Progetto Basi di Dati
Gestione di una Palestra durante il Covid-19

Bit Nicola (878249), Campanelli Alessio (878170), Gottardo Mario (879088)

AA. 2020/2021

Indice

1 Introduzione

Lo scopo del progetto è la creazione di una web application per la gestione di una palestra nel corso di una pandemia, la quale renda necessario il controllo degli accessi nella struttura tramite prenotazioni, al fine di gestire il numero di persone e permettere un efficace tracciamento dei contatti di una persona all'interno della struttura.
La nostra web application permette di effettuare prenotazioni online, gestire corsi con istruttori, effettuare contact tracing sugli utenti con un occhio di riguardo per il mantenimento di vincoli di integrità tramite robusti trigger e solide politiche di sicurezza.
Ad eccezione del contact tracing, il prodotto è generale e perfettamente utilizzabile anche in periodi non di emergenza pandemica.
La web application comprende un front-end composto da HTML-CSS-JS, un back-end in Python realizzato usando le librerie Flask e SQLAlchemy e si interfaccia con una base di dati sottostante gestita tramite il DBMS Postgres.

1.1 Nota

Nel codice non sono presenti apostrofi per evitare errori di rendering con LaTeX.

2 Database

La base di dati sottostante alla web app è stata realizzata usando il DBMS Postgres. Lo schema comprende dodici tabelle, di seguito riportate e motivate.

2.1 Tabelle

2.1.1 generali

Contiene informazioni generali riguardanti la palestra. È composta da una riga unica che viene aggiornata e può essere vista come un insieme di variabili globali, per questo motivo la sua chiave primaria è arbitraria.

2.1.2 orariPalestra

Contiene gli orari della palestra per ogni giorno della settimana, quindi contiene solamente 7 righe modificabili.

La chiave primaria sono i giorni della settimana, che partono da domenica (1) fino a sabato (7)

2.1.3 giorniFestivi

Contiene gli orari della palestra per i giorni festivi. Se orario di apertura e chiusura sono `NULL` la palestra è considerata chiusa.

La chiave primaria è la data, in quanto unica.

2.1.4 persone

Contiene le informazioni delle persone, come nome, cognome, mail, ecc. Contiene anche le password hashate degli utenti

La chiave primaria è il codice fiscale dell'utente.

2.1.5 clienti

Partiziona persone, specificando le informazioni sul pagamento.

La chiave primaria è anche una chiave esterna che punta al CF della persona.

2.1.6 staff

Partiziona persone, specificando il ruolo del membro dello staff (istruttore o gestore)

La chiave primaria è anche una chiave esterna che punta al CF della persona.

2.1.7 prenotazioni

Ogni prenotazione corrisponde all'accesso di un cliente alla struttura, possono essere disapprovate in caso di irregolarità. Possono essere relative sia ad allenamenti liberi che ai corsi, per i quali viene segnato l'ID.

Ad ogni prenotazione è associato un ID che funge da chiave primaria. Possiedono inoltre diverse chiavi esterne, come il CF dell'utente, l'ID del corso a cui sono associate (`NULL` se è un allenamento libero) e l'ID della sala in cui si svolgerà.

2.1.8 corsi

Sono allenamenti con data, orari e sale specifici, tenuti da un istruttore. Possono avere un numero massimo di persone che è minore o uguale a quello della sala in cui si tengono.

La chiave primaria è un ID generato sequenzialmente, dato che rappresenta una singola lezione, mentre il nome può essere visto come un "tag" da poter seguire. Si collegano allo staff per l'ID dell'istruttore e all'ID della sala in cui si svolgeranno.

2.1.9 corsiSeguiti

Un cliente può seguire un corso, di conseguenza ogni corso con lo stesso nome sarà più facilmente visualizzabile dall'utente.

Essendo una tabella ausiliaria per una relazione molti a molti, tutti gli attributi sono chiave primaria.

2.1.10 sale

Le sale della palestra, possiedono un ID e un nome. Possiedono inoltre un numero massimo di persone che possono entrarci per evitare assembramenti

La chiave primaria è un ID sequenziale

2.1.11 notifiche

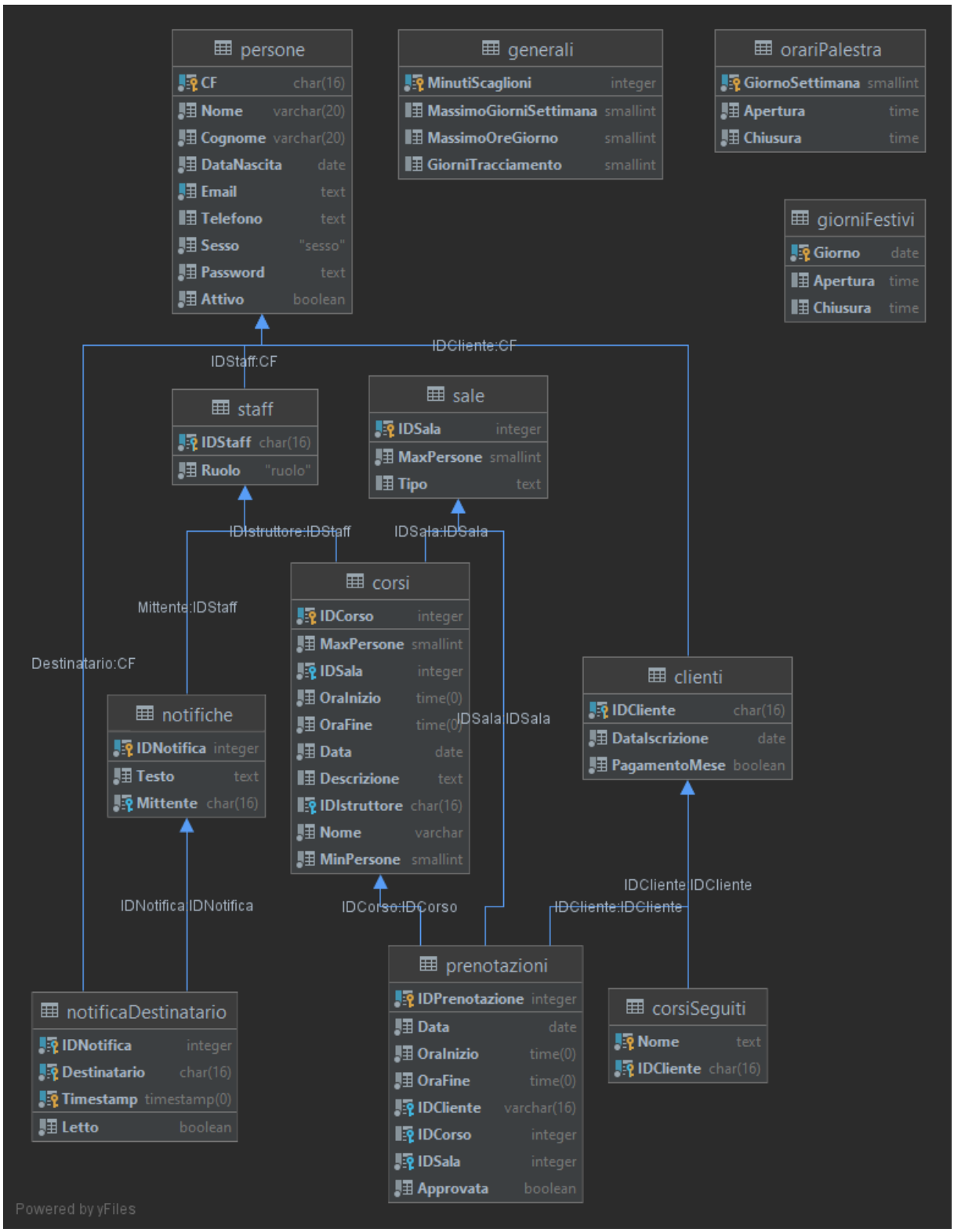
Testo e mittente di un messaggio, inviato dal sistema o dallo staff ai clienti della palestra

La chiave primaria è un'ID sequenziale, generato per le nuove notifiche, anche se possono essere riutilizzate nel caso ci siano messaggi predefiniti.

2.1.12 notificaDestinatario

Tabella che collega messaggio di notifica ai rispettivi destinatari e contenente un timestamp dell'istante in cui è stato inviato

Essendo una tabella ausiliaria per una relazione molti a molti, tutti gli attributi sono chiave primaria.



Raffigurazione dello schema

2.2 Trigger

2.2.1 Trigger su prenotazioni

Trigger che controlla se il cliente è attivo e ha pagato il mese corrente, se solo una delle due condizioni non è valida la prenotazione viene settata come non approvata (la prenotazione in un secondo momento potrà essere approvata e diventare quindi valida).

```
BEGIN
  IF NOT(
    (SELECT "PagamentoMese"
       FROM "clienti" c
       WHERE c."IDCliente" = NEW."IDCliente")
     AND
    (SELECT "Attivo"
     FROM "persone" p
     WHERE p."CF" = NEW."IDCliente")) THEN NEW."Approvata" = FALSE;
  END IF;

  RETURN NEW;
END;
```

```
CREATE TRIGGER controlla_1_pagamento
BEFORE INSERT OR UPDATE
ON public.prenotazioni
FOR EACH ROW
EXECUTE FUNCTION public.contolla_pagamento();
```

Trigger che controlla che gli orari della nuova prenotazione siano pertinenti con gli orari della palestra (controllando anche eventuali giorni festivi) e che la prenotazione sia per un momento futuro, non passato. Se almeno una di queste condizioni viene violata elimina la nuova prenotazione.

```
BEGIN
  IF (EXISTS
      (SELECT *
         FROM "giorniFestivi" gf
       WHERE NEW."Data" = gf."Giorno" AND (NEW."OraInizio" < gf."Apertura" OR
    NEW."OraFine" > gf."Chiusura"))
      OR EXISTS
      (SELECT *
       FROM "orariPalestra" op
       WHERE EXTRACT(DOW FROM NEW."Data") + 1 = op."GiornoSettimana" AND (NEW."
    OraInizio" < op."Apertura" OR NEW."OraFine" > op."Chiusura"))
    OR NEW."Data" < CURRENT_DATE
    OR (NEW."Data" = CURRENT_DATE AND NEW."OraInizio" < CURRENT_TIME)) THEN
    DELETE FROM "prenotazioni" pr WHERE pr."IDPrenotazione" = NEW."
    IDPrenotazione";
  END IF;

  RETURN NULL;
END;
```

```
CREATE TRIGGER controlla_2_orario_e_giorno_prenotazione
AFTER INSERT OR UPDATE
ON public.prenotazioni
FOR EACH ROW
EXECUTE FUNCTION public.controlla_orario_e_giorno_prenotazione();
```

Trigger che controlla che la nuova prenotazione non si sovrapponga ad un'altra dello stesso cliente, se ciò accade elimina la nuova prenotazione.

```
BEGIN
  IF(EXISTS(
    SELECT p."IDPrenotazione"
    FROM "prenotazioni" p
    WHERE p."IDCliente" = NEW."IDCliente" AND p."IDPrenotazione" <> NEW."
    IDPrenotazione" AND p."Data" = NEW."Data" AND
          ((NEW."OraInizio" BETWEEN p."OraInizio" AND p."OraFine" OR NEW."
    OraFine" BETWEEN p."OraInizio" AND p."OraFine") OR
          (p."OraInizio" BETWEEN NEW."OraInizio" AND NEW."OraFine" OR p."OraFine
    " BETWEEN NEW."OraInizio" AND NEW."OraFine"))))
  THEN
       DELETE FROM "prenotazioni" p WHERE p."IDPrenotazione" = NEW."
    IDPrenotazione";
  END IF;

  RETURN NULL;
END;
```

```
CREATE TRIGGER controlla_3_non_doppie_prenotazioni
AFTER INSERT OR UPDATE
ON public.prenotazioni
FOR EACH ROW
EXECUTE FUNCTION public.controlla_non_doppie_prenotazioni();
```

Trigger che controlla che la nuova prenotazione non violi il numero massimo di persone che possono essere presenti nella sala corrispondente, se ciò avviene la prenotazione viene eliminata.

```
BEGIN
  SELECT COUNT(*)
  INTO persone_attuali
  FROM prenotazioni p
  WHERE p."IDSala" = NEW."IDSala" AND p."OraFine" > NEW."OraInizio" AND p."
    OraInizio" < NEW."OraInizio" AND p."Approvata";

  IF(NEW."IDCorso" IS NOT NULL) THEN
    SELECT cr."MaxPersone"
    INTO numero_massimo
    FROM corsi cr
    WHERE cr."IDCorso" = NEW."IDCorso";
  ELSE
    SELECT s."MaxPersone"
    INTO numero_massimo
      FROM sale s
      WHERE s."IDSala" = NEW."IDSala";
  END IF;

  FOR f IN (
    SELECT Orario, SUM(NumPrenotazioni) AS Differenza
    FROM (
      SELECT p."OraInizio" AS Orario, COUNT(*) AS NumPrenotazioni
        FROM prenotazioni p
        WHERE p."IDSala" = NEW."IDSala" AND (p."OraInizio" BETWEEN NEW."
    OraInizio" AND NEW."OraFine") AND p."Approvata"
      GROUP BY p."OraInizio"
    UNION
      SELECT p."OraFine" AS Orario, -COUNT(*) AS NumPrenotazioni
      FROM prenotazioni p
      WHERE p."IDSala" = NEW."IDSala" AND (p."OraFine" BETWEEN NEW."OraInizio"
    AND NEW."OraFine") AND p."Approvata"
      GROUP BY p."OraFine") AS storico_entrate_uscite
    GROUP BY Orario
```

```
    ORDER BY Orario) LOOP
    persone_attuali = persone_attuali + f.Differenza;
    IF(persone_attuali > numero_massimo) THEN
      DELETE FROM "prenotazioni" WHERE NEW."IDPrenotazione" = "IDPrenotazione";
    END IF;
  END LOOP;

  RETURN NULL;
END;
```

```
CREATE TRIGGER controlla_4_posti_liberi
AFTER INSERT OR UPDATE
ON public.prenotazioni
FOR EACH ROW
EXECUTE FUNCTION public.controlla_posti_liberi();
```

Trigger che controlla che la nuova prenotazione non violi il numero massimo di allenamenti a settimana o le ore massime giornaliere che il singolo utente può effettuare. Se ciò avviene la prenotazione viene eliminata.

```
BEGIN
  IF (
    (SELECT COUNT(DISTINCT("Data"))
     FROM "prenotazioni" p
     WHERE p."IDCliente" = NEW."IDCliente" AND
      EXTRACT(WEEK FROM p."Data") = EXTRACT(WEEK FROM NEW."Data") AND
      EXTRACT(YEAR FROM p."Data") = EXTRACT(YEAR FROM NEW."Data")) > (SELECT "
    MassimoGiorniSettimana" FROM "generali")
    OR
    (SELECT SUM(p."OraFine" - p."OraInizio")
     FROM "prenotazioni" p
     WHERE p."IDCliente" = NEW."IDCliente" AND
      p."Data" = NEW."Data") > MAKE_INTERVAL(hours => (SELECT "MassimoOreGiorno"
     FROM "generali"))
  ) THEN
    DELETE FROM "prenotazioni" p WHERE p."IDPrenotazione" = NEW."IDPrenotazione"
    ;
  END IF;
  RETURN NULL;
END;
```

```
CREATE TRIGGER controlla_5_frequenza_massima
AFTER INSERT OR UPDATE
ON public.prenotazioni
FOR EACH ROW
EXECUTE FUNCTION public.controlla_frequenza_massima();
```

2.2.2 Trigger su corsi

Trigger che controlla che gli orari del nuovo corso siano pertinenti con gli orari della palestra (controllando anche eventuali giorni festivi) e che il corso sia per un momento futuro, non passato. Se almeno una di queste condizioni viene violata elimina il corso.

```
BEGIN
  IF (EXISTS
      (SELECT *
         FROM "giorniFestivi" gf
       WHERE NEW."Data" = gf."Giorno" AND (NEW."OraInizio" < gf."Apertura" OR
    NEW."OraFine" > gf."Chiusura"))
      OR EXISTS
```

```
      (SELECT *
       FROM "orariPalestra" op
       WHERE EXTRACT(DOW FROM NEW."Data") + 1 = op."GiornoSettimana" AND (NEW."OraInizio" < op."Apertura" OR NEW."OraFine" > op."Chiusura"))
      OR NEW."Data" < CURRENT_DATE
    OR (NEW."Data" = CURRENT_DATE AND NEW."OraInizio" < CURRENT_TIME)) THEN
    DELETE FROM "corsi" cr WHERE cr."IDCorso" = NEW."IDCorso";
  END IF;

  RETURN NULL;
END;
```

```
CREATE TRIGGER controlla_orario_e_giorno_corso
AFTER INSERT
ON public.corsi
FOR EACH ROW
EXECUTE FUNCTION public.controlla_orario_e_giorno_corso();
```

Trigger che controlla che il nuovo corso non si sovrapponga ad un altro dello stesso istruttore o ad un altro che si svolge nella stessa stanza. Se una delle due condizioni viene violata il corso viene eliminato.

```
BEGIN
  IF(EXISTS(
    SELECT *
    FROM "corsi" c
    WHERE (c."IDSala" = NEW."IDSala" OR c."IDIStruttore" = NEW."IDIstruttore")
    AND c."IDCorso" <> NEW."IDCorso" AND c."Data" = NEW."Data" AND
          ((NEW."OraInizio" BETWEEN c."OraInizio" AND c."OraFine" OR NEW."OraFine" BETWEEN c."OraInizio" AND c."OraFine") OR
          (c."OraInizio" BETWEEN NEW."OraInizio" AND NEW."OraFine" OR c."OraFine" BETWEEN NEW."OraInizio" AND NEW."OraFine"))))
  THEN
        DELETE FROM "corsi" c WHERE c."IDCorso" = NEW."IDCorso";
  END IF;

  RETURN NULL;
END;
```

```
CREATE TRIGGER controlla_sovrapposizioni_corsi
AFTER INSERT OR UPDATE
ON public.corsi
FOR EACH ROW
EXECUTE FUNCTION public.controlla_sovrapposizioni_corsi();
```

2.2.3 Trigger su clienti

Trigger che si attiva quando il cliente viene segnato come non pagante e setta le sue prenotazioni future come non approvate.

```
BEGIN
  UPDATE "prenotazioni"
  SET "Approvata" = FALSE
  WHERE "IDCliente" = NEW."IDCliente" AND "Data" >= CURRENT_DATE;

  RETURN NULL;
END;
```

```
CREATE TRIGGER disapprova_prenotazioni_non_paganti
AFTER UPDATE OF "PagamentoMese"
```

```
ON public.clienti
FOR EACH ROW
WHEN (new."PagamentoMese" = false)
EXECUTE FUNCTION public.disapprova_prenotazioni_non_paganti();
```

Trigger che si attiva quando il cliente viene nuovamente segnato come pagante e setta le sue prenotazioni future come approvate.

```
BEGIN
  UPDATE "prenotazioni" p
  SET "Approvata" = TRUE
  WHERE p."IDCliente" = NEW."IDCliente" AND p."Data" >= CURRENT_DATE;

  RETURN NULL;
END;
```

```
CREATE TRIGGER approva_prenotazioni_paganti
AFTER UPDATE OF "PagamentoMese"
ON public.clienti
FOR EACH ROW
WHEN (new."PagamentoMese" = true)
EXECUTE FUNCTION public.approva_prenotazioni_paganti();
```

2.2.4 Trigger su persone

Trigger che si attiva quando la persona viene segnata come non attiva e setta le sue prenotazioni future come non approvate.

```
BEGIN
  UPDATE "prenotazioni"
  SET "Approvata" = FALSE
  WHERE "IDCliente" = NEW."CF" AND "Data" >= CURRENT_DATE;

  RETURN NULL;
END;
```

```
CREATE TRIGGER disapprova_prenotazioni_disattivati
AFTER UPDATE OF "Attivo"
ON public.persone
FOR EACH ROW
WHEN (new."Attivo" = false)
EXECUTE FUNCTION public.disapprova_prenotazioni_disattivati();
```

Trigger che si attiva quando la persona viene nuovamente segnata come attiva e setta le sue prenotazioni future come approvate.

```
BEGIN
  UPDATE "prenotazioni"
  SET "Approvata" = TRUE
  WHERE "IDCliente" = NEW."CF" AND "Data" >= CURRENT_DATE;

  RETURN NULL;
END;
```

```
CREATE TRIGGER approva_prenotazioni_attivati
AFTER UPDATE OF "Attivo"
ON public.persone
FOR EACH ROW
```

```
WHEN (new."Attivo" = true)
EXECUTE FUNCTION public.approva_prenotazioni_attivati();
```

2.2.5 Trigger su giorniFestivi

Trigger che si attiva quando viene aggiunto un nuovo giorno festivo, elimina eventuali prenotazioni e/o corsi che violano gli orari imposti per il nuovo giorno festivo.

```
BEGIN
  DELETE FROM "prenotazioni" p
  WHERE p."Data" = NEW."Giorno" AND (NEW."Apertura" IS NULL OR NEW."Chiusura" IS
     NULL OR p."OraInizio" < NEW."Apertura" OR p."OraFine" > NEW."Chiusura");

  DELETE FROM "corsi" c
  WHERE c."Data" = NEW."Giorno" AND (NEW."Apertura" IS NULL OR NEW."Chiusura" IS
     NULL OR c."OraInizio" < NEW."Apertura" OR c."OraFine" > NEW."Chiusura");

  RETURN NULL;
END;
```

```
CREATE TRIGGER ellimina_prenotazioni_e_corsi_giorni_festivi
AFTER INSERT
ON public."giorniFestivi"
FOR EACH ROW
EXECUTE FUNCTION public.ellimina_prenotazioni_e_corsi_giorni_festivi();
```

2.3 Ruoli

Per questa applicazione abbiamo creato 4 ruoli, con diversi permessi.

2.3.1 Ospite

Gli ospiti sono coloro che visualizzano il sito senza essere autenticati. Possono visualizzare i corsi e le prenotazioni, e aggiungere nuove persone e clienti (registrazione)

2.3.2 Utente

Dopo aver eseguito il login o essersi registrati, l'ospite diventa utente. Gli utenti possono in più creare ed eliminare prenotazioni, leggere notifiche e cancellare istanze di *notificaDestinatario*. Possono inoltre visualizzare le tabelle di "variabili globali", necessarie per la prenotazione.

2.3.3 Istruttore

Gli istruttori non possono creare prenotazioni, ma possono creare notifiche.

2.3.4 Gestore

Il gestore oltre a poter visualizzare tutte le tabelle, può creare e modificare (dove possibile) corsi, sale, nuovi staff, giorni festivi.

3 Funzionalità principali

La web application permette la gestione online di una palestra, sia da parte dei clienti, che da parte dei gestori. I primi possono prenotarsi degli allenamenti e rimanere aggiornati sui corsi offerti, i secondi possono inserire nuovi corsi, notificare gli utenti e stabilire diverse politiche interne riguardanti orari e limitazioni, nonché eseguire contact tracing nel caso di clienti positivi. Ogni utente ha inoltre a disposizione un calendario per poter visualizzare velocemente le informazioni che gli interessano.

3.1 Corsi

I corsi vengono creati dai gestori nella propria dashboard. Per una questione di rapidità nella creazione, il sistema permette di impostare una settimana di inizio, con successiva scelta dei giorni, dell'orario e del numero di settimane (ripetizioni) per le quali lo si vuole tenere. Nel caso si volesse aggiungere lo stesso corso ad orari diversi, è necessario eseguire più volte tale operazione.

Due trigger controllano in ordine che:

1. gli orari siano coerenti con quelli della palestra
2. i corsi non si sovrappongano nella stessa stanza o con lo stesso istruttore

3.2 Prenotazioni

Le prenotazioni possono essere di due tipi: **ai corsi**, per le quali le informazioni principali sono definite da un membro dello staff, oppure **libere**, quindi non legate a nessun corso.

Per gli allenamenti liberi, è sufficiente per un utente entrare nella propria area riservata, inserire data, orari e una sala. È però necessario che questo possa poter eseguire la prenotazione (non disattivato e in regola con i pagamenti) e che inserendo questa prenotazione, non vada a superare i limiti imposti dalla palestra.
Ogni prenotazione approvata genera in maniera automatica un codice QR contenente le informazioni della prenotazione stessa, ciò renderebbe possibile un controllo automatico all'ingresso della palestra mediante dei tornelli dotati di scanner.

Cinque trigger controllano in ordine che:

1. l'utente possa effettivamente iscriversi (attivo e in regola)
2. l'orario e il giorno della prenotazioni siano coerenti con gli orari della palestra
3. non esistano prenotazioni dell'utente che si sovrappongono
4. le sale o il corso abbiano posti liberi a disposizione
5. l'utente non superi i limiti di allenamento

3.3 Contact Tracing

L'algoritmo di contact tracing prende in ingresso un istanza di Persona che sappiamo essere positiva (da qui in poi "positivo") e un numero di giorni da analizzare (al più 7 per evitare calcoli irrealistici), al termine dell'esecuzione, l'algoritmo ritorna una lista di potenziali positivi.
L'algoritmo cerca l'ultima data in cui il positivo è entrato nella struttura e determina il range usando il numero di giorni in input.
Trova tutte le volte in cui il positivo è entrato, per ogni ingresso trova tutte le prenotazioni che

intersecano quella in esame, salva il cliente e l'eventuale istruttore nel caso la prenotazione sia riferita ad un corso.
Una volta ottenuta la lista di potenziali positivi (renderizzata in un'apposita pagina di report), l'amministratore ha la possibilità di notificare gli interessati, disattivarli, in modo da rendere loro impossibile l'accesso alla struttura ed eventuali nuove prenotazioni fino alla riattivazione, e infine ha la possibilità di esportare un documento pdf con tutte le informazioni.

3.3.1 pseudocodice

```
def contact_tracing(zero, days):
    potential_infected = [zero.CF]
    days = int(days)
    if days > 7:
        days = 7

    last_zero_appearance_date = ultimo ingresso di positivo nella struttura

    if last_zero_appearance_date is not None:
        last_zero_appearance_date = last_zero_appearance_date.Data
    else:
        return [get_persona_by_cf(zero.CF)] # non abbiamo contatti da tracciare

    lower_limit_date = last_zero_appearance_date - timedelta(days=days)
    # definiamo il range usando il numero di giorni in input

    last_zero_appearances = tutti gli ingressi di positivo nell intervallo

    for appearance in last_zero_appearances:
        prenotazioni = tutte le prenotazioni che intersecano quella in esame

        if appearance.IDCorso is not None:
            istruttore = istruttore del corso
            potential_infected.append(istruttore)

        for p in prenotazioni:
            potential_infected.append(p.IDCliente)
    return [get_persona_by_cf(cf) for cf in (list(set(potential_infected)))]
    # Ritorniamo l istanza delle persone il cui codice fiscale risulta tra
    # i potenziali positivi, rimuovendo prima i duplicati
```

3.3.2 Query di contact tracing

Query per trovare l'ultimo ingresso del paziente zero

```
SELECT *
FROM prenotazioni p
WHERE p.IDCliente = zero.CF AND p.Data <= CURRENT_DATE AND
 p.Approvata = TRUE
ORDER BY p.DATA DESC
LIMIT 1
```

Query per trovare gli ingressi da analizzare del paziente zero

```
SELECT *
FROM prenotazioni p
WHERE p.IDCliente = zero.CF AND p.Approvata = TRUE AND
  p.Data BETWEEN lower_limit_date AND CURRENT_DATE
ORDER BY p.DATA DESC
```

Query per trovare gli ingressi da analizzare del paziente zero

```sql
SELECT *
FROM prenotazioni p
WHERE p.IDCliente = zero.CF AND p.Approvata = TRUE AND
  p.Data BETWEEN lower_limit_date AND CURRENT_DATE
ORDER BY p.DATA DESC
```

Query per trovare gli ingressi che si sovrappongono a quelli del paziente zero

```sql
SELECT *
FROM prenotazioni p
WHERE p."Data" = appearance."Data" AND p."IDSala" = appearance."IDSala" AND
    (appearance."OraInizio" BETWEEN p."OraInizio" AND p."OraFine" OR
        appearance."OraFine" BETWEEN p."OraInizio" AND p."OraFine" OR
        p."OraInizio" BETWEEN appearance."OraInizio" AND appearance."OraFine" OR
        p."OraFine" BETWEEN appearance."OraInizio" AND appearance."OraFine") AND
    p."Approvata" = TRUE
```

3.4 Calendario

Il calendario è uno strumento utile per avere una panoramica generale delle prenotazioni e dei corsi seguiti. La base è stata presa da questa pagina github, ma il codice JS è stato quasi completamente riscritto per poter assecondare le nostre necessità.

In base al ruolo dell'utente vengono passate diverse informazioni: l'admin vede tutti i corsi, gli istruttori tutti i corsi che tengono e i clienti le proprie prenotazioni e i corsi che seguono. Per ogni giorno, se si clicca sulla rispettiva cella, si apre una scheda modale contenente tutte le informazioni descritte sopra in ordine cronologico, colorate in base al fatto che siano corsi normali (azzurro), prenotazioni (verde), prenotazioni non approvate (rosso).

3.5 Limitazioni

Abbiamo deciso di offrire la possibilità di imporre delle limitazioni al numero massimo di giorni settimanali di allenamento e alle ore giornaliere. Queste sono utili in particolare nella remota possibilità che si verifichi un'epidemia globale, e ci sia la necessità di ridurre la probabilità di contagio in caso di clienti positivi.

Questi limiti possono essere modificati dalla dashboard dell'admin.

3.6 Notifiche

La web app offre un minimale sistema di messaggistica interna per gli utenti. I membri dello staff hanno la possibilità di inviare e ricevere notifiche, i clienti possono solamente riceverne. Il sistema di notifiche si basa su due tabelle nel Database, la tabella notifiche (la quale rappresenta una notifica) e la tabella notificaDestinatario (la quale rappresenta l'invio di una notifica ad un utente).
Le notifiche possono essere suddivise in due categorie: notifiche di sistema e notifiche custom. Le notifiche di sistema hanno ID noti e servono ad avvisare in maniera automatica o manuale di eventi con messaggi standardizzati. Le notifiche custom, invece, sono create al momento dell'invio e corrispondono a messaggi scritti direttamente da un membro dello staff e indirizzati ad uno o più utenti. L'invio di una notifica custom corrisponde quindi alla creazione di un record "notifica" e uno "notificaDestinatario" per ogni destinatario.
Esempi di notifiche predefinite dal sistema sono la notifica inviata in seguito alla disattivazione per contatto con un positivo (ID 0), la notifica di benvenuto (ID 3), la quale invita l'utente a recarsi in palestra per attivare il proprio account.
Una volta che una notifica è stata visualizzata è segnata in automatico come "letta", inoltre,

ogni utente ha la possibilità di cancellare la propria inbox, ciò corrisponde all'eliminazione dei relativi record nella tabella notificaDestinatario.

4 Scelte progettuali

4.1 Sicurezza

4.1.1 Salvataggio delle password

Le password degli utenti hashate al momento della registrazione e sono salvate nella tabella persone. Per gestire l'hashing delle password usiamo la libreria di Python bcrypt.
Procedura di hashing:

```
bcrypt.hashpw(request.form['password'].encode("utf-8"), bcrypt.gensalt()).decode
    ("utf-8")
```

Procedura di controllo in fase di login

```
bcrypt.checkpw(password, user.Password.encode("utf-8"))
```

Esempio di password hashata:
`$2b$12$JdlrwcQ6ChlUBsbQA2jWb.FxQnaVt8JKh4aE0wlE9TGouErquaZtu`

4.1.2 Connessioni

Abbiamo creato 4 diverse session di sqlalchemy, una per ogni ruolo (ospite, utente, istruttore, gestore) e in ogni query abbiamo utilizzato la session con i minimi privilegi necessari, seguendo quindi il principio dei privilegi minimi.

4.1.3 Transazioni

Dato che l'unica operazione che risulta critica a livello di concorrenza è l'inserimento di una nuova prenotazione abbiamo utilizzato un'apposita engine di sqlalchemy ORM (con relativa session) settata con livello di isolamento SERIALIZABLE, in modo da prevenire tutte le criticità (compresi i fantasmi).

4.2 Organizzazione progetto

Per l'organizzazione dell'applicazione abbiamo seguito le linee guida della documentazione ufficiale. Il file `config.py` contiene le opzioni di configurazione, mentre `run.py` contiene il codice essenziale per far partire l'applicazione. Le altre cartelle contengono:

- `appF`: la visualizzazione del DB in SQLAlchemy.orm (`models.py`) e la logica delle pagine (`views.py`)
- `instance`: contiene le configurazioni più sensibili come chiavi di sistema che non possono essere visualizzate, in quando non soggetto a versioning (inserito in `.gitignore`)
- `static`: contiene i file JS e CSS delle pagine web
- `templates`: contiene tutti i file HTML