DreamTeam Project

Progetto realizzato per il corso Basi di Dati dell'Università di Venezia Ca' Foscari.

Indice

Introduzione

Il progetto ha l'obiettivo di creare un'applicazione Web per la gestione delle attività di orientamento (PCTO) del DAIS, il tema è stato generalizzato prendendo in considerazione il caso in cui le attività siano svolte anche da dipartimenti diversi dal DAIS. L'applicazione si basa sulla gestione di alcune entità principali (corsi, lezioni, aule, docenti e studenti) e le relazioni tra essi. A queste si aggiungono poi altre entità per specializzare la realtà di interesse (edifici, dipartimenti e categorie).

Per fare ciò, sono stati utilizzati principalmente 3 strumenti:

- **Postgresql**, sistema di gestione di database relazionale ad oggetti che utilizza SQL (linguaggio di query strutturato) come linguaggio di query principale. Viene utilizzato per gestire i dati dell'applicazione;
- **Python**, linguaggio col quale viene creato il server, fa da interfaccia tra sito web e database;
- **SQLalchemy ORM**, API che facilita l'associazione di classi Python definite dall'utente con tabelle di database e oggetti di tali classi con righe nelle tabelle corrispondenti. Le modifiche negli stati degli oggetti e delle righe vengono sincronizzate tra loro. SQLAlchemy consente di esprimere query di database in termini di classi definite dall'utente e le loro relazioni definite.
- **Flask**, micro-framework web scritto in python che consente una facile integrazione tra server e sito web. Implementa un sistema di template html popolati dinamicamente da *python*.

Qualunque utente può accedere al sito e, se opportunamente registrato, può avere accesso a funzionalità in base al suo ruolo di docente o studente. Il semplice visitatore (anonymous) ha una panoramica generale del sito con la possibilità di visualizzare i corsi esistenti e la loro descrizione, ovviamente potrà poi decidere di registrarsi al sito web.

Funzionalità

Dal sito web dell'applicazione sono presenti 4 pagine principali, alle quali si può accedere tramite la barra di navigazione presente nel bordo superiore. Le pagine sono le seguenti:

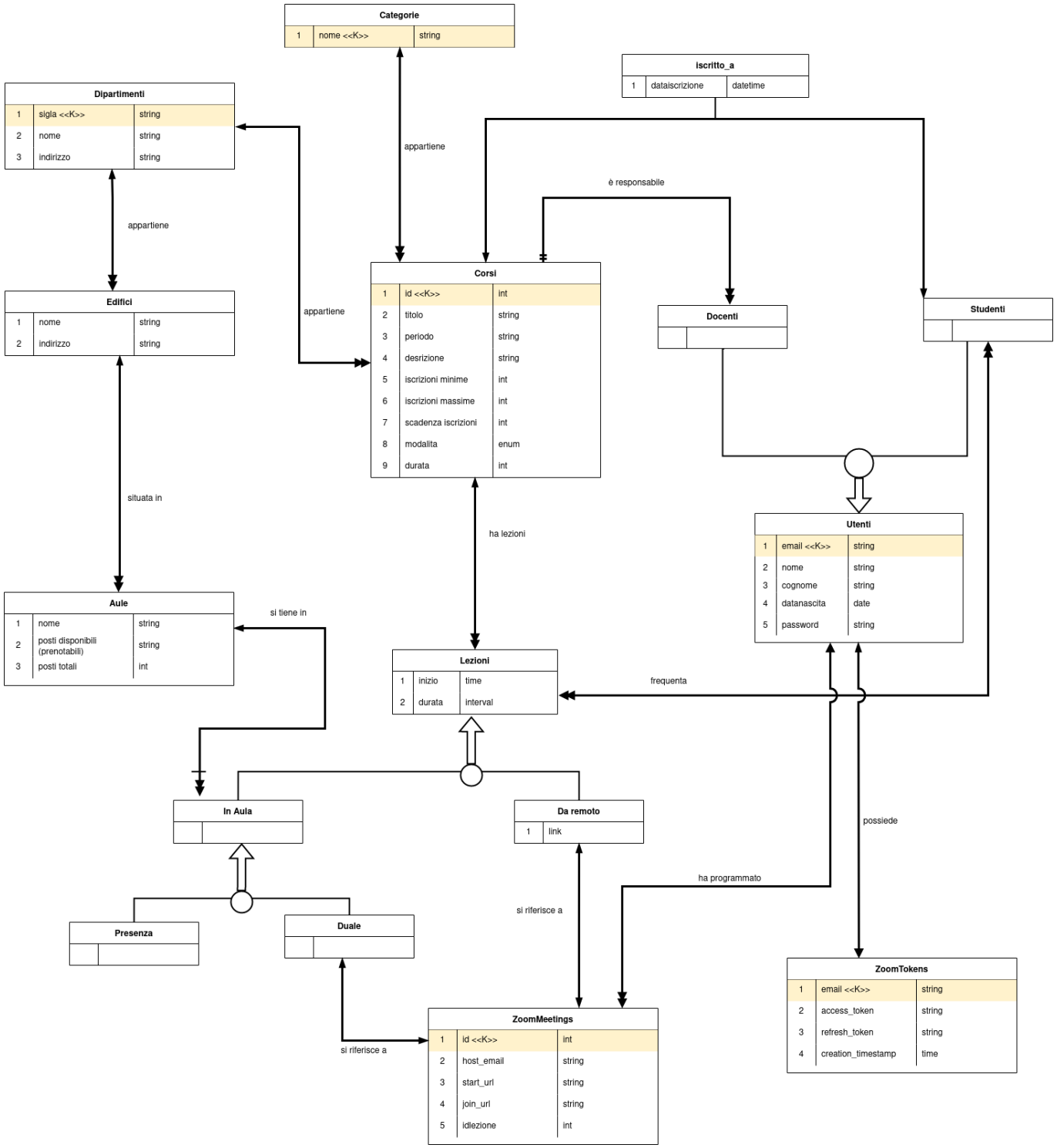
- **Home**, pagina principale, che riporta semplicemente una foto dell'Università e, nel caso l'utente abbia effettuato l'accesso, un messaggio di benvenuto;

- **Corsi**, pagina relativa ai corsi presenti all'interno dell'Ateneo. Per questa pagina bisogna effettuare 3 distinzioni a seconda dell'utente che ha fatto l'accesso:
    - *Studente*: vede i corsi a cui è iscritto e i restanti disponibili. Ovviamente potrà disiscsriversi oppure iscriversi nei limiti dei tempi prestabiliti dal responsabile del corso.
    - *Docente*: vede i corsi da lui creati e i corsi di altri docenti. Può aggiungere nuovi corsi e modificare o eliminare i suoi corsi già presenti. Inoltre è presente la possibilità di visualizzare le statistiche riguardanti gli iscritti ad ognuno dei suoi corsi.
    - *Anonymous*, utente non loggato, vede i corsi disponibili, ma ovviamente non può iscriversi o effettuare altre operazioni.
- **Lezioni**, pagina relativa alle lezioni dei corsi della pagina sopracitata. La pagina è accessibile solo ad un utente loggato, in quanto cambiano le funzionalità a seconda della tipologia di utente:
    - *Studente*: vede tutte le informazioni delle lezioni relative ai corsi a cui è iscritto, e può prenotarsi o annullare la prenotazione ad/di esse.
    - *Docente*: vede le lezioni relative ai suoi corsi, presenti e passate. Inoltre può aggiungere nuove lezioni e modificare o eliminare quelle già presenti. Inoltre quando inserisce una lezione, essa viene automaticamente schedulata su zoom in caso fosse da remoto o duale.
- **Profilo**, pagina relativa all'account con cui si ha fatto l'acceso, si possono visualizzare e modificare le proprie informazioni o effettuare il logout.
- **Login:** l'utente può effettuare il login, altrimenti ha la possibilità di registrasi come studente.

Database

Per salvare tutti i dati necessari per lo sviluppo della nostra applicazione abbiamo scelto di utilizzare il database PostgreSQL in quanto visto e utilizzato anche nel corso Basi di Dati.
Per prima cosa è stato progettato il database, decidendo quali e quante informazioni sarebbero servite durante lo sviluppo dell'applicazione.

Di seguito la rappresentazione ad oggetti della base di dati:



Legenda colori sfondi:

- **Giallo**: Key.

Si presuppone che ci siano due tipi di utenti, studenti e docenti, che condividono lo stesso tipo di informazioni ma che avranno poi funzionalità diverse, infatti le sottoclassi della gerarchia sono disgiunte. Sono presenti poi i corsi, i quali avranno un una serie di informazioni, collegati diversamente a docente e studente. I docenti potranno esserne responsabili, mentre gli studenti potranno iscriversi. Ogni corso appartiene a una categoria (esempio Informatica e statistica) e a un dipartimento (esempio DAIS). Ogni corso può essere composto da zero (lezioni non ancora inserite) o più lezioni che potranno essere frequentate dagli studenti. Le lezioni sono svolte in tre diverse modalità: presenza, remoto e duale. Questo è visualizzato tramite una doppia gerarchia, nel quale si distinguono lezioni remote da lezioni in aula. Una particolare categoria delle lezioni in aula sono

quelle svolte in duale. Per queste ultime e per le lezioni da remoto verranno schedulate i meeting di zoom a cui potranno partecipare gli studenti. Le lezioni in aula verranno svolte in un'aula (esempio Aula 1), di un particolare edificio (esempio edificio Zeta) di un certo dipartimento.

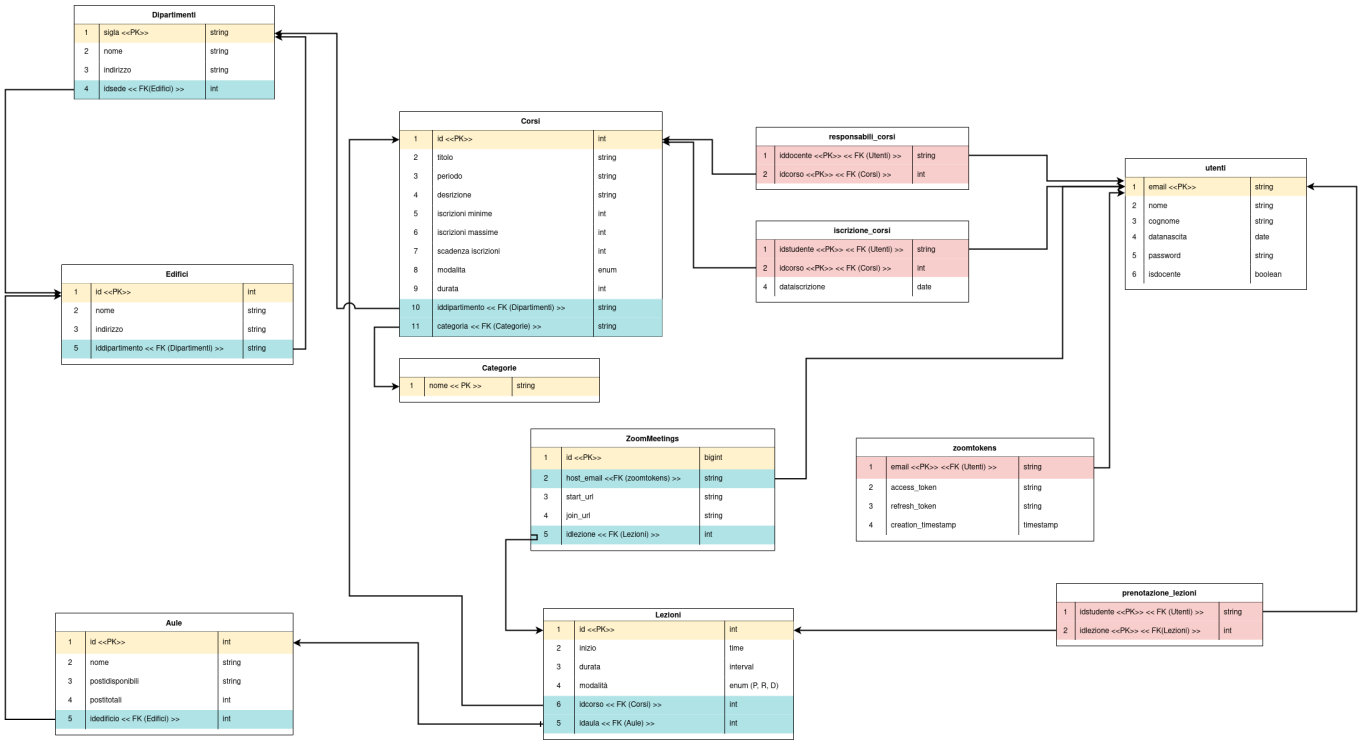
Quando un docente prova a inserire, modificare o cancellare una lezione da remoto o duale, la modifica interna al database viene anche riflessa su zoom. Per poter ottenere questo, l'utente deve concedere all'applicazione il permesso di accedere ai dati del suo account zoom. Questo si traduce a livello pratico alla cessione di una serie di token che vengono salva nella tabella zoomtokens:

- access_token: ha validità di un'ora e va incluso, tramite opportuna encryption, agli header di ciascuna richiesta all'API.
- refresh_token: ogni volta che l'access_token scade viene usato per mandare una richiesta speciale all'API di zoom per ottenerne uno nuovo. A differenza dell'access_token, il refresh_token ha una validità di 15 anni.
- creation_timestamp: timestamp (sufficientemente approssimativo) dell'inizio validità dell'access_token, necessario a calcolarne l'età ed eventualmente richiederne il refresh.

Le informazioni relative agli zoom meetings creati vengono poi storati all'interno della tabella zoommeetings, la quale mantiene per ogni meeting i seguenti attributi:

- id: identificativo gestito internamente da zoom, utilizzabile per indirizzare questo particolare meeting nelle richieste all'API
- host_email: l'email del docente che ha schedulato il meeting
- start_url: l'url che il docente può usare per avviare il meeting.
- join_url: l'url che lo studente può usare per entrare nel meeting. Se il meeting non è già stato avviato gli verrà presentata una schermata di attesa. Altrimenti entrerà in una waiting room e sarà compito del docente farlo entrare o meno.

Di seguito la rappresentazione relazionale di della nostra base di dati:



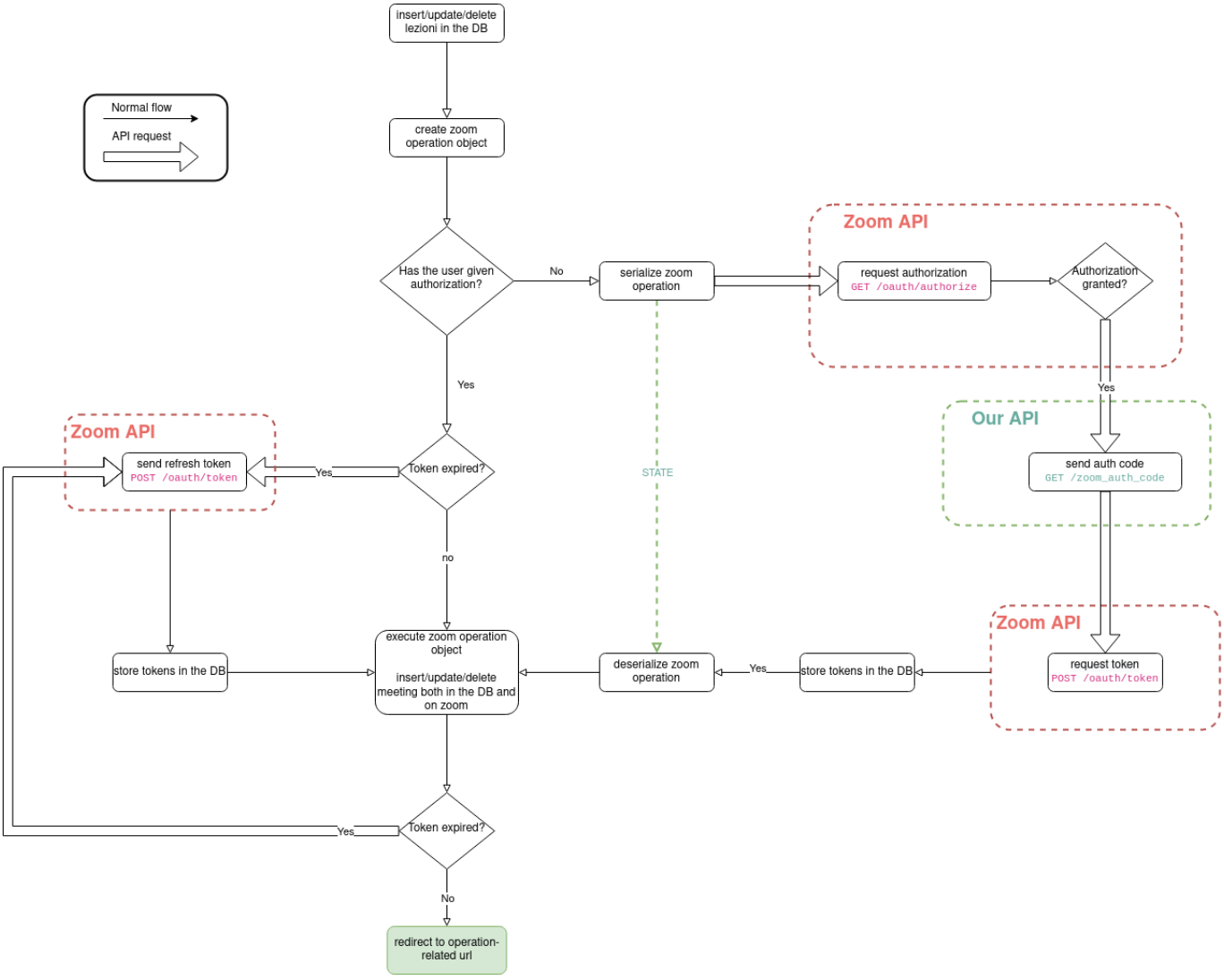
Legenda colori sfondi:

- **Giallo**: Primary Key;
- **Azzurro**: Foreign Key;
- **Rosso**: Primary Key e Foreign Key.

Dato che le sottoclassi delle gerarchie non avevano attributi che le differenziavano particolarmente viene effettuata una riduzione della gerarchia a una tabella unica. Per quanto riguarda gli utenti essi si differenziano tramite l'attributo *isDocente* di tipo boolean, mentre le lezioni si distinguono tramite l'attributo *modalità* di tipo enum, di seguito sono spiegati i valori:

- **P**: lezione in presenza;
- **R**: lezione da remoto;
- **D**: lezione duale.

Per completezza di seguito viene visualizzato il diagramma dell'interazione tra il sito web e l'API di zoom:



Server Python e Flask

Per interfacciare il database con l'applicazione web viene utilizzato Python, sfruttando la libreria SQL Alchemy. In particolare viene sfruttato l'ORM (object relational mapping), una tecnica di programmazione che consente di ottenere l'indipendenza dallo specifico DBMS sottostante, è supportato abbastanza bene da parte del compilatore, non richiede un'approfondita conoscenza di SQL e permette di astrarre da dettagli di

basso livello. Ovviamente ha dei svantaggi, come il fatto che sia più lento rispetto ad SQL e poco adatto a query complesse, ma si possono sopportare in cambio di evidenti vantaggi.

All'interno della nostra applicazione sono presenti i seguenti file:

- **db.py**: al suo interno c'è la rappresentazione del database in linguaggio python. Serve per interfacciare il server e il database. Ha un livello di astrazione maggiore rispetto al database perché ti consente di effettuare le query con ORM come se fosse un linguaggio a oggetti. Sostanzialmente, ad ogni tabella principale dello schema relazionale corrisponde una classe, mentre per le tabelle intermedie (ad esempio le tabelle necessarie per rappresentare le relazioni molti a molti) vengono utilizzate solo per costruire delle relazioni.
- **views.py**: al suo interno vi sono delle classi ulteriori create per permettere ai form presenti nell'applicazione web di interagire col database e viceversa.
- **app.py**: al suo interno vi sono gli end-point, riceve le richieste dal parte dall'utente, le quali vengono elaborate e viene restituita una risposta all'utente. Vi sono inoltre dei filtri per gestire l'interfacciamento con Jinja.
- **zoom.py**: si interfaccia con le API dell'applicativo Zoom, permettendo quindi di creare lezioni anche all'interno di esso. Si occupa inoltre di gestire i Token, i quali sono salvati all'interno di una tabella nel database.

Ogni file è opportunamente commentato, pertanto per un maggiore approfondimento si può consultare direttamente il file interessato.

Per quanto riguarda il front-end, tutti i file HTML dell'applicazione web sono contenuti nella cartella 'Templates'.
Dato che il sito si compone di pagine web abbastanza complesse, Flask aiuta tramite un sistema di **template** ereditato da **Jinja**. I template (file html) hanno una sintassi speciale che permette il passaggio di parametri da flask all'HTML in modo da generarlo dinamicamente. Questo approccio permette di disaccoppiare la logica dell'applicazione web dalla struttura della pagina HTML visualizzata.

Vi sono inoltre altre due cartelle riguardanti l'aspetto estetico e dinamico:

- static/javascript/ : la quale contiene i file .js relativi ai corrispettivi file html. Semplicemente sono degli script che permettono di passare da pagina statica a dinamica.
- static/styles/ : la quale contiene i fogli di stile .css relativi ai corrispondenti file html. Servono a definire delle regole per la visualizzazione degli oggetti che compongono la pagina.

Variabili d'ambiente

Per poter eseguire il progetto, bisogna valorizzare le seguenti variabili all'interno del file "utilities/env.txt":

`FLASK_KEY_PATH` : Path della chiave
`DB_HOST` : Indirizzo Ip del database
`DB_PORT` : Porta del database
`ZOOM_CLIENT_ID` : Id dell'applicazione per la registrazione su zoom
`ZOOM_CLIENT_SECRET` : Codice segreto fornito da Zoom
`ZOOM_REDIRECT_URI` : Url fornito da noi a zoom. Esso viene usato da zoom per dare il codice di autorizzazione nel momento in cui l'utente fornisce l'autorizzazione all'accesso ai dati al suo profilo di zoom

Scelte progettuali

All'interno del database sono stati definiti due ruoli principali: *studente* e *docente*. Si differenziano in base ai permessi per effettuare operazioni sulle varie tabelle in modo da evitare che qualcuno comprometta l'integrità del database.
Un terzo ruolo importante è l'*anonymous* che corrisponde ad un utente ordinario che visita il sito, il quale potrà visualizzare informazioni generali e iscriversi o effettuare il login. Una volta effettuato il login accederà al database con il suo ruolo principale di studente o docente.

Inoltre si è pensato di inserire un terzo ruolo *segreteria* che si occupa di operazioni particolari come l'inserimento di un docente nel database (infatti non può direttamente iscriversi dato che sarebbe un rischio), l'inserimento di aule, dipartimenti e categorie. Per mancanza di tempo questa feature non è stata ancora implementata. Per l'aggiunta di questi dati eseguire il file */docs/SQL/sampleData.sql*.

Permessi ruolo *anonymous*:

- **select**: corsi, utenti,dipartimenti, categorie.
  permesso necessario per poter visualizzare le informazioni presenti nel sito senza aver effettuato il login
- **insert**: utenti.
  permesso necessario per permettere l'iscrizione al sito

Permessi ruolo *docente*:

- **tutti i permessi *(select, update, insert , delete, trigger ...)*:** corsi, lezioni, responsabili_corsi, zoommeetings;
- **select** aule, categorie, dipartimenti, edifici, iscrizione_corsi, prenotazione_lezioni, utenti, zoomtokens.
  permesso necessario leggere e quindi visualizzare i dati presenti nelle tabelle indicate;
- **insert:** zoomtokens;
- **update:** utenti, zoomtokens.
  permesso necessario aggiornare i dati del proprio profilo e aggiornare i token di zoom.

Permessi ruolo *studente*:

- **select:** aule, categorie, corsi, dipartimenti, edifici, iscrizione_corsi, lezioni, prenotazione_lezioni, responsabili_corsi, utenti.
  permesso necessario leggere e quindi visualizzare i dati presenti nelle tabelle indicate;
- **insert:** iscrizione_corsi, prenotazioni_lezioni, utenti.
  permesso necessario per iscriversi al corso, prenotarsi alla lezione e registrarsi al sito;
- **update:** utenti.
  permesso necessario aggiornare i dati del proprio profilo.;
- **delete:** iscrizioni_corsi, prenotazioni_lezioni
  permesso necessario per la disiscrizione al corso e l'annullamento della prenotazione della lezione.

Le istruzioni sql per la creazione dei ruoli e assegnamento dei permessi si trovano in

/docs/SQL/permissions.sql

Sono stati opportunamente definiti alcuni **trigger** per garantire l'integrità dei dati. Essi riguardano principalmente corsi e lezioni.

Per i corsi è stato inserito un trigger *trg_closed_subscriptions* per evitare che lo studente si iscriva dopo la scadenza delle iscrizioni, a livello più alto viene comunque effettuato un controllo sulle date per impedire la visualizzazione dell'opzione per iscriversi, evitando di scomodare il controllo da parte del database.

Per quanto riguarda le lezioni il discorso è un po' più complicato, infatti sono stati definiti vari trigger per evitare la sovrapposizione di più lezioni nei vari casi:

- *trg_no_class_overlap_same_course_insert*: prima dell'inserimento controlla che la lezione non sia sovrapposta per data e orario ad un'altra dello stesso corso, nemmeno se vengono effettuate in aule diverse;
- *trg_no_class_overlap_insert*: prima dell'inserimento controlla che la lezione non sia sovrapposta ad un'altra nella stessa aula, anche se di corsi diversi;
- *trg_no_class_overlap_update*: si occupa di controllare che dopo l'aggiornamento della lezione essa non sia stata inserita in un'aula già occupata.

Si è presa la decisione di mantenere un controllo rilassato, permettendo la sovrapposizione di lezioni di corsi diversi dello stesso docente, in quanto è possibile avere più responsabili dello stesso corso.

- *trg_closed_subscriptions*: prima di ogni inserimento o aggiornamento sulla tabella iscrizioni_corsi, controlla che le iscrizioni al corso siano aperte;
- *trg_class_before_subscriptions_lezioni*: prima dell'inserimento o dell'aggiornamento della lezione controlla che essa venga programmata dopo la scadenza delle iscrizioni a quel particolare corso;
- *trg_closed_class_subscriptions*: prima dell'inserimento della prenotazione controlla che la lezione non sia ancora iniziata, se è già iniziata non permette di effettuare la prenotazione;

Tutti i trigger elencati sopra, in caso di fallimento, lanciano un messaggio di errore che sarà poi visualizzato nel sito per informare l'utente sull'esito dell'operazione effettuata.

Inoltre per quanto riguarda la coerenza tra la modalità delle lezioni e la modalità dei corsi sono stati definiti due trigger, *trg_modalita_lezioni* e *trg_modalita_corsi*. In questo modo quando si inserisce o si aggiorna una lezione o un corso viene verificata la modalità, se è coerente allora l'operazione viene effettuata, altrimenti l'utente visualizzerà un messaggio di errore.

Per le tabelle riguardanti zoom è stato inserito un trigger *trg_zoom_token_time* che si occupa di aggiornare l'ora di inizio validità ogni volta che vengono aggiornati i token.

La definizione dei trigger si trova in */docs/SQL/triggers.sql*

Inoltre nella creazione delle tabelle sono stati definiti alcuni vincoli check per assicurare che particolari condizioni vengano rispettate da tutti i valori presenti nell'attributo sul quale è stato definito il vincolo. Nella tabella lezioni il check inserito si occupa di garantire che le lezioni da remoto non abbiano un'aula assegnata e che invece, quelle in presenza o duali ce l'abbiano. Per quanto riguarda la tabella dei corsi i check inseriti svolgono la funzione di mantenere la coerenza tra iscrizioni minime e massime, e di verificare che la scadenza delle iscrizioni avvenga dopo l'inizio di esse. All'interno della tabella aula i check fanno in modo che posti disponibili e posti totali non vadano in conflitto tra loro.

La creazione delle tabelle e la conseguente dichiarazione dei vincoli check si può trovare nel file */docs/SQL/tables.sql*

Autori

- Giulia Cogotti, matricola 884383
- Luca Daminato, matricola 887364
- Nicola Marizza, matricola 887004