Gym Casteo

Gruppo Casteo: Andrea Rosa 882014, Sebastiano Quintavalle 878500,
Simone Jovon 882028

Indice

1.Introduzione

Con la nostra applicazione abbiamo cercato di fare del nostro meglio per realizzare una piattaforma di supporto alle funzionalità di una palestra in tempi di Covid.
Per farlo abbiamo pensato ai punti critici di una palestra in questo periodo di pandemia e secondo noi sono i seguenti: capienza, tempo di permanenza, tracciamento dei contatti, flessibilità ai cambiamenti e controllo degli accessi.
Questi 5 punti li abbiamo usati come base per sviluppare le funzionalità che compongono la nostra applicazione, cercando di implementare solo quelle più importanti ed essenziali.
A livello di front-end, abbiamo cercato di realizzare un'interfaccia che fosse essenziale ma allo stesso tempo facilmente approcciabile e utilizzabile dall'utente.

2.Funzionalità Principali

Le varie funzionalità che compongono la nostra applicazione si distinguono, in gran parte, a seconda del ruolo assegnato al proprio account con il quale si è loggati in quel momento.
I ruoli sono 3: Cliente, Personal Trainer e Amministratore.

Il Cliente ha la possibilità di iscriversi ai corsi offerti dalla palestra e tenuti da un Personal Trainer, ppure ha la possibilità di prenotare un turno di allenamento individuale in una delle stanze che compongono la palestra.

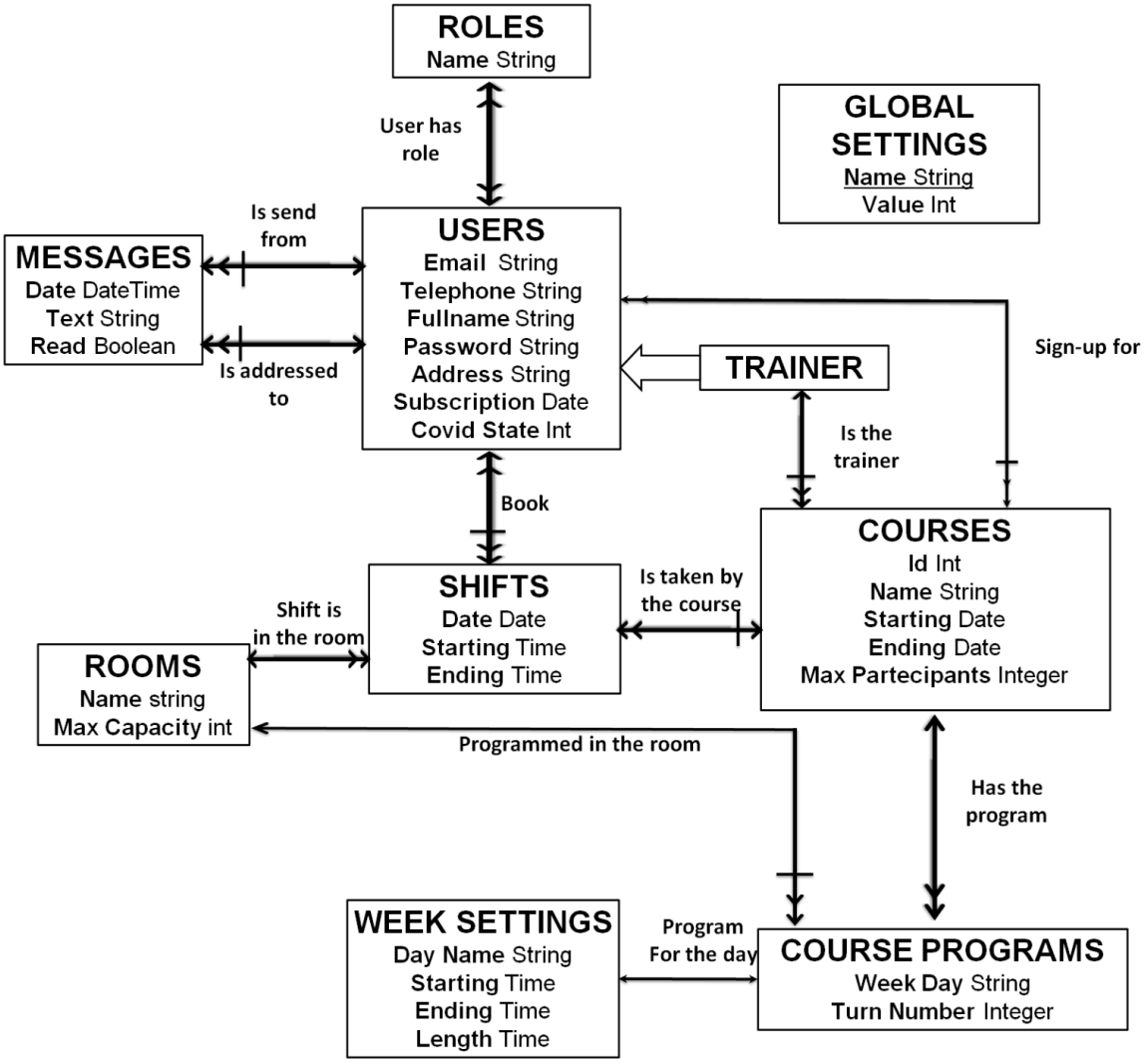
Il Personal Trainer, oltre alle funzionalità precedenti, può creare e gestire i propri corsi, pianificandone orari, giorni, stanze, ecc.

L'amministratore ha una visione completa della palestra, infatti dalla sua pagina personale può controllare la situazione di corsi, prenotazioni e utenti. Inoltre ha la possibilità di poter decidere, in base alle Disposizioni Ministeriali del momento, tutte le limitazioni di tempo, capienza delle stanze, apertura e chiusura della palestra.

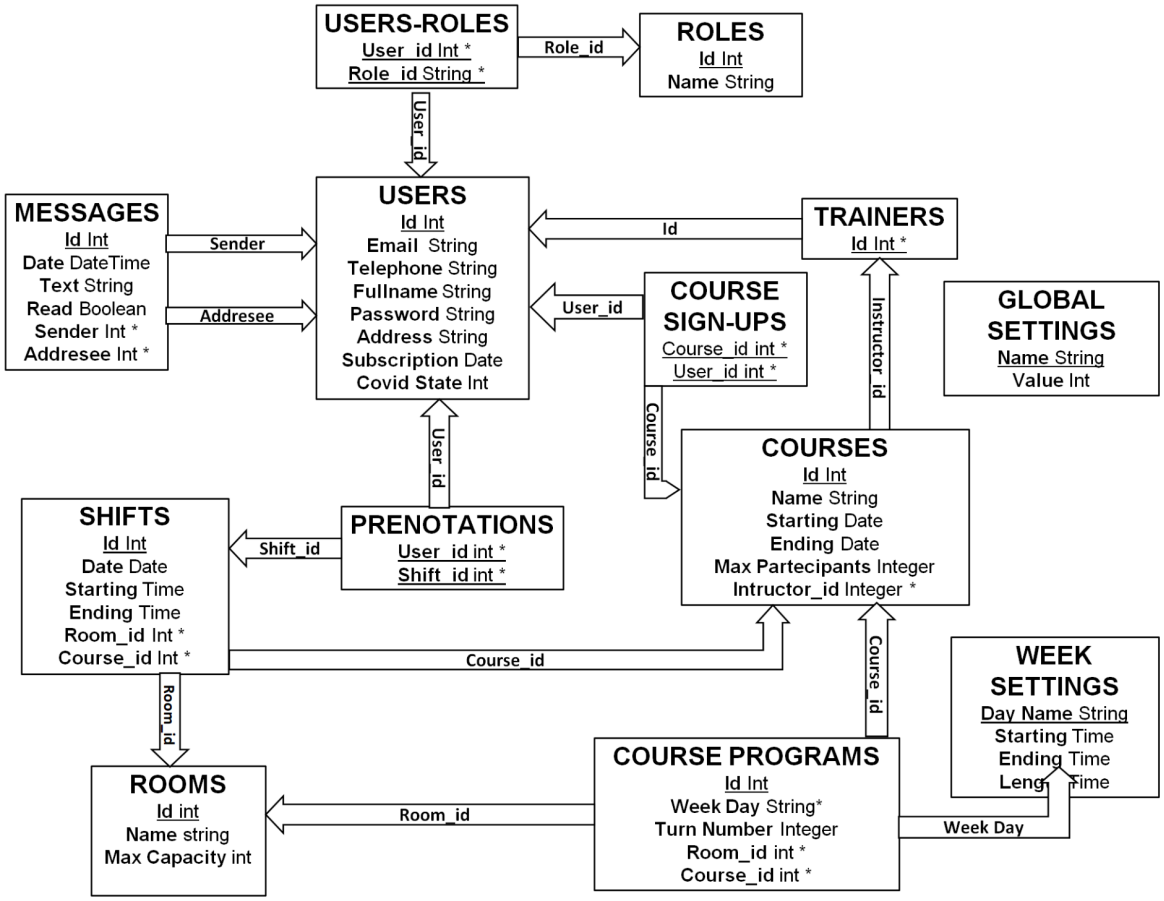
Per quanto riguarda il tracciamento dei contatti, Clienti e Personal Trainer hanno la possibilità di poter segnalare la propria positività agli altri Clienti, che sono stati in loro compagnia durante un turno o un corso, e ai Personal Trainer con cui sono venuti in contatto.
Ai Clienti e ai Personal Trainer che risultano essere dei contatti stretti di un positivo, verrà recapitato un messaggio, visibile dalla propria area personale, con cui li si informa del contatto avvenuto.
L'Admin può vedere e gestire la situazione Covid di Clienti e Personal Trainer tramite la propria area personale.

3.Progettazione Concettuale e Logica

Schema Concettuale



Schema Relazionale



Users

Le righe di questa tabella rappresentano gli utenti della palestra. Un utente avrà un id che lo identifica, un nome e cognome, un'email che assieme alla password permetteranno all'utente di accedere all'app, un numero di telefono, un indirizzo di casa, la data fino alla quale la sua iscrizione alla palestra è valida e per finire un campo che rappresenta il suo stato rispetto al Coronavirus (negativo, positivo oppure contatto con un positivo). L'attributo email inoltre sarà anche quello identificativo per l'utente in quanto due utenti non potranno registrarsi con la stessa email.

Trainers

E' una sottoclasse della tabella "Users" e serve a rappresentare un trainer della palestra, il quale potrà tenere dei corsi. La tabella avrà solo un campo id che identifica i vari trainer e che sarà anche una Foreign Key verso la tabella "Users".

Roles

Questa tabella serve per attuare le politiche di autorizzazione alla nostra applicazione, infatti le sue righe memorizzano i vari ruoli che un utente potrà avere all'interno della palestra e quindi anche dell'applicazione. I suoi campi sono l'id del ruolo che lo identifica e il nome del ruolo diverso per ognuno.

Users_roles

Non è altro che una tabella di supporto per permettere la relazione molti-a-molti tra la tabella “User” e la tabella “Roles”. Le righe di questa tabella avranno quindi solo una coppia di campi user_id (Foreign Key alla tabella “Users”) e role_id (Foreign Key alla tabella “Users”), entrambi Primary Key.

Messages

Le righe di questa tabella rappresentano messaggi inviati all’interno dell’applicazione tra un utente e un altro della palestra. Questi messaggi serviranno ad esempio a comunicare l’eventuale contatto con un cliente della palestra risultato positivo oppure ricevere comunicazioni da un corso al quale l’utente si è iscritto. I suoi campi saranno un id che identifica i vari messaggi, la data in cui il messaggio è stato inviato, un campo booleano che indica se il messaggio è già stato letto, il testo del messaggio ed infine il mittente e il destinatario del messaggio che saranno due Foreign Key alla tabella “Users”

Rooms

All’interno di questa tabella vengono memorizzate le varie stanze di una palestra, le quali avranno un id che le identifica, un nome (diverso per ogni stanza) e la massima capienza che quella stanza può ospitare.

Prenotations

Un utente per allenarsi individualmente dovrà prenotarsi per uno specifico turno della palestra e questa tabella memorizza proprio queste prenotazioni. Una prenotazione avrà una coppia user_id e shift_id che la identifica, user_id rappresenta l’utente che effettua la prenotazione e sarà una Foreign Key verso la tabella “Users”, mentre shift_id rappresenta il turno in cui l’utente si prenota e sarà una Foreign Key verso la tabella “Shifts”.Abbiamo deciso di creare un’altra tabella che rappresenta i turni e di non memorizzare i dati di un turno direttamente sulle prenotazioni per non aver dati ripetuti inutilmente e quindi evitare anche possibili anomalie Inoltre come vedremo in seguito questa scelta è stata fatta anche per unificare i turni prenotabili da un utente e quelli assegnati ad un corso poichè concettualmente sono essenzialmente la stessa entità.

Shifts

In questa tabella vengono memorizzati i vari turni della palestra. Ogni turno ha un id che lo identifica, la data a cui appartiene, l’orario di inizio e di fine del turno, la stanza a cui il turno si riferisce (che sarà una Foreign Key alla tabella “Rooms”) ed per finire un campo che è una Foreign Key per la tabella “Courses”, infatti se il turno appartiene ad un corso offerto dalla palestra questo punterà ad un corso altrimenti se il turno è libero e prenotabile per allenamento individuale il valore sarà settato a null. Per motivi di efficienza spaziale un turno verrà memorizzato nella base di dati solo se è assegnato ad un corso o se almeno un utente si è prenotato per quel turno.

Courses

Rappresenta i corsi offerti dalla palestra, ogni corso ha un id che lo rappresenta, un nome che sarà diverso per ogni corso, una data di inizio e di fine del corso, il numero massimo di utenti che si possono iscrivere al corso e l’istruttore del corso che sarà rappresentato da una

Foreign Key verso la tabella “Trainers”. Ulteriori informazioni riguardanti il corso sono memorizzate nella tabella “Course_programs”.

Course_programs
All’interno di questa tabella vengono memorizzati i vari programmi dei corsi attraverso i quali sarà possibile assegnare i vari turni. Un programma è caratterizzato da un id che lo identifica, dal corso a cui appartiene che sarà rappresentato attraverso una Foreign Key verso la tabella “Courses”, dal giorni della settimana a cui si riferisce che sarà anche una Foreign Key verso la tabella “Week_settings”, dal numero di turno assegnato al corso in quello specifico giorno della settimana e per finire la stanza assegnata per quel specifico turno del corso.

Course_signs_ups
Rappresenta le iscrizioni degli utenti ai vari corsi. L’iscrizione ad un corso sarà caratterizzata da una coppia user_id e course_id che la rappresenterà, inoltre user_id sarà una Foreign Key alla tabella “Users” mentre course_id sarà una Foreign Key alla tabella “Courses”.

Week_settings
Questa tabella memorizza le informazioni sugli shifts e gli orari della palestra per ogni giorno della settimana. Ogni riga della tabella avrà un campo per il nome del giorno della settimana (che inoltre identificherà la riga della tabella), l’orario di apertura e chiusura della palestra per quel giorno della settimana ed infine la lunghezza dei turni per quel giorno della settimana.

Global_settings
Questa tabella serve per memorizzare i vari settings della palestre e dell’applicazione, come ad esempio orario di apertura e chiusura massimo, numero massimo di prenotazioni a settimana per un utente, massima e minima durata possibile di un turno, etc. Ogni riga avrà quindi due campi, il primo con il nome del setting da memorizzare (che identificherà la riga della tabella) e il secondo con il valore del setting.

4.Query Principali

Turni prenotati rimanenti di un utente

```
SELECT s.date, s.starting, s.ending, r.name AS room_name
FROM users u JOIN prenotations p ON u.id = p.user_id
         JOIN shifts s ON  s.id = p.shift_id
         JOIN rooms r ON s.room_id = r.id
WHERE u.fullname = 'Name Surname' AND
      ((s.date = 'YYYY-MM-DD' AND s.ending >= 'hh:mm:ss')
      OR s.date > 'YYYY-MM-DD')
```

Questa query, implementata per la sezione ‘Prenotations’ dell’area personale di Clienti e Personal Trainer, restituisce i turni prenotati rimanenti di un utente.

Iscrizioni di un utente

```
SELECT c.name, c.starting, c.ending
FROM users u JOIN course_signs_up cs ON u.id = cs.user_id
       JOIN courses c ON c.id = cs.course_id
WHERE u.fullname = 'Name Surname' AND c.ending >= 'YYYY-MM-DD'
```

Questa query, utilizzata nella sezione ‘Sign ups’ dell’area personale di Clienti e Personal Trainer, ritorna tutti i corsi, non ancora finiti, a cui l’utente è iscritto.
Nella stessa sezione si possono vedere anche le iscrizioni passate, quindi la query sarà identica ma con la differenza nel confronto tra la data corrente e la data di fine del corso.

Turni di un corso

```
SELECT s.date, s.starting, s.ending, r.name as room_name
FROM courses c JOIN shifts s ON c.id = s.course_id
        JOIN rooms r ON r.id = s.room_id
WHERE c.name = 'Course Name' AND s.date >= 'YYYY-MM-DD'
```

Questa query la ritroviamo nella sezione ‘Sign ups’ e per ogni corso non finito viene fornita una lista dei turni rimanenti del corso.

Utenti di un corso

```
SELECT u.fullname
FROM users u JOIN course_signs_up cs ON u.id = cs.user_id
       JOIN courses c ON c.id = cs.course_id
WHERE c.name='Course Name'
```

Nella sezione ‘Courses’ l’Amministratore può vedere, per ogni corso, la lista degli utenti iscritti.
Lo stesso può fare il Personal Trainer nella sezione ‘My Own Courses’.

Utenti di un turno

```
SELECT u.fullname
FROM shifts s JOIN prenotations p ON s.id = p.shift_id
        JOIN users u ON u.id = p.user_id
WHERE s.date = 'YYYY-MM-DD' AND s.starting = 'hh:mm:ss'
```

Nella sezione ‘Shift Booking’ l’Amministratore può vedere, per ogni turno che abbia delle prenotazioni, la lista degli utenti che hanno prenotato quel turno.

5.Principali scelte progettuali

Uno dei principali aspetti che abbiamo voluto curare nel nostro progetto è l' integrità della base di dati con cui la nostra app interagisce. Per assicurare ciò innanzitutto a livello dell' applicazione eseguiamo una serie di controlli per garantire il più possibile un'integrità della base di dati sottostante. Per avere un livello di sicurezza adeguato però questo non basta, quindi abbiamo implementato una serie di CHECK direttamente all'interno delle tabelle del database e altrettanti TRIGGER che assicurano il più possibile una corretta integrità e interazione con il database.

Un altro aspetto fondamentale di questo progetto sono la gestione dei ruoli degli utenti della nostra applicazione e le varie politiche di autorizzazione. Per la procedura di autenticazione di un utente all'interno dell'applicazione abbiamo deciso di utilizzare la libreria Flask-Login in quanto ci sembrava intuitiva da utilizzare e adatta alle nostre necessità. Per quanto riguarda la politica di autorizzazione esistono svariate librerie in merito ma noi abbiamo preferito implementare una nostra soluzione in base alle nostre esigenze.
Per fare ciò abbiamo creato all'interno della nostra base di dati una tabella "Roles" nella quale vengono memorizzati tutti i ruoli che i vari utenti potranno avere nella nostra applicazione. Questa tabella avrà quindi una relazione molti-a-molti con la tabella "Users" (che memorizza i vari utenti dell'applicazione), in quanto ogni utente potrà avere più di un ruolo. Successivamente all'interno della classe SessionUser, la quale estendendo la classe UserMixin verrà usata dalla libreria Flask-Login per eseguire l'autenticazione di un utente all'interno dell'applicazione, avremo un campo ".roles" che sarà una lista dei vari ruoli di un determinato utente loggato, i quali verranno ricavati attraverso appunto le tabelle "Roles" e "Users".

```python
class SessionUser(UserMixin):
    def __init__(self, id, email, pwd, roles, active=True):
        self.id = id
        self.email = email
        self.pwd = pwd
        self.roles = roles
        self.active = active
```

Avendo quindi memorizzato questa informazione per l'utente loggato nella sessione corrente, durante le varie richieste HTTP gestite dall' applicazione, sarà sufficiente controllare i vari ruoli e continuare l'esecuzione in base ai ruoli posseduti appunto dall'utente loggato.

Per quanto riguarda la base di dati abbiamo deciso di utilizzare il DBMS Postgres essendo compatibile e supportato da SQLAlchemy, vale a dire il tool che ci permette di far comunicare la nostra applicazione con la base di dati sottostante. Per la struttura della base di dati rimandiamo alla sezione "Progettazione Concettuale e Logica".

Come accennato precedentemente per l'interazione tra la base di dati e l'applicazione abbiamo utilizzato il tool SQLAlchemy e in particolare la sua funzionalità di ORM, per astrarre il più possibile il database sottostante. Per aver un livello di astrazione ancora maggiore abbiamo inoltre creato una serie di funzioni che interagendo con il database eseguiranno vari task specifici, così che la sezione nel nostro progetto che si occuperà di gestire le varie richieste HTTP non dovrà mai interagire direttamente con la base di dati, ma bensì si appoggerà a queste utilities. Un altro aspetto che ci ha indirizzati verso l'utilizzo della funzionalità di ORM di SQLALchemy é un suo elemento denominato Session, attraverso la quale avverrà la "conversazione" con il database e soprattutto astrae il concetto di transazione. Quindi creando una Session all'inizio di ogni richiesta HTTP avremo che questa richiesta interagirà con la base di dati all'interno di un'unica transazione e perciò il fallimento di una qualsiasi operazione all'interno della richiesta comporterà un ripristino della basi di dati nella situazione precedente alla richiesta stessa.

6.Struttura del Progetto

Il progetto che abbiamo consegnato è composto dai seguenti file:

1) **automap.py:** in cui sono definite in ORM tutte le tabelle che compongono il nostro database;
2) **gendb.py:** in cui viene creata l'istanza del database e vengono aggiunti tutti i trigger di controllo;
3) **populate.py:** si occupa di popolare tutte le tabelle del nostro database così da simulare un utilizzo reale;
4) **destroydb.py:** si occupa di distruggere la base di dati e tutti i dati memorizzati in caso di evenienza;
5) **model.py:** contiene tutti i metodi che servono per le funzionalità della nostra applicazione;
6) **app.py:** racchiude le tutte le routes della nostra applicazione;
7) **template:** contiene tutti i template html della nostra applicazione;
8) **static:** al suo interno ci sono le immagini utilizzate nell'applicazione.

La sequenza di comandi, per utilizzare in locale la nostra applicazione, è la seguente:

1) **python gendb.py**
2) **python populate.py**
3) **set FLASK_APP=app.py**
4) **flask run**
5) **sul browser aprire l'url: http://localhost:5000**

Il comando per distruggere la base di dati, e tutti i suoi dati, è il seguente:

1) **python destroydb.py**

Nei seguenti file dovranno essere inserite le proprie credenziali del proprio database locale nell'url sottostante, per permetterne la connessione:

1) **app.py**
2) **gendb.py**
3) **destroydb.py**
4) **populate.py**

```
engine = create_engine('postgresql://utente:password@localhost:5432/Gym', echo=False)
```