

PROGETTO DI BASI - GIUGNO/LUGLIO 2022

Gruppo:
Arceus

Opzione:
Gestione delle attività di
orientamento

Componenti:
Carola Pessotto 885384
Riccardo Sale 884471
Serena Zanon 887050

DBMS:
PostgreSQL

Versione di Python:
2022.1.3

1. Premessa e informazioni utili per l'avvio

Per la realizzazione di questo progetto, l'applicazione è stata sviluppata con PyCharm Community Edition (versione 2022.1) in Flask, utilizzando SQLAlchemy Core (ORM) per l'interfacciamento con il DBMS sottostante (PostgreSQL). Le pagine per la web app sono state sviluppate in HTML con l'ausilio di codice CSS, successivamente renderizzate da Flask. Alcuni elementi di grafica sono stati realizzati prendendo spunto dai templates di Bootstrap (<https://getbootstrap.com/>), infatti nelle pagine HTML ci sono dei link e riferimenti a file .css e .js presi appunto da Bootstrap.

Istruzioni per avviare la web app:

In primo luogo aprire la cartella “flask_basi” su PyCharm, andare nel file “settings.py” e modificare la sequenza “1234” con la propria password di PgAdmin 4. Successivamente aprire PgAdmin 4 e creare un database con il nome “dais_pcto”.

A questo punto aprire il terminale, spostarsi nella cartella del progetto e digitare “venv\Scripts\activate.bat” per entrare nell'ambiente virtuale venv, digitare “set FLASK_APP=autoapp.py” e scrivere la seguente sequenza di comandi:

- flask db init (questo creerà la cartella migrations)
- flask db migrate
- flask db upgrade

Al termine di queste operazioni il vostro DBMS conterrà tutti i campi di tutte le tabelle descritte nel codice python.

Infine, per avviare l'applicazione digitare “flask run”. Se l'istruzione avrà successo potrete visitare la web app aprendo il vostro browser di riferimento e digitando [“http://127.0.0.1:5000/”](http://127.0.0.1:5000/) nella barra degli URL.

2. Introduzione

Il progetto si basa sulla realizzazione di un'applicazione web per la gestione delle attività di orientamento (PCTO) del DAIS.

L'idea di base è realizzare una web application dove gli studenti degli istituti superiori potranno iscriversi a corsi, ciascuno composto da una o più lezioni tematiche, da svolgersi in presenza oppure online. Alla fine di ogni lezione il professore del corso fornisce un token agli

studenti che hanno seguito la lezione, i quali potranno utilizzare il suddetto token per dimostrare di aver seguito la lezione e quindi per poter ottenere alla fine del corso il certificato di partecipazione.

L'area comune parte dalla home page dove chiunque può accedervi, anche se non autenticato. A sinistra si può cliccare su "Login", per essere rimandati al form di autenticazione e poter quindi accedere alla propria area riservata, oppure su "Registrazione", per potersi registrare come nuovo utente nella piattaforma.

È possibile fare Log Out in qualsiasi momento della sessione in cui il cliente è attivo con il pulsante "Logout" presente in alto a destra della schermata.

Nell'area profilo ogni utente (studente degli istituti superiori, professore o personale della segreteria universitaria) ha accesso alla pagina di modifica dei propri dati e può visualizzare le proprie prenotazioni.

Studenti degli istituti superiori, professori o personale della segreteria universitaria avranno accesso a schermate differenti in base alle loro autorizzazioni erogate tramite i ruoli, in particolare:

- Uno studente potrà iscriversi ai corsi e visualizzare la lista delle lezioni di ogni corso
- Un professore può creare nuovi corsi da lui insegnati, può aggiungere e rimuovere lezioni dai corsi, può visualizzare le informazioni di ogni suo corso
- Il personale della segreteria può creare ed eliminare corsi, aggiungere o eliminare lezioni, aggiornare la lista di istituti superiori i cui studenti si iscrivono ai corsi offerti dal Dipartimento di Informatica e Scienze Ambientali

3. Progettazione della base di dati

A seguito della necessità di dover implementare un sistema per la gestione delle attività di orientamento (PCTO) del DAIS che ne permetta un utilizzo adeguato da parte di studenti, professori e segreteria, è necessario progettare una base di dati adeguata.

Per una realizzazione esaustiva di una base di dati utile all'interfacciamento di un sistema applicativo, capace di implementare una serie di funzionalità utili alla gestione delle attività di orientamento (PCTO), è necessario stilare un elenco di tutti gli elementi che concorrono alla formazione della stessa. Questi verranno descritti di seguito attraverso la costruzione del modello concettuale.

Analizzando le specifiche richieste per la realtà che si vuole andare a descrivere, vi è la necessità di identificare e relazionare tra loro le entità che ne fanno parte.

3.1 Modello concettuale

Entità:

- *Users*
Il sistema completo deve poter essere utilizzato da persone, siano essi professori, lavoratori della segreteria oppure studenti. Questa entità è necessaria per poter rappresentare al meglio tutti coloro che usufruiranno dei servizi dell'applicativo.
- *Courses*
Riporta tutti i corsi erogati dal Dipartimento di Scienze Ambientali, Informatica e Statistica. Ciascun corso sarà tenuto da un professore responsabile di organizzarne

le lezioni (Lessons) nel periodo previsto. Questa entità è necessaria per poter distinguere i corsi che il dipartimento offre.

- *Lessons*

Collegandosi con l'entità Courses presente nella base di dati, professori o membri della segreteria possono organizzare delle lezioni (Lessons) a cui possono partecipare gli utenti iscritti al corso.

- *HSchools*

I corsi erogati (Courses) rientrano nel piano di metodologia didattica di “Percorsi per le Competenze Trasversali e l’Orientamento” (PCTO) e sono rivolti a studenti di Istituti Superiori, pertanto ogni studente avrà una scuola (HSchools) di appartenenza.

Date le finalità di questo sistema applicativo, delineiamo le seguenti relazioni:

- *Users - (è iscritto a) - Courses*

La relazione permette ad ogni studente di iscriversi a un corso. Nel caso in cui uno studente si volesse iscrivere a ulteriori corsi, dovrà procedere con una nuova iscrizione.

- Si suppone quindi che uno studente possa iscriversi a uno o più corsi (oppure nessuno) e un corso può avere uno o più studenti iscritti (oppure nessuno)
- La relazione viene definita con cardinalità molti a molti ($N : N$)

- *Users - (insegna) - Courses*

La relazione permette di delegare un professore come responsabile di un corso, permettendogli così di modificarlo, eliminarlo e/o gestirlo. Non c'è un limite a quanti corsi uno stesso professore possa insegnare.

- Si suppone che un professore possa insegnare uno o più corsi (oppure nessuno), mentre un corso può essere insegnato da un solo professore
- La relazione viene definita con cardinalità uno a molti ($1 : N$)

- *Users - (frequenta) - HSchools*

La relazione specifica la scuola di appartenenza di uno studente per mantenere un livello alto di rintracciabilità

- Si suppone che uno studente frequenti una sola scuola, mentre una scuola può essere frequentata da uno o più studenti
- La relazione viene definita con cardinalità molti a uno ($N : 1$)

- *Users - (ha partecipato) - Lessons*

La relazione permette di tenere traccia di quali e quante lezione a cui ogni studente ha partecipato, ciò si rivelerà essere molto importante soprattutto per poter erogare il certificato di partecipazione al corso

- Si suppone che uno studente possa partecipare a una o più lezioni (oppure nessuna) e a una lezione possono partecipare uno o più studenti
- La relazione viene definita con cardinalità molti a molti ($N : N$)

- *Courses - (ha) - Lessons*

La relazione delinea la composizione dei corsi in una o più lezioni che si tengono

durante il periodo definito al momento della creazione del corso. Durante tutta la durata del corso, i giorni in cui si svolgono le lezioni potranno variare oppure rimanere inalterati.

- Si suppone che un corso possa avere una o più lezioni (oppure nessuno), mentre una lezione può appartenere a un solo corso
- La relazione viene definita con cardinalità uno a molti (1 : N)

Ora che sono state definite le relazioni, le entità necessitano di una caratterizzazione per completare la strutturazione del modello concettuale:

Users:

- user_id: identificativo univoco per ogni utente
- name: nome dell'utente registrato
- surname: cognome dell'utente registrato
- email: e-mail dell'utente registrato utilizzata nella fase di login nell'applicazione web
- password: password criptata dell'utente registrato
- role: ruolo dell'utente (professore, studente oppure segreteria)

Courses:

- course_id: identificativo univoco per ogni corso erogato dal dipartimento
- name: nome del corso associato al codice identificativo.
- mode: modalità del corso (in presenza, online oppure blended)
- description: breve descrizione del corso. Qui vengono elencati gli argomenti che si andranno a svolgere durante il corso
- max_student: numero massimo di studenti che possono iscriversi al corso
- min_student: numero minimo di studenti che possono iscriversi al corso
- n_hour: monte ore del corso
- start_month: data di inizio corso
- end_month: data di fine corso

HSchools:

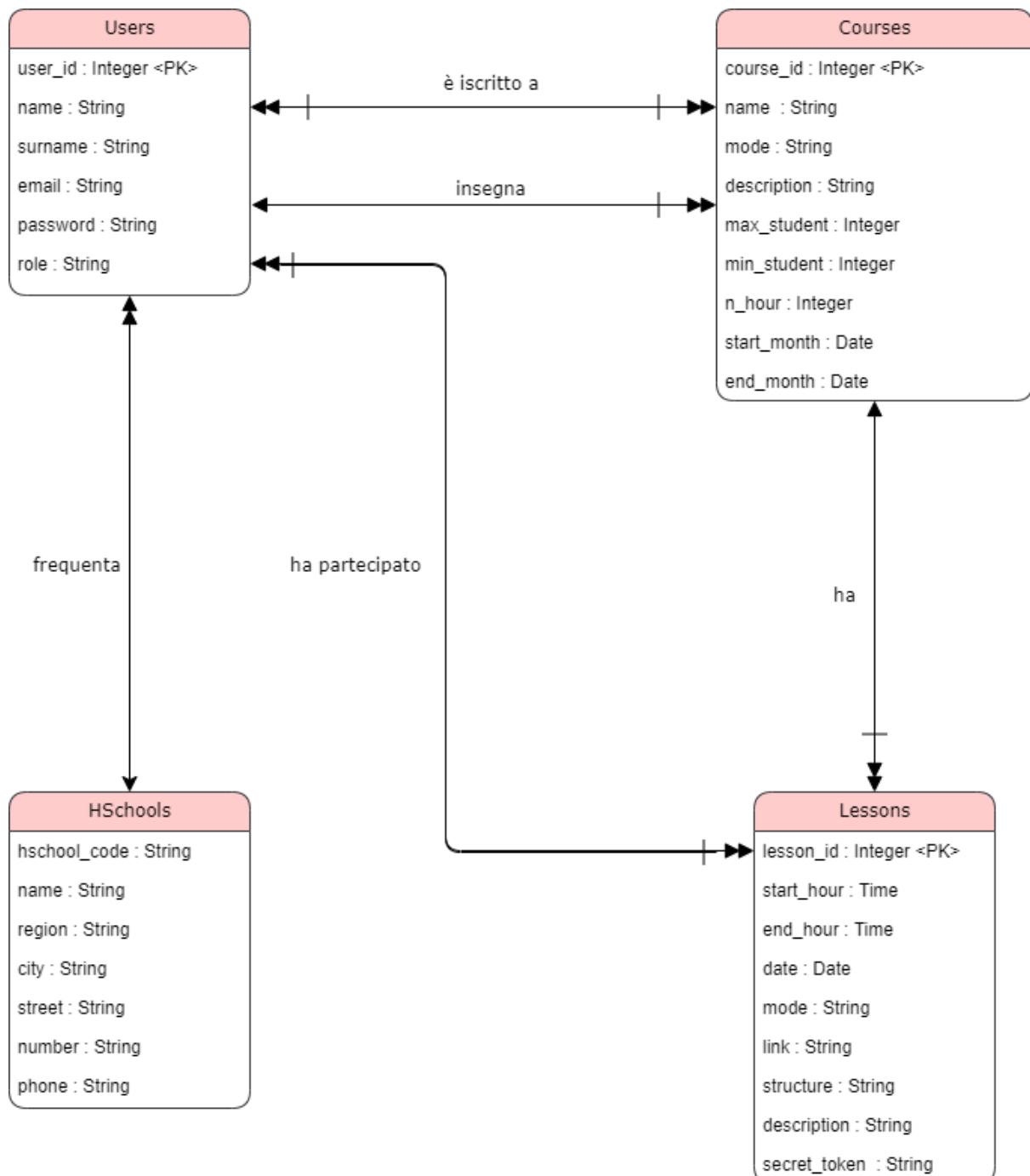
- hschool_code: identificativo univoco (codice meccanografico) dell'istituto superiore
- name: nome della scuola
- region: regione in cui si trova la scuola
- city: città in cui si trova la scuola
- street: via in cui si trova la scuola
- number: numero civico della via in cui si trova la scuola
- phone: numero di telefono della scuola

Lessons:

- lesson_id: identificativo univoco per ogni lezione
- start_hour: orario di inizio lezione
- end_hour: orario di fine lezione
- date: data in cui avrà luogo la lezione
- mode: modalità di erogazione della lezione (in presenza, online oppure blended)
- link: link zoom della lezione (necessario nel caso in cui la lezione sia online oppure in modalità blended)

- structure: edificio in cui avrà luogo la lezione (nel caso in cui la lezione sia in presenza oppure in modalità blended)
- description: breve descrizione della lezione. Qui vengono elencati gli argomenti che si andranno a svolgere durante la lezione
- secret_token: codice generato casualmente che è alla base del sistema di verifica delle partecipazioni

A questo punto è possibile definire la struttura finale del modello concettuale per la realtà considerata:



3.2 Modello logico

La progettazione logica della base di dati consiste nella traduzione dello schema concettuale dei dati in uno schema logico che rispecchia il modello dei dati scelto, come nel nostro caso, il modello relazionale.

Date le entità riportate e attraverso l'utilizzo del modello concettuale precedentemente descritto, riportiamo la loro rappresentazione finale nella seguente forma, comprendendo i vincoli di chiave primaria (PK), di chiave esterna (FK(nome entità)) e i vincoli not null (NOT NULL). Per questioni di leggibilità abbiamo omesso i vincoli NOT NULL sugli attributi che non corrispondono a chiavi esterne.

Avendo due relazioni molti a molti nel modello concettuale, troveremo nel modello relazionale due nuove entità:

- *Enrollments*

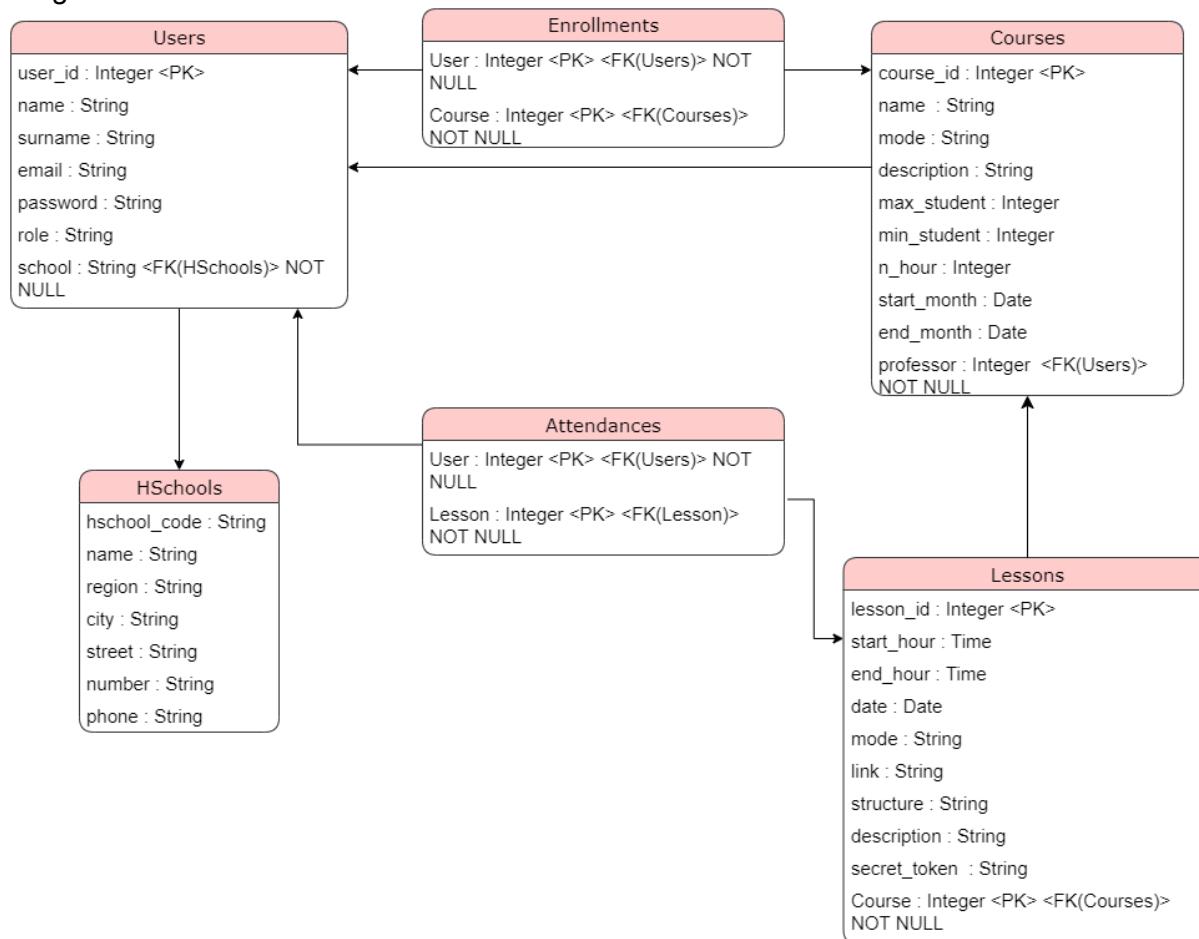
Questa entità rappresenta l'iscrizione di uno studente a un corso, infatti ha come attributi un codice identificativo per lo studente e un codice identificativo per il corso

- *Attendances*

Questa entità rappresenta la partecipazione di uno studente a una lezione, infatti ha come attributi un codice identificativo per lo studente e un codice identificativo per la lezione

Si noti che la relazione "Attendances" nel codice ha il nome "user_lesson" e la relazione "Enrollments" nel codice ha il nome "user_course".

La rappresentazione finale delle entità all'interno della base relazionale avrà quindi la seguente forma:



3.3 Definizione delle entità nella base di dati

Sono necessarie delle definizioni precise in base alla realtà che si vuole descrivere e per riuscire a mantenere una situazione di integrità dei dati all'interno della base relazionale. Quest'ultime sono così definite per le varie entità:

User

```
class User(UserMixin, db.Model):
    __tablename__ = "users"
    _user_id = db.Column(db.Integer, primary_key=True)
    _name = db.Column(db.String(64), unique=False, nullable=False)
    _surname = db.Column(db.String(64), unique=False, nullable=False)
    _email = db.Column(db.String(60), unique=True, nullable=False, index=True)
    _password = db.Column(db.String(128), nullable=False)

    # Campo per l'identificazione del ruolo dell'utente
    _role = db.Column(db.String(10), nullable=False, default="user")

    # Campo nato dalla relazione molti a molti con la tabella 'courses'
    _courses = db.relationship("Course", secondary=user_course, back_populates="_users")

    # Campo nato dalla relazione molti a molti con la tabella 'lessons'
    _lessons = db.relationship("Lesson", secondary=user_lesson, back_populates="_users")

    # Campo nato dalla relazione uno a molti con la tabella 'h_schools'
    _school = db.Column(db.String, db.ForeignKey('h_schools._hschool_code'))

    # Campo nato dalla relazione uno a molti con la tabella 'courses' (solo per gli utenti con ruolo '_professor')
    _courses_prof = db.relationship('Course', backref='users', cascade="all, delete")
```

Courses

```
class Course(UserMixin, db.Model):
    __tablename__ = "courses"
    _course_id = db.Column(db.Integer, primary_key=True)
    _name = db.Column(db.String(64), unique=True, nullable=False, index=True)
    _mode = db.Column(db.String(10), nullable=False)
    _description = db.Column(db.TEXT, nullable=False, default="La descrizione non è ancora disponibile!")
    _max_student = db.Column(db.Integer, CheckConstraint('max_student > min_student', name='max_stud_costr'), nullable=False)
    _min_student = db.Column(db.Integer, CheckConstraint('min_student < max_student', name='min_stud_costr'), nullable=False)
    _n_hour = db.Column(db.Integer, nullable=False)
    _start_date = db.Column(db.DATE, CheckConstraint('str(end_date) > str(start_date)', name='start_date_costr'), nullable=False)
    _end_date = db.Column(db.DATE, CheckConstraint('str(start_date) < str(end_date)', name='end_date_costr'), nullable=False)

    # Campo derivato dalla relazione molti a uno, dove uno si riferisce al corso
    _lessons = db.relationship('Lesson', backref='courses', cascade="all, delete")

    # Chiave esterna relativa al professore associato al corso
    _professor = db.Column(db.Integer, db.ForeignKey('users._user_id'))

    # Collegamento per la relazione molti a molti con utenti, quando si rimuove un corso questo collegamento è rimosso in automatico
    _users = db.relationship("User", secondary=user_course, back_populates="_courses")
```

HSchools

```
class Hschool(UserMixin, db.Model):
    __tablename__ = "h_schools"
    _hschool_code = db.Column(db.String(10), primary_key=True)
    _name = db.Column(db.String(64), nullable=False)
    _region = db.Column(db.String(20), nullable=False)
    _city = db.Column(db.String(40), nullable=False)
    _street = db.Column(db.String(64), nullable=False)
    _number = db.Column(db.String(6), nullable=False)
    _phone = db.Column(db.String(15), nullable=False)

    # Campo nato dalla relazione tra le scuole e gli utenti (con ruolo 'user')
    r_users = db.relationship('User', backref='h_schools')
```

Lessons

```
class Lesson(UserMixin, db.Model):
    __tablename__ = "lessons"
    _lesson_id = db.Column(db.Integer, primary_key=True)
    _start_hour = db.Column(db.TIME, nullable=False)
    _end_hour = db.Column(db.TIME, nullable=False)
    _date = db.Column(db.DATE, nullable=False)
    _mode = db.Column(db.String(10), nullable=False)
    _link = db.Column(db.String(2083), nullable=False)
    _structure = db.Column(db.String(64), nullable=True)
    _description = db.Column(db.TEXT, nullable=False, default="Descrizione non ancora disponibile")
    _secret_token = db.Column(db.String(32), nullable=False)

    # Chiave esterna derivata dalla relazione uno a molti con lezioni
    course = db.Column(db.Integer, db.ForeignKey('courses._course_id'))

    # Collegamento nato dalla relazione molti a molti con gli utenti (con ruolo 'user')
    _users = db.relationship("User", secondary=user_lesson, back_populates="_lessons")
```

4. Funzionalità principali e visualizzazione schermate

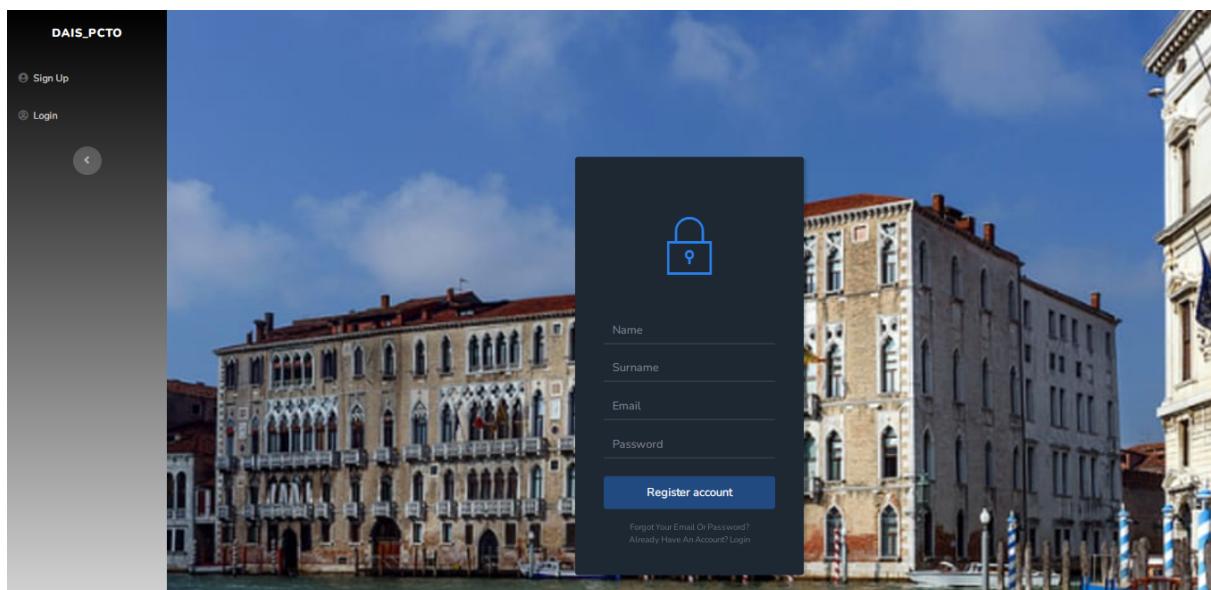
4.1 Schermata iniziale

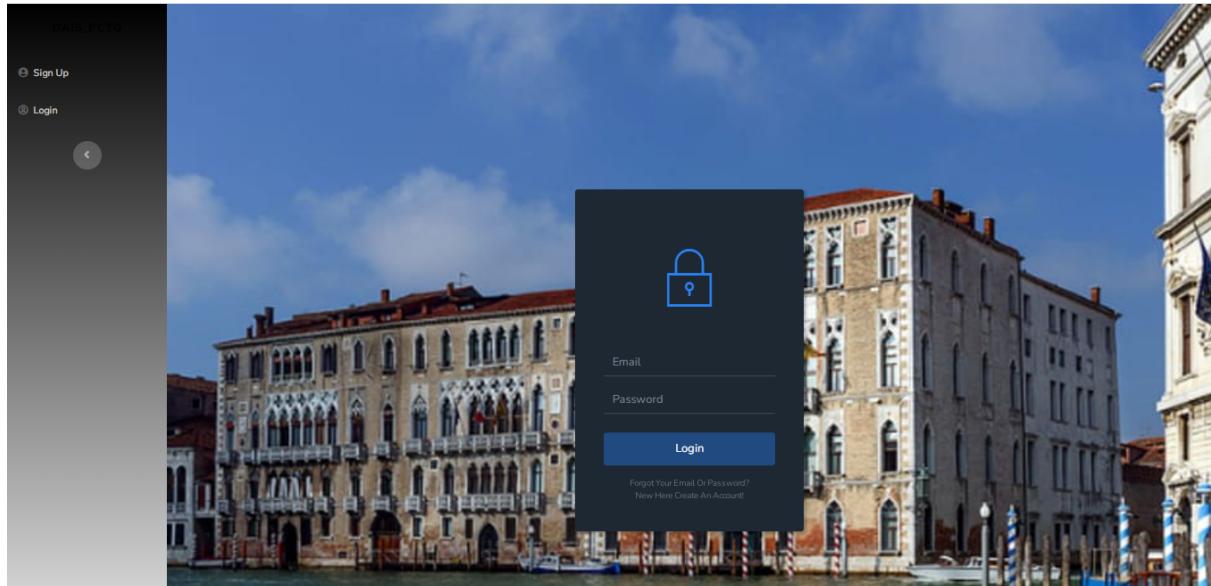


4.2 Registrazione e login

Il ruolo che un utente ha all'interno della web application è assegnato automaticamente in base al dominio dell'indirizzo email fornito dall'utente in fase di iscrizione:

- gli indirizzi email che terminano “segunive.it” appartengono a utenti che lavorano come componenti della segreteria del campus
- gli indirizzi email che terminano “unive.it” appartengono a utenti che lavorano come docenti
- tutti gli altri indirizzi email appartengono a studenti





4.3 Visualizzazione schermata studente (autenticato)

La schermata a cui ogni utente è rimandato immediatamente dopo aver completato il processo di autenticazione è quella del suo profilo, dove è possibile visualizzare e, volendo, modificare i propri dati personali. Gli studenti, inoltre, possono visualizzare nella stessa pagina una lista dei corsi a cui sono iscritti che all'inizio sarà ovviamente vuota.

Nome	Modalità	Lezioni Con Presenza Registrata	Inizio Lezioni	Fine Lezioni	Pagina Corso
No Data Available In Table					

Proprio per questo motivo nella navbar laterale lo studente può scegliere di visitare la pagina "Lista Corsi" in cui troverà una lista di tutti i corsi erogati dal dipartimento, cliccando sul simbolo nella colonna "Pagina Corso" per ogni voce della tabella sarà possibile visitare le pagine dei singoli corsi.

All'interno di quest'ultime, uno studente potrà verificare se il corso è aperto oppure chiuso e, nel primo caso, se desidererà farlo potrà iscriversi al corso cliccando sull'apposito pulsante.

Sempre nella pagina del corso, lo studente potrà visualizzare la lista delle lezioni in ordine cronologico

e per ognuno potrà visualizzare le informazioni relative ad ogni lezione, data e orari, una sezione "Extra" dove si troveranno eventuali materiali relativi alla lezione (pdf, slides, etc.) e

infine la sezione “Token Lezioni” che permette allo studente di inserire il token della lezione (se lo studente ha partecipato alla lezione allora avrà ricevuto il token da parte del professore che ad ogni lezione ha il compito di divulgare il token della lezione) e di registrare la propria presenza alla lezione ai fini di ottenere il certificato di partecipazione al corso una volta che lo stesso sarà terminato.

The first screenshot shows a general view of a course section titled "Lezione N-1". It includes tabs for "Info", "Date E Orari", "Extra", and "Token Lezioni". Below the tabs, it displays "Modalità Della Lezione: Presenza." and "Descrizione: Elisir d'Amore.", along with a link labeled "Link Lezione".

The second screenshot shows the "Date E Orari" tab selected. It displays the date "Data : 2022-07-02" and the time range "Ora Inizio : 09:00:00 Fine : 10:00:00".

The third screenshot shows the "Token Lezioni" tab selected. It has a text input field labeled "Inserisci token" and a red button labeled "Registra la tua presenza".

In un qualunque momento, lo studente può decidere di disiscriversi dal corso cliccando sull'apposito tasto. Si noti che la disiscrizione dal corso elimina la relazione tra studente e corso ma non le relazioni tra studente e le varie lezioni a cui ha partecipato, questo perché se lo studente si è disiscritto per sbaglio in questo modo non perderà le presenza certificate in precedenza.

4.4 Visualizzazione schermata professore (autenticato)

I professori, così come gli studenti, dopo l'autenticazione sono rimandati alla pagina del loro profilo, dove è possibile visualizzare e, volendo, modificare i propri dati personali. I professori, inoltre, possono visualizzare nella stessa pagina una lista dei corsi da loro creati che all'inizio sarà ovviamente vuota.

DAIS_PCTO

Severus Snape

INFORMAZIONI DEL PROFILO

Username: 5 Email: severus.snape@unive.it

Nome: Severus Cognome: Snape

Modifica i tuoi dati

LISTA CORSI CREATI

Nome	Modalità	Numero Studenti Iscritti	Inizio Lezioni	Fine Lezioni	Pagina Corso
Posizioni	Presenza	1	2022-07-02	2022-08-03	1

Showing 1 To 1 Of 1 Entries

Search:

Previous 1 Next

Proprio per questo motivo nella navbar laterale il professore può scegliere di visitare la pagina “Crea Corso” all’interno della quale si trova un form con vari campi vuoti che il professore deve riempire in maniera opportuna.

DAIS_PCTO

Severus Snape

AGGIUNGI CORSO

Nome

presenza

Descrizione

Numero massimo studenti

Numero minimo studenti

ore corso totali

gg/mm/aaaa

gg/mm/aaaa

Aggiungi

Anche il professore può visualizzare la pagina “Lista Corsi” che è uguale a quella visitabile dagli studenti. Nelle singole pagine dei corsi, un professore può vedere le informazioni generali del corso, inclusa la barra di progressione del corso che mostra la percentuale di ore di lezione sono già state svolte.

Sono disponibili tre tasti:

- “Aggiungi lezione”, cliccando sul quale si apre un modale con un form da compilare con le informazioni necessarie per creare una lezione
- “Modifica Corso”, cliccando sul quale si apre un modale con un form con alcuni dei campi del form per creare il corso, ovvero i campi di un corso modificabili
- “Rimuovi Corso”, cliccando sul quale viene chiesto di inserire la password per confermare l’azione e dopodichè il corso è eliminato

The screenshot shows a user profile sidebar on the left with options: Profilo, Lista Corsi, Crea Corso, and Logout. The main content area displays course details for 'Arte di fabbricare pozioni'. It includes a progress bar at 5%, a table with course metadata, and a description section. At the bottom are buttons for Adding a Lesson, Modifying the Course, and Removing the Course.

Informazioni Del Corso		
Percentuale Progresso Corso 5 %		
Nome Pozioni	Modalità Presenza	Num Massimo Studenti 50
Num Minimo Studenti 20	Data Inizio 2022-06-30	Data Fine 2022-08-05
Prof Referente Severus Snape	Email Prof severus.snape@unive.it	Stato Corso Corso Aperto!
Descrizione Arte di fabbricare pozioni		

Aggiungi Lezione **Modifica Corso** **Rimuovi Corso**

Sempre nella pagina del corso, proprio come lo studente anche il professore potrà visualizzare la lista delle lezioni in ordine cronologico e per ognuno potrà visualizzare le informazioni relative ad ogni lezione, data e orari, una sezione “Extra” che contiene due tasti, “Modifica Lezione” (cliccando sul quale si apre il form di modifica lezione) e “Elimina Lezione” cliccando sul quale viene chiesto di inserire la password per confermare l’azione e dopodichè la lezione è eliminata, e infine la sezione “Token Lezioni” dove il professore trova il Token Segreto generato automaticamente dalla web application che avrà il compito di divulgare alla fine o all’inizio di ogni lezione per permettere agli studenti che hanno partecipato alla stessa di registrare la propria presenza ai fini di ottenere il certificato di partecipazione al corso una volta che lo stesso sarà terminato.

The first screenshot shows the 'Lezione N-1' page with tabs for Info, Date E Orari, Extra, and Token Lezioni. The Token Lezioni tab is active, displaying the text "Token Segreto Per Prof:" followed by a long hex string: B2249ed65f88d4294b4f24e9caa11771.

The second screenshot shows the same page with the Extra tab active. It includes buttons for 'Modifica Lezione' (blue) and 'Rimuovi Lezione' (red).

4.5 Visualizzazione schermata segreteria (autenticato)

I componenti della segreteria, così come professori e studenti, dopo l'autenticazione sono rimandati alla pagina del loro profilo, dove è possibile visualizzare e, volendo, modificare i dati. Nella navbar laterale la segreteria può scegliere di visitare la pagina “Crea Corso” all'interno della quale si trova un form con vari campi vuoti (tra cui il campo “Professore” in cui va inserita la mail del professore) da riempire in maniera opportuna.

Anche la segreteria può visualizzare la pagina “Lista Corsi” che è uguale a quella visitabile dagli studenti e dai professori. Nelle singole pagine dei corsi, un componenete della segreteria può vedere le informazioni generali del corso. Sono disponibili tre tasti:

- “Aggiungi lezione”, cliccando sul quale si apre un modale con un form da compilare con le informazioni necessarie per creare una lezione
- “Modifica Corso”, cliccando sul quale si apre un modale con un form con alcuni dei campi del form per creare il corso, ovvero i campi di un corso modificabili
- “Rimuovi Corso”, cliccando sul quale viene chiesto di inserire la password per confermare l’azione e dopodichè il corso è eliminato

Sempre nella pagina del corso, proprio come lo studente anche il professore potrà visualizzare la lista delle lezioni in ordine cronologico e per ognuno potrà visualizzare le informazioni relative ad ogni lezione, data e orari e infine una sezione “Extra” che contiene due tasti, “Modifica Lezione” (cliccando sul quale si apre il form di modifica lezione) e “Elimina Lezione” cliccando sul quale viene chiesto di inserire la password per confermare l’azione e dopodichè la lezione è eliminata.

Sempre nella navbar laterale la segreteria può scegliere di visitare la pagina “Gestisci Scuole” dove può visualizzare la lista di scuole superiori già create all’interno della web app, che inizialmente sarà vuota, e dove si trova il tasto “Aggiungi scuola”, cliccando sul quale si apre un modale contenente un form da compilare opportunatamente con le informazioni richeiste relative a un istituto superiore. Nella colonna “Rimuovi Scuola” per ogni voce della tabella si trova un tasto “Elimina” cliccando sul quale si esegue la cancellazione immediata dell’istanza dalla web application.

I componenti della segreteria possono anche navigare nella pagina “Gestisci Utenti” dove si trova una lista di tutti gli studenti registrati all’interno della web app e per ognuno, nella colonna “Modifica Utente”, si può selezionare la scuola di appartenenza semplicemente cliccando sul tasto “Modifica”

5. Scelte progettuali

Sono necessarie delle scelte progettuali utili per poter delineare al meglio tutte le modalità per implementare nella maniera più efficiente e pulita tutto il codice. Per questo motivo abbiamo deciso di implementare dei vincoli, trigger, check su attributi e ruoli nella base di dati per gestire le autorizzazioni degli utenti.

5.1 Vincoli

NOT NULL:

Vi sono righe di tabelle che sono accompagnate dal vincolo Not Null. Infatti quelle righe non ammettono che non venga inserito alcun valore.

PRIMARY KEY:

Tutte le tabelle presentano un codice identificativo come primary key in grado di identificare univocamente ogni entry inserita nella base di dati.

FOREIGN KEY:

Il vincolo di Foreign Key è facile da posizionare per poter ottenere relazioni tra più tabelle collegandole tra di loro.

UNIQUE:

Il vincolo Unique ci torna utile per garantire che non vengano immessi valori duplicati in colonne specifiche che non fanno parte di una chiave primaria.

Nella tabella User c'è un solo vincolo (unique=True) applicato al campo "email" con lo scopo di avere un'unica email per ogni utente.

```
_email = db.Column(db.String(60), unique=True, nullable=False, index=True)
```

Nella tabella Course c'è un solo vincolo (unique=True) applicato al campo "name" con lo scopo di avere un nome diverso per ogni corso

```
_name = db.Column(db.String(64), unique=True, nullable=False, index=True)
```

5.2 Check e trigger

Si è deciso di imporre dei vincoli e delle proprietà sull'applicazione web in modo da renderla più affidabile e reale. I controlli che ne seguono vengono applicati non solo sul codice JavaScript ma anche a livello di DB, il livello più basso possibile. Per realizzarli si sono implementati diversi check e trigger.

CHECK SU 'lessons'

```
ALTER TABLE lessons
ADD CONSTRAINT check_hour_lessons
CHECK(_start_hour >= '09:00' AND _end_hour <= '20:00'
      AND _end_hour > _start_hour);
```

- ❖ Questo controllo deve essere rispettato da ogni tupla e dato che coinvolge attributi appartenenti alla sola tabella 'lessons' è possibile fare un check. Tale check verifica che l'orario delle lezioni: l'ora di inizio non deve essere prima delle '09:00'; l'ora di fine non deve essere dopo le '20:00'; l'ora di fine della lezione deve essere posteriore all'ora di inizio di essa.

```
ALTER TABLE lessons
ADD CONSTRAINT check_mode_lessons
CHECK(_mode = 'presenza' OR _mode = 'online' OR _mode = 'blended');
```

- ❖ Questo controllo coinvolge un solo attributo senza far riferimento a particolari condizioni. Per questo motivo è possibile eseguire un check: il

‘check_mode_lessons’ verifica che i valori che possono essere assunti dall’attributo ‘_mode’ siano solo 3: presenza, online o blended.

CHECK SU ‘users’

```
ALTER TABLE users
ADD CONSTRAINT check_role_users
CHECK (_role = 'user' OR _role = 'professor' OR _role = 'admin');
```

- ❖ Questo controllo coinvolge un solo attributo senza far riferimento a particolari condizioni. Per questo motivo è possibile eseguire un check: il ‘check_role_users’ verifica che i valori che possono essere assunti dall’attributo ‘_role’ siano solo 3: user, professor o admin.

```
ALTER TABLE users
ADD CONSTRAINT check_role_professor_email_users
CHECK (NOT(_role = 'professor') OR (_email LIKE '%unive.it'));
```

```
ALTER TABLE users
ADD CONSTRAINT check_role_admin_email_users
CHECK (NOT(_role = 'admin') OR (_email LIKE '%segunive.it'));
```

- ❖ Nonostante il controllo sia su più attributi è possibile comunque fare un check dato che non sono coinvolte altre tabelle oltre a ‘users’. Il check prevede che gli utenti con ruolo ‘professor’ debbano iscriversi con una mail che termina in ‘unive.it’, mentre gli utenti che svolgono il ruolo di ‘segreteria’ debbano iscriversi con una mail ‘segunive.it’. Non sono stati fatti ulteriori controlli sul campo ‘_email’, come ad esempio verificare che lo studente abbia una email differente da quella di uno studente universitario perché, dal nostro punto di vista, è un errore dello studente universitario iscriversi a corsi a lui non dedicati.

CHECK SU ‘courses’

```
ALTER TABLE courses
ADD CONSTRAINT check_date_courses
CHECK(_start_date < _end_date);
```

- ❖ Questo controllo verifica che la data di fine corso sia posteriore alla relativa data di inizio. Questa proprietà deve essere sempre rispettata da ogni singola tupla della tabella.

```

ALTER TABLE courses
ADD CONSTRAINT check_mode_courses
CHECK(_mode = 'presenza' OR _mode = 'blended' OR _mode = 'online');

```

- ❖ Questo controllo coinvolge un solo attributo senza far riferimento a particolari condizioni. Per questo motivo è possibile eseguire un check: il ‘check_mode_courses’ verifica che i valori che possono essere assunti dall’attributo ‘_mode’ siano solo 3: presenza, online o blended.

```

ALTER TABLE courses
ADD CONSTRAINT check_num_student_courses
CHECK(_max_student > _min_student AND _min_student >= '10');

```

- ❖ Il ‘check_num_student_courses’ verifica le seguenti condizioni:
 - il numero di studenti minimo deve essere superiore a 10
 - il numero massimo di studenti deve essere superiore del numero minimo di studenti previsti per quel corso

TRIGGERS

I seguenti trigger possono essere scatenati da operazioni che possono essere eseguite da un qualsiasi utente con un qualsiasi ruolo in base ai privilegi di cui gode.

TRIGGER SU ‘user_corse’ e ‘user_lesson’

```

CREATE FUNCTION only_user() RETURNS trigger AS $$ 
BEGIN
IF(NEW.user_id NOT IN
(SELECT _user_id FROM users WHERE _role = 'user')) THEN
    RETURN NULL;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER OnlyUserCourse
BEFORE INSERT ON user_course
FOR EACH ROW
EXECUTE FUNCTION only_user();

CREATE TRIGGER OnlyUserLesson
BEFORE INSERT ON user_lesson
FOR EACH ROW
EXECUTE FUNCTION only_user();

```

- ❖ La funzione “only_user” viene richiamata da due trigger: OnlyUserCourse e OnlyUserLesson. Questi trigger servono per garantire che solo gli utenti con ruolo ‘user’ si iscrivano a corsi e lezioni. I trigger sono stati implementati come BEFORE sulla sola operazione di INSERT. Non è possibile avere una violazione dall’operazione UPDATE visto che se uno studente sbaglia a iscriversi a un corso o a una lezione, si disiscrive e riscrive su ciò che desidera.

TRIGGER SU ‘user_course’

```
CREATE FUNCTION no_over_max() RETURNS trigger AS $$  
BEGIN  
    IF((SELECT _max_student  
        FROM courses  
        WHERE _course_id = NEW.course_id)  
    <=  
    (SELECT count(*)  
        FROM user_course  
        WHERE course_id = NEW.course_id)) THEN  
        RETURN NULL;  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER NoOverMax  
BEFORE INSERT ON user_course  
FOR EACH ROW  
EXECUTE FUNCTION no_over_max();
```

- ❖ Il trigger “NoOverMax” impedisce che si iscrivano ai corsi più persone del numero massimo dichiarato dal corso. Si è utilizzato un trigger e non un check perchè serve la tabella ‘courses’ per ricavare l’attributo ‘max_student’ e la tabella ‘user_course’ per contare il numero di studenti già iscritti. È un BEFORE trigger sull’operazione di INSERT. (Non è possibile fare l’UPDATE: se uno studente si iscrive a un corso e poi cambia idea allora si disiscrive e ri-iscrive in quello che desidera).

TRIGGER SU ‘user_lesson’

```
CREATE FUNCTION no_user_in_course() RETURNS trigger AS $$  
BEGIN  
    IF( (SELECT course  
        FROM lessons  
        WHERE _lesson_id = NEW.lesson_id)  
    NOT IN  
    (SELECT course_id  
        FROM user_course  
        WHERE user_id = NEW.user_id)) THEN  
        RETURN NULL;  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER NoUserInCourse  
BEFORE INSERT ON user_lesson  
FOR EACH ROW  
EXECUTE FUNCTION no_user_in_course();
```

- ❖ Il trigger “NoUserInCourse” impedisce che un utente si iscriva a lezioni di corsi che non frequenta. Sono necessarie due tabelle: ‘lessons’ per ricavare il corso a cui la lezione è associata e ‘user_course’ per ricavare i corsi a cui l’utente è iscritto; se il corso della lezione a cui l’utente intende iscriversi non è presente nella lista di corsi a cui l’utente si è iscritto, allora l’operazione viene vietata. Il trigger è implementato come un BEFORE trigger su INSERT. (Non è possibile l’UPDATE dato che se un utente ha sbagliato a iscriversi a una lezione allora si disiscrive da essa e si iscrive a quella desiderata).

TRIGGERS SU ‘lessons’

```

CREATE FUNCTION no_lesson_over() RETURNS trigger AS $$ 
BEGIN
    IF((SELECT sum(l._end_hour-l._start_hour)
        FROM lessons as l
        WHERE l.course = NEW.course)
    >
    (SELECT make_interval(hours => co._n_hour)
        FROM courses as co
        WHERE co._course_id = NEW.course)) THEN
        DELETE FROM lessons WHERE _lesson_id = NEW._lesson_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER NoLessonOver
AFTER INSERT ON lessons
FOR EACH ROW
EXECUTE FUNCTION no_lesson_over();

```

```

CREATE FUNCTION no_lesson_over_m() RETURNS trigger AS $$ 
BEGIN
    IF((SELECT sum(l._end_hour-l._start_hour)
        FROM lessons as l
        WHERE l.course = NEW.course)
    >
    (SELECT make_interval(hours => co._n_hour)
        FROM courses as co
        WHERE co._course_id = NEW.course)) THEN
        IF(NEW._start_hour <> OLD._start_hour) THEN
            UPDATE lessons
            SET _start_hour = OLD._start_hour
            WHERE _lesson_id = NEW._lesson_id;
        END IF;
        IF(NEW._end_hour <> OLD._end_hour) THEN
            UPDATE lessons
            SET _end_hour = OLD._end_hour
            WHERE _lesson_id = NEW._lesson_id;
        END IF;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER NoLessonOverM
AFTER UPDATE ON lessons
FOR EACH ROW
EXECUTE FUNCTION no_lesson_over_m();

```

- ❖ Per far in modo che non vengano registrate più ore di lezione rispetto a quelle dichiarate dal corso a cui sono associate sono stati implementati 2 trigger: uno sull'operazione di INSERT e uno sull'operazione di UPDATE.
 - Il trigger sull'operazione di INSERT è un AFTER trigger e controlla se la somma delle ore delle lezioni è maggiore in senso stretto del totale di ore dichiarato dal corso; se così fosse, allora vengono eliminate le nuove tuple.
 - Analogamente il trigger sull'operazione di UPDATE, ma al posto di eliminare le tuple coinvolte vengono assegnati i valori precedenti agli attributi '_start_hour' e '_end_hour' eseguendo un controllo su quale attributo fosse stato modificato in modo "anomalo".
- ❖ Entrambi i trigger sono AFTER trigger visto che risulta necessario valutare la proprietà dopo che la tupla è stata inserita o aggiornata per motivi di espressività del database.

```

CREATE FUNCTION correct_dates() RETURNS trigger AS $$ 
BEGIN
    IF(NEW._date < (SELECT _start_date FROM courses WHERE NEW.course = _course_id) OR
       NEW._date > (SELECT _end_date FROM courses WHERE NEW.course = _course_id) OR
       NEW._date <= current_date) THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER CorrectDates
BEFORE INSERT ON lessons
FOR EACH ROW
EXECUTE FUNCTION correct_dates();

```

- ❖ Il trigger “CorrectDates” opera sulla tabella ‘lessons’ e controlla che la data che si vuole inserire sia compresa tra la data di inizio e la data di fine del corso a cui viene associata e che sia necessariamente dopo il giorno in cui questa viene inserita. Tale trigger è un BEFORE trigger dato che tutte le informazioni necessarie per controllare la proprietà si hanno a priori e viene eseguito sull’operazione di INSERT.

```

CREATE FUNCTION no_lessons_overlying() RETURNS trigger AS $$ 
BEGIN
    IF(NEW._date IN (SELECT _date
                      FROM lessons as l
                     WHERE l.course = NEW.course AND
                           ((NEW._start_hour <= l._start_hour
                            AND NEW._end_hour <= l._end_hour AND NEW._end_hour >= l._start_hour) OR
                            (NEW._start_hour >= l._start_hour AND NEW._start_hour <= l._end_hour
                             AND NEW._end_hour >= l._end_hour) OR
                            (NEW._start_hour >= l._start_hour AND NEW._end_hour <= l._end_hour) OR
                            (NEW._start_hour <= l._start_hour AND NEW._end_hour >= l._end_hour))) THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER NoLessonsOverlying
BEFORE INSERT ON lessons
FOR EACH ROW
EXECUTE FUNCTION no_lessons_overlying();

```

- ❖ È necessario che le lezioni di un corso non si accavallino tra loro, ovvero che non siano svolte nella stessa fascia oraria. Per rendere possibile questo è necessario che nessuna lezione avvenga in concomitanza con un’altra lezione del medesimo corso. Il trigger “NoLessonsOverlying” evita questo. È un BEFORE trigger su INSERT e controlla se ci sono lezioni con la stessa data di quella che si vuole inserire che sono associate allo stesso corso. Se le condizioni precedenti risultano vere si controllano gli orari. Se anche solo una delle seguenti situazioni è vera allora l’inserimento viene annullato:
(si vuole inserire una lezione denominata ‘lezione1’ e viene confrontata con ‘lezione2’ entrambe associate allo stesso corso)

- (l'ora di inizio di lezione1 è prima dell'ora di inizio di lezione2) && (l'ora di fine di lezione1 è compresa tra l'ora di inizio di lezione2 e l'ora di fine di lezione2)
- (l'ora di inizio di lezione1 è compresa tra l'ora di inizio di lezione2 e l'ora di fine di lezione2) && (l'ora di fine di lezione1 è dopo l'ora di fine di lezione2)
- (l'ora di inizio di lezione1 è dopo l'ora d'inizio di lezione2) && (l'ora di fine di lezione1 è prima dell'ora di fine di lezione2)
- (l'ora di inizio di lezione1 è prima dell'ora di inizio di lezione2) && (l'ora di fine di lezione1 è dopo l'ora di fine di lezione2)

```

CREATE FUNCTION limit_hours() RETURNS trigger AS $$ 
BEGIN
    IF(NEW._end_hour - NEW._start_hour > '06:00') THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER LimitHours
BEFORE INSERT OR UPDATE ON lessons
FOR EACH ROW
EXECUTE FUNCTION limit_hours();

```

- ❖ Si è deciso che le lezioni non possono durare più di 6 ore. Il trigger “LimitHours” verifica l’effettiva durata della lezione. È un BEFORE trigger e le operazioni che possono violare il vincolo sono la INSERT e la UPDATE.

```

CREATE FUNCTION limit_min_hours() RETURNS trigger AS $$ 
BEGIN
    IF(NEW._end_hour - NEW._start_hour < '01:00') THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER LimitMinHours
BEFORE INSERT OR UPDATE ON lessons
FOR EACH ROW
EXECUTE FUNCTION limit_min_hours();

```

- ❖ Oltre a un limite massimo, si è deciso di imporre un limite minimo sulla durata di una singola lezione: una lezione deve durare almeno un’ora. Il trigger “LimitMinHour” è stato implementato per questo motivo ed è un BEFORE trigger sulle operazioni di INSERT e UPDATE.

```

CREATE FUNCTION lesson_update() RETURNS trigger AS $$

BEGIN
    IF((NEW._mode <> 'presenza' OR NEW._link <> '' OR NEW._structure = '') AND
        NEW.course IN (SELECT _course_id FROM courses WHERE _mode = 'presenza'))
    OR
        ((NEW._mode <> 'online' OR NEW._link = '' OR NEW._structure <> '') AND
        NEW.course IN (SELECT _course_id FROM courses WHERE _mode = 'online'))
    OR
        ((NEW._link = '' OR NEW._structure = '') AND
        NEW.course IN (SELECT _course_id FROM courses WHERE _mode = 'blended'))
    OR
        OLD._date <= current_date
    OR
        NEW._date <= current_date
    OR
        NEW._date < (SELECT _start_date
                      FROM courses
                      WHERE _course_id = NEW.course)
    OR
        NEW._date > (SELECT _end_date
                      FROM courses
                      WHERE _course_id = NEW.course)
    OR
        NEW._date IN (SELECT _date
                      FROM lessons AS l
                      WHERE NEW._lesson_id <> l._lesson_id AND
                            NEW.course = l.course AND
                            ((NEW._start_hour <= l._start_hour AND NEW._end_hour <= l._end_hour
                            AND NEW._end_hour >= l._start_hour) OR
                            (NEW._start_hour >= l._start_hour AND NEW._start_hour <= l._end_hour
                            AND NEW._end_hour >= l._end_hour) OR
                            (NEW._start_hour >= l._start_hour AND NEW._end_hour <= l._end_hour) OR
                            (NEW._start_hour <= l._start_hour AND NEW._end_hour >= l._end_hour))) THEN
            RETURN NULL;
        END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER LessonUpdate
BEFORE UPDATE ON lessons
FOR EACH ROW
EXECUTE FUNCTION lesson_update();

```

- ❖ Il trigger “LessonUpdate” è un BEFORE trigger sull’operazione UPDATE che richiama la funzione "lesson_update()" che esegue diversi controlli al suo interno: se anche una soltanto delle condizioni è vera allora la modifica della lezione non è consentita.
- ❖ Le condizioni per cui l’eventuale modifica viene bloccata sono le seguenti:
 - Se il corso relativo alla lezione interessata si svolge in presenza e se
 - o l’attributo ‘_mode’ è diverso da ‘presenza’
 - e/o l’attributo ‘_link’ non è vuoto
 - e/o l’attributo ‘_structure’ è vuoto;

- Se il corso relativo alla lezione interessata si svolge online e se
 - o l'attributo '_mode' è diverso da 'online'
 - e/o l'attributo '_link' è vuoto
 - e/o l'attributo '_structure' non è vuoto;
- Se il corso relativo alla lezione interessata si svolge in modalità blended e se
 - o l'attributo '_link' è vuoto
 - e/o l'attributo '_structure' è vuoto;

(Si è deciso che una lezione di un corso blended può assumere una qualsiasi modalità, l'importante è che siano sempre presenti il link e la struttura)
- Se la lezione è già stata effettuata;
- Se la nuova data inserita precede la data di oggi;
- Se la nuova data inserita non è compresa nel periodo indicato dal corso;
- Se la nuova data e i nuovi orari vanno a sovrapporsi ad altre lezioni già presenti nel database relative al medesimo corso della lezione che si vuole aggiornare.

```

CREATE FUNCTION lesson_mode() RETURNS trigger AS $$

BEGIN
  IF((NEW._mode <> 'presenza' OR NEW._link <> '' OR NEW._structure = '') AND
    NEW.course IN (SELECT _course_id FROM courses WHERE _mode = 'presenza'))
  OR
  ((NEW._mode <> 'online' OR NEW._link = '' OR NEW._structure <> '') AND
    NEW.course IN (SELECT _course_id FROM courses WHERE _mode = 'online'))
  OR
  ((NEW._link = '' OR NEW._structure = '') AND
    NEW.course IN (SELECT _course_id FROM courses WHERE _mode = 'blended')))THEN
    RETURN NULL;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER LessonMode
BEFORE INSERT ON lessons
FOR EACH ROW
EXECUTE FUNCTION lesson_mode();

```

- ❖ Il trigger 'LessonMode' è un BEFORE trigger sull'operazione di INSERT e controlla che in base alla modalità del corso a cui la lezione è associata ci sia una corretta corrispondenza con i campi della lezione.
 - Se la lezione è associata a un corso che si tiene in presenza è necessario che la lezione sia in presenza, che l'attributo '_link' sia vuoto e che l'attributo '_structure' contenga un valore;
 - Se la lezione è associata a un corso che si tiene online è necessario che la lezione sia online, che l'attributo '_link' contenga un valore e che l'attributo '_structure' sia vuoto;
 - Se la lezione è associata a un corso che viene svolto in modalità blended è necessario che ci sia sempre un valore per l'attributo '_link' e per l'attributo '_structure'. (La modalità della lezione può essere una qualsiasi tra le 3 possibili).

TRIGGERS SU ‘courses’

```
CREATE FUNCTION only_professor() RETURNS trigger AS $$  
BEGIN  
    IF(NEW._professor NOT IN (SELECT _user_id FROM users WHERE _role = 'professor')) THEN  
        RETURN NULL;  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER OnlyProfessor  
BEFORE INSERT OR UPDATE ON courses  
FOR EACH ROW  
EXECUTE FUNCTION only_professor();
```

- ❖ Nel campo ‘professor’ in ‘courses’ l’id associato può essere solo ed esclusivamente quello di un professore. Il trigger “OnlyProfessor” verifica questa proprietà. Le operazioni che possono violare il vincolo sono la INSERT e l’UPDATE (Quest’ultima operazione è riservata agli utenti con ruolo ‘admin’, ovvero è possibile che un professore non possa più tenere un corso e che di conseguenza la segreteria ne assegni uno successivamente).

```
CREATE FUNCTION max_student_sup() RETURNS trigger AS $$  
BEGIN  
    IF(NEW._max_student < OLD._max_student) THEN  
        RETURN NULL;  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER MaxStudentSup  
BEFORE UPDATE ON courses  
FOR EACH ROW  
EXECUTE FUNCTION max_student_sup();
```

- ❖ Tra i vincoli da noi imposti vi è quello che riguarda il massimo numero di studenti ammessi ad un corso. Si è scelto che questo numero può solo aumentare e non diminuire per non andare a scapito degli utenti che si sono iscritti. Di conseguenza viene implementato come un BEFORE trigger sull’operazione di UPDATE.

```

CREATE FUNCTION n_hour_sup() RETURNS trigger AS $$ 
BEGIN
    IF(NEW._n_hour < OLD._n_hour) THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER NHourSup
BEFORE UPDATE ON courses
FOR EACH ROW
EXECUTE FUNCTION n_hour_sup();

```

- ❖ Il vincolo per il numero delle ore è analogo a quello del numero massimo di studenti. Anche in questo caso si è deciso che è possibile solo l'aumento del numero delle ore del corso. Come il “MaxStudentSup”, anche “NHourSup” è un BEFORE trigger sulla sola operazione di UPDATE.

```

CREATE FUNCTION no_modify_mode_course() RETURNS trigger AS $$ 
BEGIN
    IF(OLD._mode <> NEW._mode) THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER NoModifyModeCourse
BEFORE UPDATE ON courses
FOR EACH ROW
EXECUTE FUNCTION no_modify_mode_course();

```

- ❖ Un altro vincolo da noi è scelto è il non poter modificare la modalità di un corso quando questo viene creato. Questo trigger serve proprio a controllare che la modalità rimanga sempre la stessa. È un BEFORE trigger sull'operazione di UPDATE perchè è l'unica operazione che potrebbe violare questa proprietà.

```

CREATE FUNCTION insert_dates() RETURNS trigger AS $$ 
BEGIN
    IF(NEW._start_date <= current_date)THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER InsertDates
BEFORE INSERT ON courses
FOR EACH ROW
EXECUTE FUNCTION insert_dates();

```

- ❖ “InsertDates” è un trigger che controlla che la data di inizio del corso sia successiva alla data odierna nel momento in cui il corso viene creato. È un BEFORE trigger su INSERT.

```

CREATE FUNCTION modify_start_date_update() RETURNS trigger AS $$ 
BEGIN
    IF(OLD._start_date <= current_date
    OR NEW._start_date <= current_date
    OR NEW._start_date > (SELECT min(_date)
                           FROM lessons
                           WHERE course = NEW._course_id)) THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER ModifyStartDateUpdate
BEFORE UPDATE OF _start_date ON courses
FOR EACH ROW
EXECUTE FUNCTION modify_start_date_update();

```

```

CREATE FUNCTION modify_end_date_update() RETURNS trigger AS $$ 
BEGIN
    IF(OLD._end_date <= current_date
    OR NEW._end_date <= current_date
    OR NEW._end_date < (SELECT max(_date)
                           FROM lessons
                           WHERE course = NEW._course_id)) THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER ModifyEndDateUpdate
BEFORE UPDATE OF _end_date ON courses
FOR EACH ROW
EXECUTE FUNCTION modify_end_date_update();

```

- ❖ Se si desidera modificare la data di inizio o di fine di un corso allora nessuna delle seguenti condizioni deve essere vera:

- la data che si vuole modificare è già passata o è la data odierna
- la nuova data inserita è la data odierna o una che la precede
- la nuova data di inizio è posta dopo la data della prima lezione del corso
- la nuova data di fine è posta prima della data dell'ultima lezione del corso

(Se si volesse modificare la data di inizio e di fine del corso ma ci sono lezioni che lo impediscono, si cancellano tali lezioni e poi si modificano le date del corso). Non si controlla se la data di fine è dopo la data di inizio perché è stato creato un check apposito per quel controllo. È un BEFORE trigger sull'operazione di UPDATE.

5.3 Definizione dei ruoli

In questo progetto, un utente può assumere un determinato ruolo a seconda della carica che ricopre. I ruoli sono divisi su due livelli e in entrambi sono presenti 3 figure diverse.

Il primo livello di ruoli si nota banalmente a livello di WebApp, in cui un utente può essere uno studente o un professore o un addetto alla segreteria. Invece, il secondo livello fa riferimento al DB. In quest'ultimo caso sono stati creati 3 tipologie di utente differenti: segreteria, professor e admin.

```
db.session.execute("CREATE USER admin WITH password 'admin';")
db.session.execute("CREATE USER segreteria WITH password 'segreteria';")
db.session.execute("CREATE USER professor WITH password 'professor';")
```

- L'utente 'professor' corrisponde a un utente che ha ruolo 'professor' nella web application;
- L'utente 'segreteria' corrisponde a un utente che ha ruolo 'admin' nella web application;
- L'utente 'admin' rappresenta l'utente che gestisce la base di dati.

Non è presente un utente che rappresenta uno studente perché non gli è possibile accedere alla base di dati in modo diretto.

5.4 Assegnamento dei privilegi e dei permessi

```
db.session.execute('GRANT USAGE ON SCHEMA "Pcto" TO admin; ')
```

Come si può dedurre dal comando sopra, si garantisce solo all'utente 'admin' di poter operare sullo schema creato.

TABELLA USERS

```
# PERMESSI TABELLA Users
db.session.execute('GRANT DELETE, INSERT, UPDATE, SELECT ON TABLE users TO admin; ')
db.session.execute('GRANT SELECT, INSERT ,UPDATE ON TABLE users TO segreteria; ')
db.session.execute('GRANT SELECT ON TABLE users TO professor; ')
```

- ADMIN: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)
- SEGRETERIA: può selezionare, inserire e modificare i dati della tabella (SELECT, INSERT, UPDATE)
- PROFESSOR: può selezionare, inserire e modificare i dati della tabella (SELECT, INSERT, UPDATE)

TABELLA COURSES

```
# PERMESSI TABELLA Courses
db.session.execute('GRANT DELETE, SELECT, INSERT, UPDATE ON TABLE courses TO admin;')
db.session.execute('GRANT DELETE, SELECT, INSERT, UPDATE ON TABLE courses TO segreteria;')
db.session.execute('GRANT DELETE, SELECT, INSERT, UPDATE ON TABLE courses TO professor;')
```

- ADMIN: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)
- SEGRETERIA: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)
- PROFESSOR: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)

TABELLA H_SCHOOLS

```
# PERMESSI TABELLA Schools
db.session.execute('GRANT DELETE, SELECT, INSERT, UPDATE ON TABLE h_schools TO admin;')
db.session.execute('GRANT DELETE, SELECT, INSERT, UPDATE ON TABLE h_schools TO segreteria;')
db.session.execute('GRANT SELECT ON TABLE h_schools TO professor;')
```

- ADMIN: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)
- SEGRETERIA: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)
- PROFESSOR: può solamente selezionare i dati della tabella (SELECT)

TABELLA LESSONS

```
# PERMESSI TABELLA Lessons
db.session.execute('GRANT DELETE, SELECT, INSERT, UPDATE ON TABLE lessons TO admin;')
db.session.execute('GRANT SELECT, INSERT, UPDATE , DELETE ON TABLE lessons TO segreteria;')
db.session.execute('GRANT SELECT, INSERT, UPDATE , DELETE ON TABLE lessons TO professor;')
```

- ADMIN: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)
- SEGRETERIA: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)
- PROFESSOR: possiede tutti i privilegi (SELECT, INSERT, UPDATE, DELETE)

6. Parti interessanti del codice

Le parti interessanti del codice, dal nostro punto di vista, sono alcune query e come esse vengono utilizzate e il file module_extensions.py.

6.1 Query particolari nel codice e come sono state utilizzate

```
courses = db.session.query(Course).join(User_courses).filter(User.user_id == current_user.user_id)
# Si individuano tutte le lezioni dei corsi a cui l'utente vuole partecipare
for c in courses:
    list.append(db.session.query(Lesson).join(User_lessons).filter(User.user_id == current_user.user_id,
                                                                Lesson.course == c.course_id).all())
```

La query sopra è presente nel file route.py della cartella BaseRoute.

Si vuole analizzare questo pezzo di codice.

La prima query, il cui risultato finisce nella variabile ‘courses’, restituisce i corsi che quel determinato utente intende frequentare.

La query successiva si trova all'interno di un ciclo for che scorre tutti i corsi appena trovati e cerca le lezioni a cui l'utente intende partecipare. Per questo motivo è necessaria una join con ‘User_lessons’ ponendo la condizione che l'id dell'utente e il corso a cui si riferisce la lezione siano quelli specificati man mano.

```

q = user_with_id(form.user.data).first()
list = form.school.data.split(":")
# Si associa l'utente alla scuola dichiarata
school_with_phone(list[-1]).first().add_student(q)

```

Queste linee di codice sono presenti nel file route.py della cartella BaseRoute.

Le variabili ‘q’ e ‘list’ contengono rispettivamente un particolare studente e una lista di parole provenienti dal form che vanno a specificare una scuola.

L’ultimo campo del form appena citato è il telefono, perciò, nella riga successiva al commento viene cercata la scuola con quel determinato numero telefonico e viene associato lo studente presente in ‘q’.

```

# Controllo sulla validità dell'utente
def validate_id(self, data):
    # q = Lezioni che sono associate al corso interessato
    q = db.session.query(Lesson).join(Course).filter(Course.course_id == self.id.data).all()

    # subquery = Lezioni a cui l'utente ha partecipato di quel determinato corso
    # e di cui ha registrato la presenza
    subquery = db.session.query(Lesson).join(User_lessons).filter(User_.user_id == self.user.data,
                                                                Lesson.course == self.id.data).all()

    # Si è deciso che il certificato viene generato solo se l'utente ha partecipato ad almeno il 70 % delle lezioni
    # Se sono presenti lezioni
    if len(q) > 0:
        # Controllo per verificare che il corso sia terminato
        if str(course_with_id(self.id.data).first()._end_date) >= str(datetime.now()):
            raise ValidationError("Il corso non è ancora finito!")
        else:
            # Per generare il certificato è necessario che l'utente abbia registrato almeno il 70% delle presenze
            if len(subquery) / len(q) * 100 < 70:
                raise ValidationError("Non è stata attestata la partecipazione ad almeno il 70% delle lezioni!")
    else:
        flash("Non ci sono lezioni!")

```

Questo pezzo di codice è presente nel file forms.py della cartella Courses e lavora con due query.

La prima query cerca le lezioni associate a un determinato corso (q).

La seconda query restituisce le lezioni a cui l’utente ha partecipato di quel determinato corso e ne ha dichiarato la presenza (subquery).

I risultati di queste due operazioni vengono utilizzati per controllare se è possibile o meno generare per quell’utente il certificato di partecipazione al corso interessato.

Il certificato viene generato solo se l’utente ha dichiarato la presenza ad almeno il 70% delle lezioni e se il corso è terminato.

Di conseguenza, se il numero delle lezioni del corso è superiore a 0 (un corso potrebbe non aver ancora delle lezioni in programma perché è stato appena creato) ma non è ancora concluso, allora non è possibile ottenere il certificato.

Altrimenti, vengono controllate le presenze. Se queste non sono pari ad almeno il 70% del totale viene mostrato un avviso: “Non è stata attestata la partecipazione ad almeno il 70% delle lezioni!”.

```

# lessons = Lezioni che si riferiscono al corso interessato
lessons = db.session.query(Lesson).join(Course).filter(
    Course._course_id == self.course_id.data).order_by(
    Lesson._date).all()
# Se c'è una lezione dopo della nuova data di fine corso inserita allora la modifica non è concessa
if lessons[-1]._date > self.end_date.data:
    flash("Operazione non riuscita. Riaprire il form per visualizzare l'errore!", 'warning')
    raise ValidationError("Ci sono delle lezioni dopo della data di fine corso che si sta cercando di inserire!")

```

Questi comandi sono presenti nel file forms.py della cartella Courses.

La variabile ‘lessons’ contiene il risultato di una query che cerca le lezioni che si riferiscono a un determinato corso e le pone in ordine cronologico.

Il valore ‘lessons[-1]’ corrisponde esattamente all’ultima lezione cronologicamente parlando di quel determinato corso.

Questo pezzo di codice controlla infatti che la data di fine corso inserita non sia precedente all’ultima lezione del corso. Se è così, viene lanciato un avviso e si impedisce l’inserimento.

```

# lessons = Lezioni che si riferiscono al corso interessato
lessons = db.session.query(Lesson).join(Course).filter(Course._course_id == self.course_id.data).order_by(
    Lesson._date).all()
# Se c'è una lezione prima della nuova data di inizio corso inserita allora la modifica non è concessa
if lessons and lessons[0]._date < self.start_date.data:
    flash("Operazione non riuscita. Riaprire il form per visualizzare l'errore!", 'warning')
    ValidationError("Ci sono delle lezioni prima della data di inizio corso che si sta cercando di inserire!")

```

Questi comandi sono presenti nel file forms.py della cartella Courses.

Analogamente al codice appena descritto, ‘lessons’ contiene le lezioni che si riferiscono a un determinato corso in ordine cronologico e si controlla che la data di inizio corso non sia posteriore alla prima lezione del corso stesso.

La prima lezione è data dal valore di ‘lessons[0]’.

```

# course_lesson = lezioni che sono associate al corso interessato in ordine di data e di ora
course_lesson = db.session.query(Lesson).join(Course).filter(Course._name == course).order_by(
    Lesson._date).order_by(Lesson._start_hour)

numero_ore_fatte = timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)
zero = timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)

# Per ogni lezioni del corso
for x in course_lesson:
    # Se la data della lezione precede o è uguale a quella di oggi, il numero di ore fatte incrementa
    if x._date <= date.today():
        numero_ore_fatte += datetime.combine(date.today(), x._end_hour) - datetime.combine(date.today(),
            x._start_hour)

# Se il numero di ore supera 0 allora la barra di progresso del corso sarà proporzionale al numero di ore di lezione sostenute
if numero_ore_fatte > zero:
    progress_bar = int(((numero_ore_fatte.seconds / 3600) / info_corso._n_hour) * 100)

```

Il codice qui mostrato è parte del file route.py della cartella Courses.

Si vuole capire quante lezioni sono state sostenute così da rappresentare in modo corretto la barra di progresso del corso.

La variabile ‘course_lesson’ racchiude le lezioni del corso interessato in ordine cronologico e di orario (per l’ora di inizio).

Si creano delle variabili di appoggio ('zero' e 'numero_ore_fatte') e grazie al ciclo for si controlla la data di ciascuna lezione: se la data è uguale alla data odierna o la precede allora il numero di ore fatte aumenta. Se il numero di ore fatte è maggiore di 'zero' (variabile che equivale a 0 ore, 0 minuti e 0 secondi) allora si calcola in proporzione quanto deve essere coperta la barra di progresso del corso.

6.2 module_extensions.py

```
class CRUDMixin(Model):
    """Mix-in aggiunge metodi di convenienza per le operazioni CRUD (create, read, update, delete)"""

    # Funzione per l'aggiornamento di un record
    def update(self):
        try:
            db.session.commit()
        except:
            db.session.rollback()
            raise IntegrityError()

    # Funzione per il salvataggio di un record
    def save(self, commit=True):
        """Save the record."""
        try:
            db.session.add(self)
            if commit:
                db.session.commit()
            return True
        except:
            db.session.rollback()
            raise IntegrityError()

    # Funzione per la rimozione di un record
    def delete(self):
        """Remove the record from the database."""
        try:
            db.session.delete(self)
            db.session.commit()
            return True
        except:
            db.session.rollback()
            raise IntegrityError()
```

La classe CRUDMixin presente all'interno di module_extensions.py racchiude tre funzioni utilizzate per gestire il meccanismo delle transazioni: update, save e delete.

Tutte e tre le funzioni accettano in input il dato interessato e tentano di compiere rispettivamente il suo aggiornamento, il suo salvataggio e la sua rimozione dalla base di dati. Si è utilizzato il verbo "tentare" per sottolineare il fatto che l'operazione potrebbe non concludersi correttamente e a quel punto verrebbe lanciato un errore ed eseguito un rollback, ovvero si tornerebbe alla situazione precedente l'inizio dell'operazione richiesta.

Al contrario, se l'operazione giunge a termine con successo i risultati dell'aggiornamento, del salvataggio e/o della rimozione saranno permanenti nella base di dati.

Si tiene a precisare che in un ambiente di sviluppo portato alla finalizzazione del progetto si toglierebbe il lancio dell'errore e rimarrebbe semplicemente l'operazione di rollback, ma non essendo un progetto pienamente concluso in tutti i suoi aspetti si è deciso di tenerlo.

```
bcrypt = Bcrypt()
db = SQLAlchemy(model_class=CRUDMixin)
migrate = Migrate()
```

Al termine del file è possibile notare queste ultime tre linee di codice.

Tali comandi servono per inizializzare i pacchetti che vengono usati durante il progetto installati tramite Python.

7. Tecnologie usate

- Flask-Principal 0.4.0
estensione utilizzata per gestire l'autenticazione degli utenti
- Flask-Bcrypt 1.0.1
estensione utilizzata per applicare hashing alle password inserite dagli utenti
- Flask-Login 0.6.0
estensione utilizzata per gestire il login, le sessioni e il logout degli utenti
- Flask-SQLAlchemy 2.5.1
estensione utilizzata per semplificare l'uso di SQLAlchemy
- Flask-Migrate 3.1.0
estensione utilizzata per gestire le migrazioni del database SQLAlchemy
- WTForms 3.0.1
libreria utilizzata per validare i form compilati dagli utenti
- Flask-WTF 1.0.1
estensione utilizzata per integrare Flask e WTForms
- alembic 1.7.7
tool utilizzato per la migrazione del database

8. Divisione dei compiti nel progetto

Il progetto deve evidenziare la conoscenza dell'utilizzo di Flask, la conoscenza dell'utilizzo di SQLAlchemy, padronanza di SQL (nel nostro caso abbiamo utilizzato il linguaggio ORM) e di uno specifico DBMS.

Riportiamo in questa tabella i nomi di chi ha contribuito in maniera maggiore agli aspetti principali del progetto.

Sezione del progetto	Realizzazione	Revisione
Creazione schema della base di dati	Carola	Riccardo, Carola, Serena
Creazione e basi del progetto in Flask	Riccardo	Riccardo, Carola, Serena

Creazione utenti, ruoli e permessi	Riccardo, Carola, Serena	Riccardo, Carola, Serena
Meccanismo di login e registrazione	Riccardo	Riccardo, Carola, Serena
Schermate in HTML	Riccardo, Carola	Riccardo, Carola, Serena
Codice Python	Riccardo, Carola, Serena	Riccardo, Carola, Serena
Codice SQL (linguaggio ORM)	Riccardo, Carola, Serena	Riccardo, Carola, Serena
Sicurezza DB	Riccardo, Carola, Serena	Riccardo, Carola, Serena
Trigger	Serena	Riccardo, Carola, Serena
Transazioni	Riccardo	Riccardo, Carola, Serena
Commenti al codice	Serena	Riccardo, Carola, Serena
Documentazione	Riccardo, Carola, Serena	Riccardo, Carola, Serena