(Appelli.idP == idP)
    .all()
)
```

La query seguente, invece, viene utilizzata per visualizzare la lista di tutti gli esami che sono stati superati da uno studente, ovvero deve essere verificata la condizione per cui tutte le prove presentino come stato di superamento "true". Per realizzare questa query, dunque, viene effettuata un'operazione di join che mette in relazione la tabella Registrazione_esame con la tabella Esame, confrontando l'attributo idE presente in entrambe le tabelle. Il risultato delle operazioni di giunzione viene filtrato e vengono selezionate solo le tuple che soddisfano la condizione Registrazione_esame.idS == idS.

```
#lista_esami_passati is the list of exams that the student has passed, meaning that all the Prova must have the stato_superamento = True
# and the sum of their percentage must be 100
lista_esami_passati = (
    db.session.query(Esame.nome, Esame.anno_accademico, Esame.cfu, Registrazione_esame.voto, Registrazione_esame.data_superamento)
    .join(Registrazione_esame, Esame.idE == Registrazione_esame.idE)
    .filter(Registrazione_esame.idS == idS)
)
```

Successivamente, viene descritta un'ultima query che, per ogni studente, mostra tutte le prove sufficienti che ammontano a valore 100 di un certo esame.

Ossia, il peso delle prove che ha passato uno studente di un determinato esame deve ammontare al 100%, ciò implica che lo studente ha superato tutte le prove di un esame e quindi pronto per la verbalizzazione del voto.

Per semplificare il codice viene utilizzata una sottoquery in cui viene effettuata un'operazione di giunzione tra la tabella Appelli e la tabella Prova (confrontando l'attributo idP). Dal risultato di questa giunzione vengono poi selezionati solo gli appelli il cui stato superamento è uguale a "True".

È stato utilizzata la clausola GROUP BY(Appelli.idS) per effettuare l'operazione di raggruppamento e dunque partizionare le righe del risultato utilizzando l'attributo idS. Infine, viene utilizzata la clausola HAVING per mostrare solamente gli studenti che hanno superato le prove che ammontano a 100 e la cui somma del peso di ciascuna prova sia minore o uguale a 100.

```
subquery = (
    db.session.query(Appelli.idS)
    .join(Prova, Prova.idP == Appelli.idP)
    .filter(Appelli.stato_superamento == True, Prova.idE == idE)
    .group_by(Appelli.idS)
    .having(func.sum(Prova.peso) == 100 and func.sum(Prova.peso) <=100)
    .subquery()
)

lista_voti_studenti = (
    db.session.query(Studente.idS, Prova.idP, Prova.data, Prova.tipo_prova, Prova.tipo_voto, Prova.peso, Appelli.voto, Prova.sufficienza)
    .join(Appelli, Studente.idS == Appelli.idS)
    .join(Prova, Prova.idP == Appelli.idP)
    .filter(Studente.idS.in_(subquery), Prova.idE == idE)
    .all()
)
```

Triggers implementati

Di seguito viene effettuata una descrizione dei vincoli che sono stati implementati all'interno del progetto.

Sulla tabella 'registrazione_esame' sono stati implementati i trigger Passed e Valid:

```
1.  CREATE FUNCTION Is_Passed() RETURNS TRIGGER
2.  AS $$
3.  BEGIN
4.      IF(FALSE=ANY(SELECT stato_superamento
5.                   FROM prova p
6.                   WHERE NEW.idE==p.idE))
7.          THEN RETURN NULL
8.      END IF
9.      RETURN NEW
10. END
11. $$ LANGUAGE PLPGSQL
12.
13. CREATE TRIGGER Passed
14. BEFORE UPDATE OR INSERT
15. ON registrazione_esame
16. FOR EACH ROW
17. EXECUTE FUNCTION Is_Passed()
```

Il trigger soprastante Passed è stato creato per verificare lo stato superamento di un determinato esame. La funzione Is_Passed() viene chiamata prima dell'inserimento e dell'aggiornamento della tabella "registrazione_esame". Esso permette di garantire che l'esame venga registrato solamente quando tutte le prove corrispondenti sono state superate. In caso contrario viene generato un errore e l'operazione di inserimento o aggiornamento viene interrotta.

```
CREATE FUNCTION Is_Valid() RETURNS TRIGGER
AS $$
BEGIN
    IF( GETDATE()> ANY (SELECT p.data_scadenza
                        FROM prova p
                        WHERE NEW."idE"==p."idE"))
        THEN RETURN NULL;
    END IF;
    RETURN NEW;
END
$$ LANGUAGE PLPGSQL

CREATE TRIGGER Valid
BEFORE INSERT
ON registrazione_esame
FOR EACH ROW
EXECUTE FUNCTION Is_Valid()
```

Il trigger Valid sottostante è stato sviluppato per controllare la validità di una prova. Per questo, viene chiamata la funzione Is_Valid() che verifica che la data di scadenza della prova in questione non sia antecedente alla data attuale.

Sulla tabella 'Appelli' sono presenti tre triggers: Only_One, Sostitution, First_Test e Sufficiente.

```
CREATE FUNCTION Secure_Only_One() RETURNS TRIGGER
AS $$
BEGIN
    IF(EXISTS(SELECT *
              FROM "Appelli" a
              WHERE NEW."idS"=a."idS" AND NEW."idP"=a."idP"))
        THEN RETURN NULL;
    END IF;
    RETURN NEW;
END
$$ LANGUAGE PLPGSQL

CREATE TRIGGER Only_One
BEFORE INSERT
ON Appelli
FOR EACH ROW
EXECUTE FUNCTION Secure_Only_One()
```

Il trigger Only_One soprastante viene utilizzato per controllare che uno studente sia iscritto solamente ad un appello di una prova. La funzione Secure_Only_One() associata viene eseguita prima dell'operazione di inserimento nella tabella Appelli. Nel caso in cui lo studente sia già iscritto all'appello viene generato un errore, e l'operazione viene interrotta.

```
1.  CREATE FUNCTION Is_Replaced() RETURNS TRIGGER
2.  AS $$
3.  BEGIN
4.      IF(NEW."idP" IN ( SELECT "idP"
5.                        FROM "Appelli")
6.          AND NEW."idS"=( SELECT "idS"
7.                          FROM "Appelli"
8.                          WHERE "idP"=NEW."idP"))
9.          THEN
10.         UPDATE "Appelli"
11.         SET "voto"=NEW."voto", "stato_superamento"=NEW."stato_superamento"
12.         WHERE "idP"=NEW."idP" AND "idS"=NEW."idS";
13.     END IF;
14.     RETURN NEW;
15. END
16. $$ LANGUAGE PLPGSQL
17.
18. CREATE TRIGGER Sostitution
19. AFTER INSERT OR UPDATE
20. ON Appelli
21. FOR EACH STATEMENT
22. EXECUTE FUNCTION Is_Replaced()
```

Il trigger Sostitution è stato sviluppato per garantire che, dopo il sostenimento di una prova per il quale lo studente presentava già un voto, venga aggiornato lo stato della prova stessa. La funzione Is_Replaced() viene eseguita dopo l'operazione di inserimento. Se la condizione è soddisfatta, ovvero se lo studente aveva già sostenuto la prova ad un appello precedente, i dati della tabella vengono modificati seguendo il seguente criterio:

- Se la votazione ricevuta all'appello più recente (NEW.voto) è sufficiente, il voto già registrato viene sostituito da quello nuovo ricevuto;
- Se la votazione ricevuta (NEW.voto) risulta insufficiente, la prova sostenuta in precedenza viene invalidata, con conseguente registrazione della nuova (NEW.stato_superamento) a fallimento.

Il trigger First_Test è stato implementato per garantire che uno studente non si possa iscrivere per sostenere la seconda parte di un esame se non ha ancora superato la prima. La funzione Passed_First_Test() viene eseguita prima dell'operazione di inserimento e aggiornamento.

L'ultimo trigger che troviamo su questa tabella è Sufficiente. Esso è un BEFORE trigger che esegue la funzione prova_is_Sufficiente() prima di un'operazione di inserimento o aggiornamento e controlla se la prova è sufficiente.

Sulla tabella 'Esame' sono stati implementati i trigger Delete_Verbalized_Exam, Double_Exam_Name e save_history_esami.
Il trigger Delete_Verbalized_Exam è stato sviluppato per garantire che non venga eliminato un esame il cui voto è già stato verbalizzato, chiamando la funzione Deny_Delete_Verbalized_Exam().
Il trigger Double_Exam_Name verifica, tramite l'esecuzione della funzione Is_Exam_Double(), che due esami non siano uguali.

Per ultimo, il trigger save_history_esami esegue un controllo sull'anno e aggiunge l'esame nella tabella esame_audit.

Sulla tabella 'Prova' sono stati sviluppati i seguenti trigger: Delete_Test_Done, Double_Test_Different_Exam, Double_Test_Name, Double_Date_Test e save_history. Similmente al trigger Delete_Verbalized_Exam sulla tabella 'Esame', il trigger Delete_Test_Done garantisce che non venga cancellato una prova che fa parte di un esame il cui voto è già stato verbalizzato.
Sulla tabella 'Prova' è presente anche il trigger Double_Test_Different_Exam che chiamando la funzione associata Deny_Double_Test_Different_Exam() controlla che un docente non crei due prove identiche all'interno di due esami differenti.
Il trigger Double_Test_Name garantisce che un docente non crei due esami o prove con lo stesso nome.
Il trigger save_history è un BEFORE trigger che controlla l'anno e aggiunge la prova in questione nella tabella prova_audit.

```
1.  CREATE FUNCTION Deny_Insert_Double_Date_Test() RETURNS TRIGGER
2.  AS $$
3.  BEGIN
4.      IF(EXISTS ( SELECT *
5.                  FROM prova
6.                  WHERE NEW."idP"<>"idP" AND NEW."data"="data"))
7.          THEN RETURN NULL;
8.      END IF;
9.      RETURN NEW;
10. END
11. $$ LANGUAGE PLPGSQL
12.
13. CREATE TRIGGER Double_Date_Test
14. BEFORE INSERT
15. ON prova
16. FOR EACH ROW
17. EXECUTE FUNCTION Deny_Insert_Double_Date_Test()
```

Il trigger Double_Date_Test è stato sviluppato per garantire che uno studente non si iscriva a più prove lo stesso giorno. La funzione Deny_Insert_Double_Date_Test() viene eseguita prima dell'operazione di inserimento nella tabella Prova. Nel caso in cui i dati che si stanno cercando di inserire presentino la stessa data di quella già presente nel sistema, ma l'identificativo della prova non coincida, viene generato un errore e l'operazione di inserimento viene interrotta.

Sicurezza

Per garantire sicurezza all'interno della nostra WebApp sono state implementate tecniche di autenticazione e autorizzazione. Per questo, coloro che intendono utilizzarla devono essere utenti registrati ed effettuare il login. All'interno del file routes.py, infatti, possiamo trovare la parte riguardante l'autenticazione degli utenti, che gestisce il processo di accesso. Quando un utente richiede l'accesso dopo aver inserito le proprie credenziali, viene verificata la validità di tali credenziali e, se i dati sono corretti, viene fornito l'accesso.

```
if form.validate_on_submit():
    docente = Docente.query.filter_by(email=form.email.data).first()  # parametrizzato
    if not docente or not check_password_hash(docente.password, form.password.data):
        flask.flash('Invalid username or password.')
        return flask.redirect(flask.url_for('routes.login'))
    else:
        login_user(docente)
        flask.flash('Logged in successfully.')
        return flask.redirect(flask.url_for('routes.homepage'))  # Redirect to the homepage route
```

Nella porzione di codice soprastante possiamo notare l'utilizzo di check_password_hash, ossia una tecnica che serve per criptare la password originale per mantenerla segreta, tale algoritmo viene sfruttato una volta che si tenta l'accesso alla piattaforma andando a confrontare la password che è stata inserita all'interno del form con la stringa decriptata che si trova all'interno del database.
Al login abbiamo poi implementato un'ulteriore funzionalità che tiene traccia dei login degli utenti andando ad immagazzinare e salvare i dati sensibili di chi ha tentato l'accesso.
Nella figura sottostante vediamo come è stata utilizzata tramite le librerie fornite dal framework Flask:

```
logging.basicConfig(filename='accessi_utenti.log', level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

@bp.route('/', methods=['GET', 'POST'])
def login():
    from forms import Login_form
    from werkzeug.security import check_password_hash
    from flask import flash
    import time

    form = Login_form()

    if current_user.is_authenticated:
        return redirect(url_for('routes.logout'))      You, 1 second ago • Uncommitted changes

    if form.validate_on_submit():
        docente = Docente.query.filter_by(email=form.email.data).first()  # parametrizzato
        if not docente or not check_password_hash(docente.password, form.password.data):
            flash('Invalid username or password.')
            logger.warning(f"Failed login attempt for user with email: {form.email.data} from IP: {request.remote_addr}")
            return redirect(url_for('routes.login'))
        else:
            login_user(docente)
            flash('Logged in successfully.')
            logger.info(f"User with email: {form.email.data} logged in successfully from IP: {request.remote_addr}")
            return redirect(url_for('routes.homepage'))

    if request.referrer and 'logout' in request.referrer:
        flash('You need to log in to access this page.')

    return render_template('login.html', form=form)
```

Per garantire la sicurezza dell'applicazione, abbiamo implementato diverse tecniche, tra cui la parametrizzazione delle query per prevenire SQL injection. Fortunatamente l'ORM di Flask semplifica notevolmente questo aspetto, proteggendo i dati sensibili automaticamente.
Inoltre, per l'accesso al database in postgres piuttosto che utilizzare una stringa salvata all'interno del codice abbiamo deciso di utilizzare un meccanismo di python che permette di settare ed utilizzare determinate variabili salvate come variabile d'ambiente senza andare a specificare il valore di tali variabili per mantenere la segretezza dei dati sensibili.
Infatti, nella seguente immagine possiamo osservare come l'applicazione invia una richiesta di GET per andare a raccogliere il valore della variabile senza rivelarlo:

```
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
   You, yesterday • Create le history …
db.init_app(app)
app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY')
```

Struttura front-end

Sicuramente, nell'applicativo per la gestione degli esami, un ruolo fondamentale lo ricopre il front-end. Di seguito verranno elencate le tecnologie che sono state utilizzate per realizzarlo, che hanno giocato un ruolo fondamentale nello sviluppo di una buona interfaccia utente e di facile utilizzo. La nostra idea è stata quella di rendere l'esperienza utente per i docenti che intendono gestire gli esami universitari il quanto più semplice e intuitiva possibile.

Come già citato nella sezione "principali scelte progettuali", sono stati utilizzati HTML, CSS, SCSS, JavaScript e Ajax.

- HTML (HyperText Markup Language) ha svolto un ruolo fondamentale nel definire la struttura principale del contenuto. Siamo infatti stati in grado di organizzarlo in modo coerente e gerarchico, agevolando così la creazione di pagine web con una struttura ben organizzata e facilmente navigabile.
- CSS (Cascading Style Sheets) e SCSS (Sassy CSS) sono stati utilizzati per progettare e definire l'aspetto e lo stile delle pagine web. In specifico ci siamo serviti del carattere "#", o selettore di ID, che ha la funzione di selezionare gli elementi nel file HTML con lo stesso ID che segue l'hashtag, applicando lo stile scelto. Esso ha avuto una funzione particolarmente utile nel caso in cui più file HTML fossero collegati allo stesso CSS; quindi, con l'opportuna distinzione degli ID, è facile riconoscere il collegamento tra essi e il file HTML corrispondente. Esso è stato usato maggiormente per le introduzioni di testo nelle diverse pagine. Un'altra funzionalità di CSS che abbiamo sfruttato è il carattere ".", che è un selettore di classe, che si collega all'attributo class (al posto dell'ID, come detto precedentemente). Esso è stato particolarmente sfruttato per la search-bar e i bottoni del sito. Del resto, abbiamo usato i costrutti di CSS che permettono di riferirsi a tutti gli elementi di un certo tipo nel file HTML (come "table"), associandoci proprietà comunemente usate ("margin", per il margine attorno a ciò che è stato selezionato, "color" per il colore, "border" per il bordo…).
- JavaScript è stato utilizzato per fornire interattività e dinamicità all'interfaccia utente. Al suo fianco è stato utilizzato anche l'interfaccia Ajax, che ci ha permesso di elevare ulteriormente il grado di interattività del progetto. In particolare, nel file "functions.js"

si è definita una funzione che porta alla pagina precedente e una per abilitare o disattivare l'edit un campo di input.

Design finale

Di seguito vengono proposte alcune immagini che permettono di dare uno sguardo generale al design finale dell'applicativo.

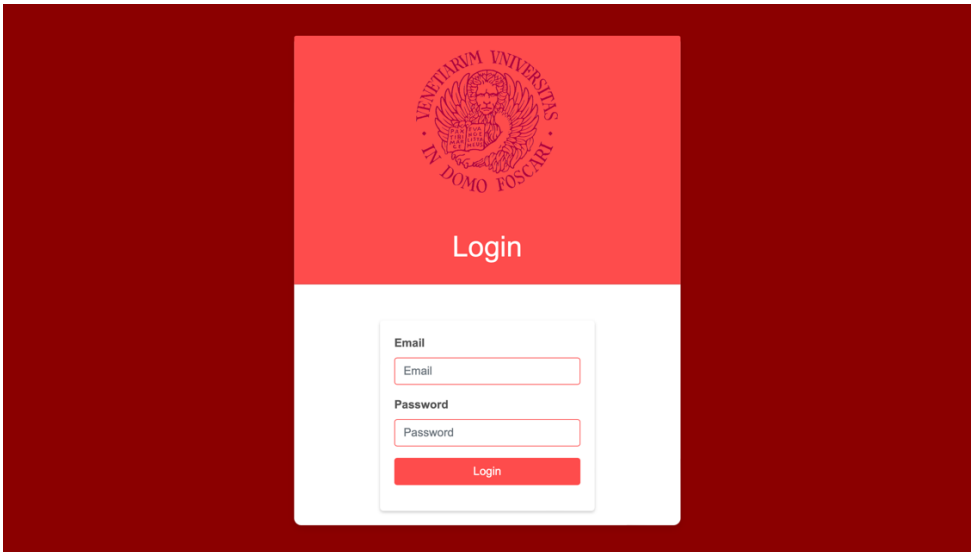


Figura 1 - Pagina di login

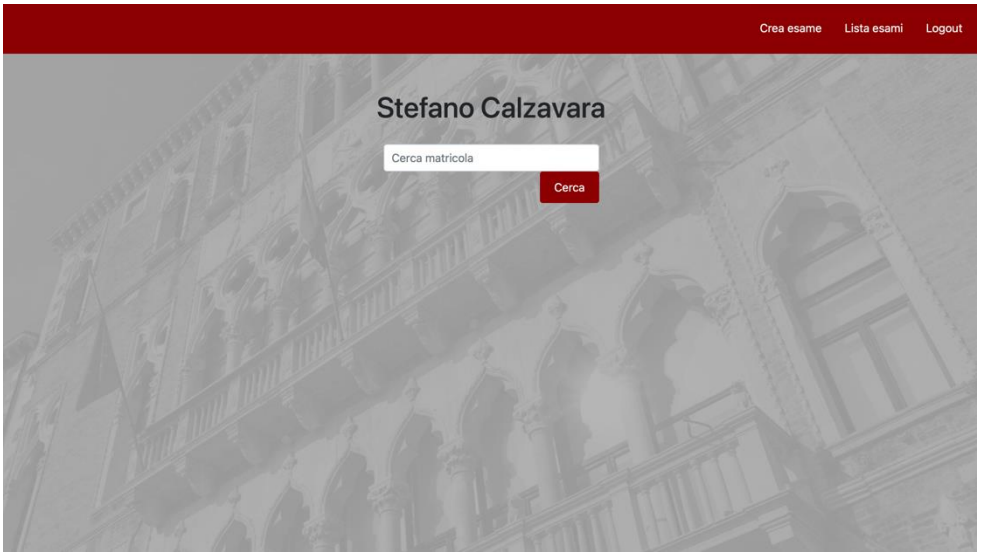


Figura 2 - Pagina che permette di visualizzare gli studenti data la matricola

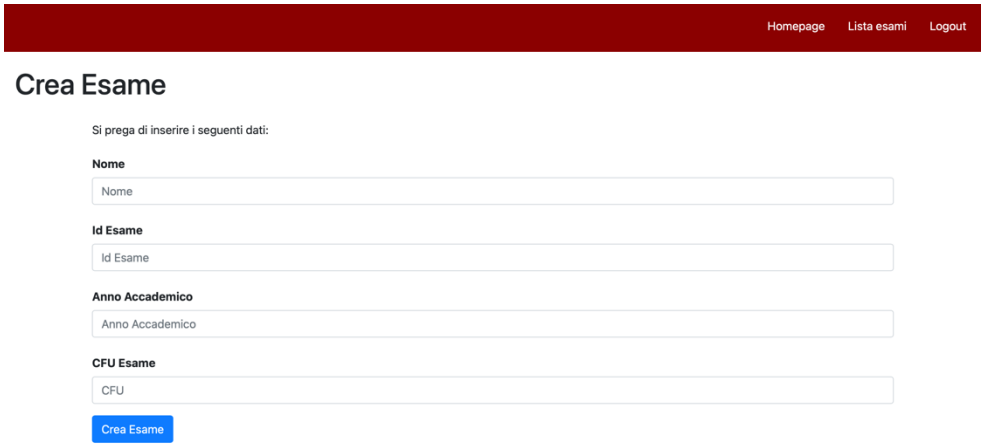


Figura 3 - Pagina di creazione di un esame

Figura 4 - Pagina di visualizzazione degli esami già presenti

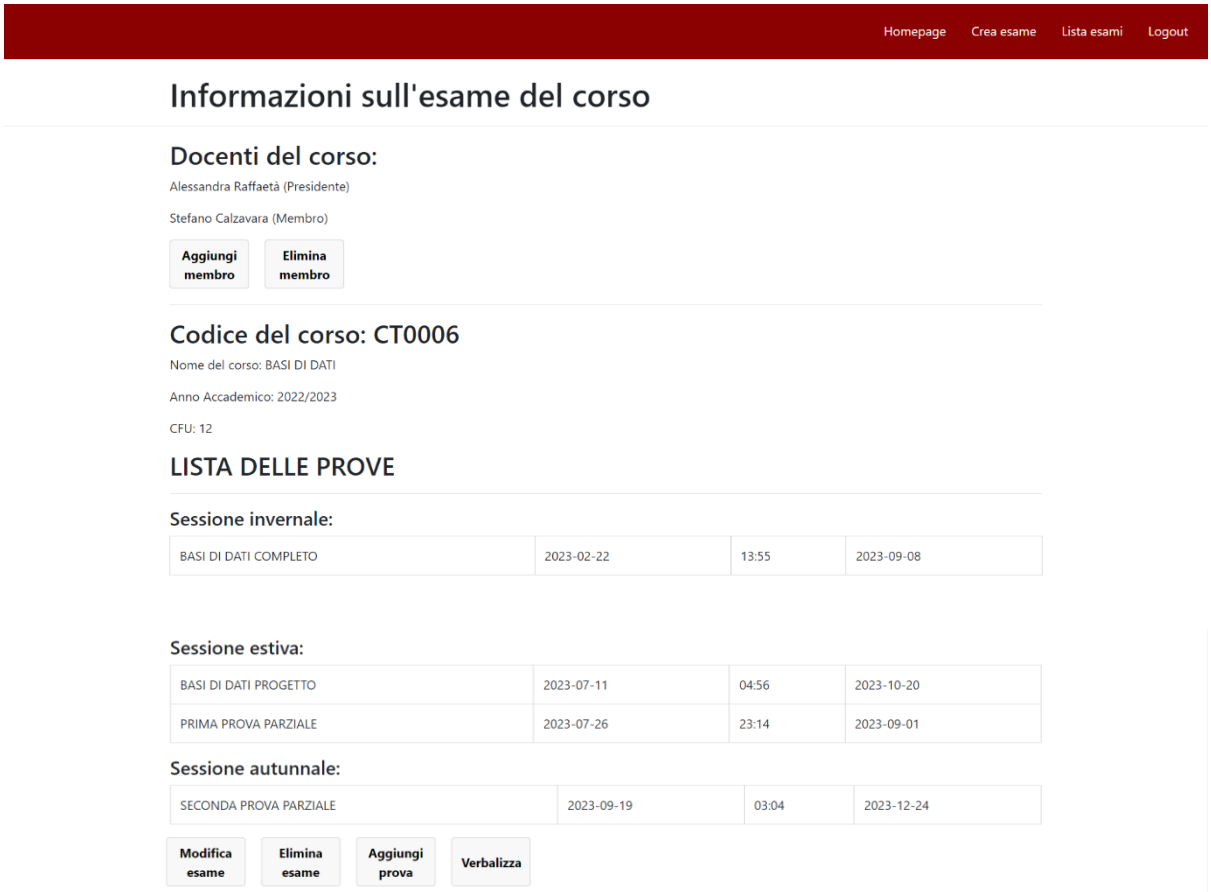


Figura 5 - Pagina riguardante un esame specifico

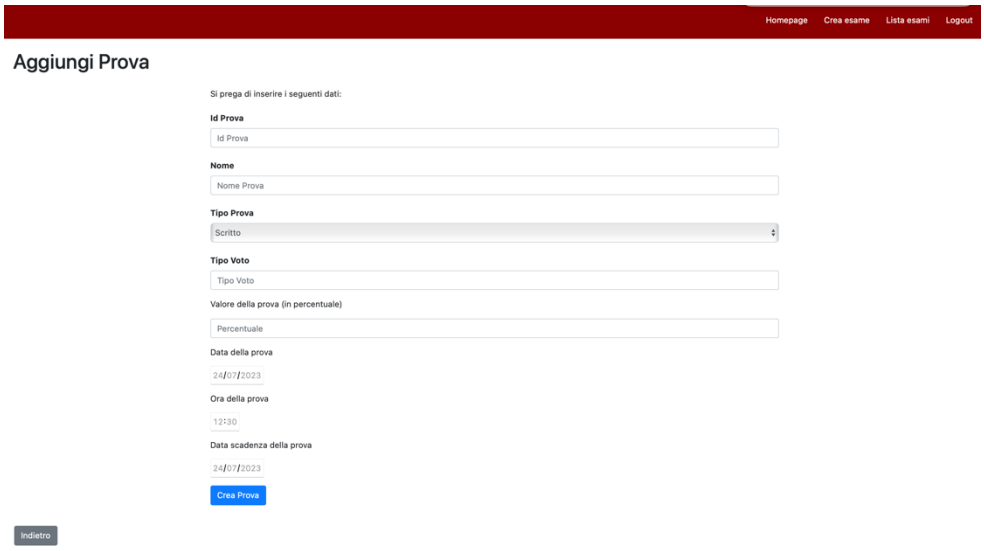


Figura 6 - Pagina per aggiungere una nuova prova

Contributo al progetto

Lo sviluppo del progetto è stato portato avanti da tutti i componenti del gruppo. In particolare, dopo una prima definizione della struttura del progetto nella sua interezza, il gruppo ha proseguito con la definizione dello schema concettuale e la successiva traduzione in schema relazionale. In seguito, il lavoro è stato diviso in tre nel seguente modo:

- Jinpeng Zhang: sviluppo front-end, back-end, sicurezza del database, implementazione triggers ed altri constraint all'interno del database
- Rebecca Frisoni: progettazione concettuale e relazionale del database, sviluppo front-end, implementazione dei trigger
- Martina Ragusa: progettazione della gestione dei ruoli, stesura documentazione, implementazione dei trigger