Eugenio Alligrande 0334000168 Nola, 11/07/2025

Traccia d’Esame - Progetto di un Sistema Informativo per la Gestione di Eventi Culturali Locali *AgoraCultura*

Il progetto prevede la realizzazione di un sistema informativo completo per supportare l’organizzazione e la promozione di eventi culturali locali, migliorando l’accessibilità per cittadini, organizzatori e amministrazioni. Il sistema sarà basato su un database relazionale che archivierà tutte le informazioni essenziali relative agli eventi.

Il database conterrà una panoramica dettagliata degli eventi previsti o già svolti, con informazioni quali titolo, descrizione, data, orario, luogo, tipologia (concerto, mostra, conferenza, ecc.) e organizzatori. Ogni evento potrà essere collegato a uno o più luoghi (ad esempio, in caso di eventi itineranti o festival su più sedi) e a eventuali materiali promozionali o documentativi (locandine, video, brochure).

Gli utenti saranno suddivisi in tre principali categorie: cittadini (partecipanti), organizzatori e amministratore comunale. I cittadini potranno creare un profilo utente per iscriversi agli eventi, lasciare recensioni e suggerimenti, mentre gli organizzatori potranno proporre eventi da approvare e gestirne ogni aspetto. L’amministratore non interverrà direttamente sugli eventi, ma coordinerà gli operatori e gli organizzatori attraverso un sistema di gestione dei ruoli e delle responsabilità interne.

Il sistema permetterà la prenotazione gratuita o a pagamento per ciascun evento, con un controllo della disponibilità dei posti e generazione di biglietti digitali. Gli utenti registrati potranno visualizzare lo storico delle partecipazioni e delle recensioni, ricevere notifiche personalizzate sugli eventi di interesse e consultare le recensioni degli altri partecipanti.

Ogni evento sarà inoltre associato a categorie tematiche per facilitarne la ricerca, e potrà essere promosso tramite una sezione dedicata agli eventi in evidenza. Il sistema conserverà anche un archivio storico degli eventi passati, con la possibilità di consultare foto, materiali condivisi e commenti raccolti.

Una sezione dedicata al personale amministrativo e agli operatori culturali permetterà di tracciare ruoli, competenze e responsabilità. In particolare, sarà possibile definire relazioni gerarchiche tra utenti, in modo che un utente possa supervisionare altri utenti (ad esempio, un amministratore che supervisiona uno o più organizzatori). Questo meccanismo servirà per il coordinamento interno e per garantire trasparenza nella gestione degli eventi.

Obiettivi del Progetto

Il progetto prevede lo sviluppo di un sistema informativo per la gestione e la promozione degli eventi culturali locali, con l’obiettivo di supportare la trasformazione digitale delle attività culturali, migliorare l'accessibilità per i cittadini e ottimizzare la gestione da parte di organizzatori e amministrazioni.

1. Gestione dei profili utente e differenziazione dei ruoli
Ogni utente disporrà di un profilo personale contenente:

- Dati anagrafici e di contatto (nome, cognome, email, ecc.);
- Storico delle attività svolte nel sistema (eventi proposti, partecipazioni, recensioni);
- Preferenze tematiche (per suggerimenti e notifiche personalizzate);
- Ruolo e relazioni gerarchiche (per gli amministratori e operatori culturali).

2. Gestione e monitoraggio delle prenotazioni agli eventi

Il sistema supporterà una gestione dinamica delle prenotazioni per ciascun evento, prevedendo:

- Prenotazioni gratuite o a pagamento con verifica automatica della disponibilità posti;
- Stato delle prenotazioni: “Richiesta”, “Confermata”, “Annullata”;
- Generazione di biglietti digitali e registrazione dello storico di partecipazione per ogni cittadino.

3. Creazione, promozione e gestione degli eventi culturali

Gli organizzatori potranno proporre nuovi eventi e gestirne ogni aspetto:

- Inserimento di titolo, descrizione, data, orario, tipologia, luogo e materiale promozionale;
- Associazione a più luoghi e a categorie tematiche per facilitare la ricerca;
- Sezione “eventi in evidenza” per la promozione;
- Collegamento a locandine, foto, video, documenti e archiviazione storica.

4. Sistema di recensioni, suggerimenti e partecipazione attiva

Il sistema stimolerà la partecipazione attiva dei cittadini attraverso:

- Possibilità di lasciare recensioni e suggerimenti sugli eventi a cui hanno partecipato;
- Consultazione delle recensioni degli altri utenti;
- Ricezione di notifiche personalizzate in base a interessi, eventi seguiti o luoghi preferiti.

5. Gestione dei ruoli, delle responsabilità e delle supervisioni

L’amministrazione comunale e gli operatori culturali potranno gestire il sistema attraverso:

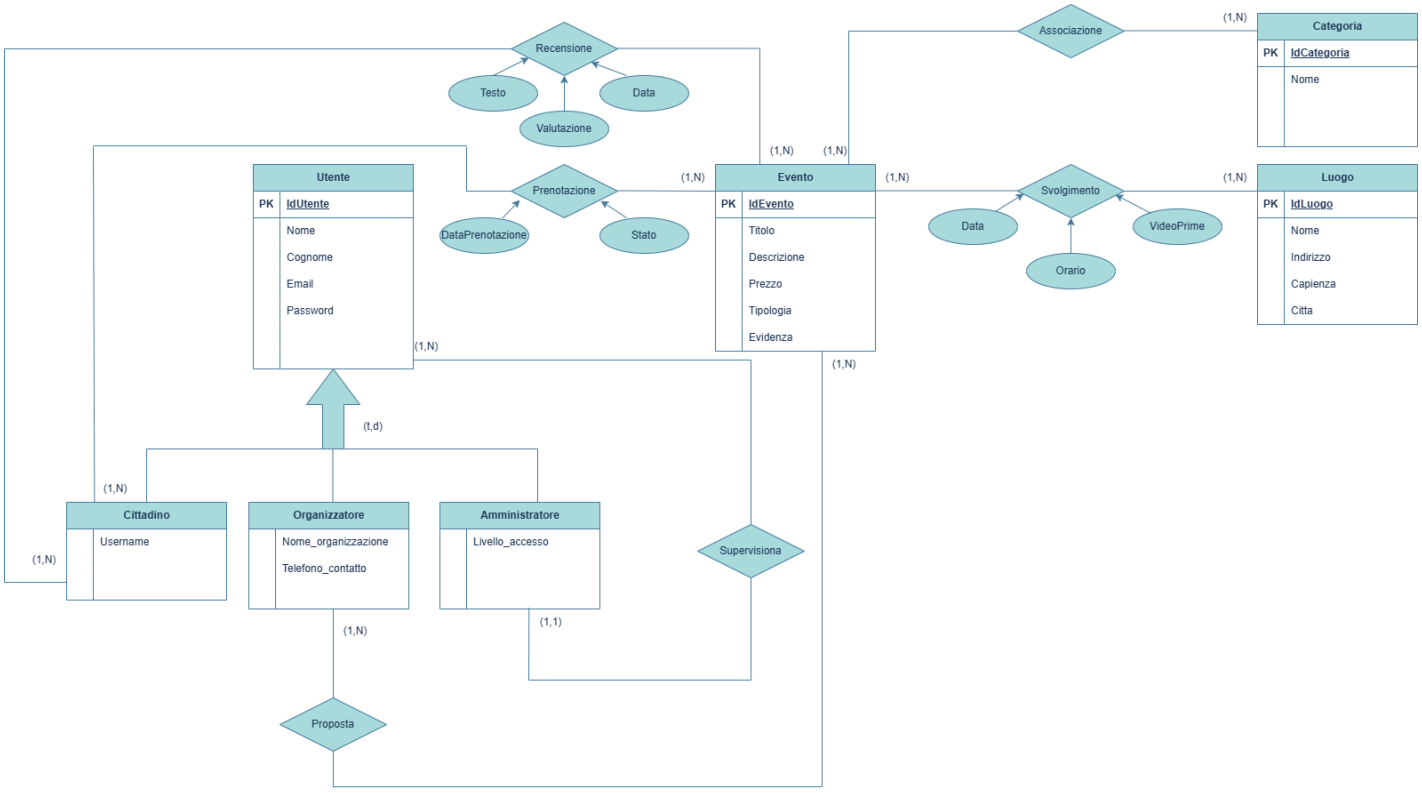
- Definizione dei ruoli e delle competenze di ciascun utente;
- Gestione delle supervisioni: possibilità di collegare gerarchicamente utenti tra loro (es. un amministratore che supervisiona più organizzatori);
- Tracciamento delle modifiche e delle responsabilità per garantire trasparenza.

6. Accesso all’archivio degli eventi passati e valorizzazione del patrimonio culturale

Il sistema manterrà uno storico completo degli eventi svolti, consentendo:

- Consultazione di materiali condivisi (foto, video, brochure);
- Accesso ai commenti e alle recensioni post-evento;
- Valorizzazione della memoria culturale locale e dei dati statistici sugli eventi.

1. Analisi e progettazione concettuale: modello E/R



Primo prototipo del modello E/R con generalizzazione totale disgiunta

Il progetto prevede inizialmente l'utilizzo di una generalizzazione totale e disgiunta per l'entità 'Utente', al fine di modellare correttamente le specializzazioni logiche previste nel dominio applicativo degli eventi culturali locali.

La generalizzazione dell'entità 'Utente' definisce tre sottotipi: 'Cittadino', 'Organizzatore', 'Amministratore'.

- **Totale**: ogni istanza dell'entità 'Utente' deve appartenere obbligatoriamente a una delle tre specializzazioni. Un utente generico non può esistere.
- **Disgiunta**: un utente può essere o cittadino, o organizzatore, o amministratore, ma non più di uno contemporaneamente.

Possibili soluzioni della generalizzazione

1) Strategia "Tutto nel padre"

Questa strategia consiste nell'accentrare tutti gli attributi direttamente nell'entità 'Utente', assorbendo le caratteristiche specifiche delle sottoclassi all'interno della super-entità. Gli attributi delle entità figlie vengono mantenuti nella stessa tabella, e un attributo ruolo viene utilizzato per distinguerli.

- Utente(id_utente, nome, cognome, email, password, ruolo [Cittadino, Organizzatore, Amministratore])
- Per i ruoli ai quali non si applicano, gli attributi assumono valore NULL.

Vantaggi:

- Minor numero di tabelle;
- Modello più semplice da gestire nel database relazionale.

Svantaggi:

- Presenza di numerosi valori NULL;
- Difficoltà nel garantire integrità semantica delle specializzazioni.

2) Strategia "Tutto nelle figlie"

Questa strategia consiste nel modellare tre entità separate (Cittadino, Organizzatore, Amministratore), ognuna delle quali eredita gli attributi comuni della super-entità 'Utente', e gestisce i propri attributi specifici.

Cittadino(id_utente, nome, cognome, email, password, username)

Organizzatore(id_utente, nome, cognome, email, password, nome_organizzazione, telefono_contatto)

Amministratore(id_utente, nome, cognome, email, password, livello_accesso)

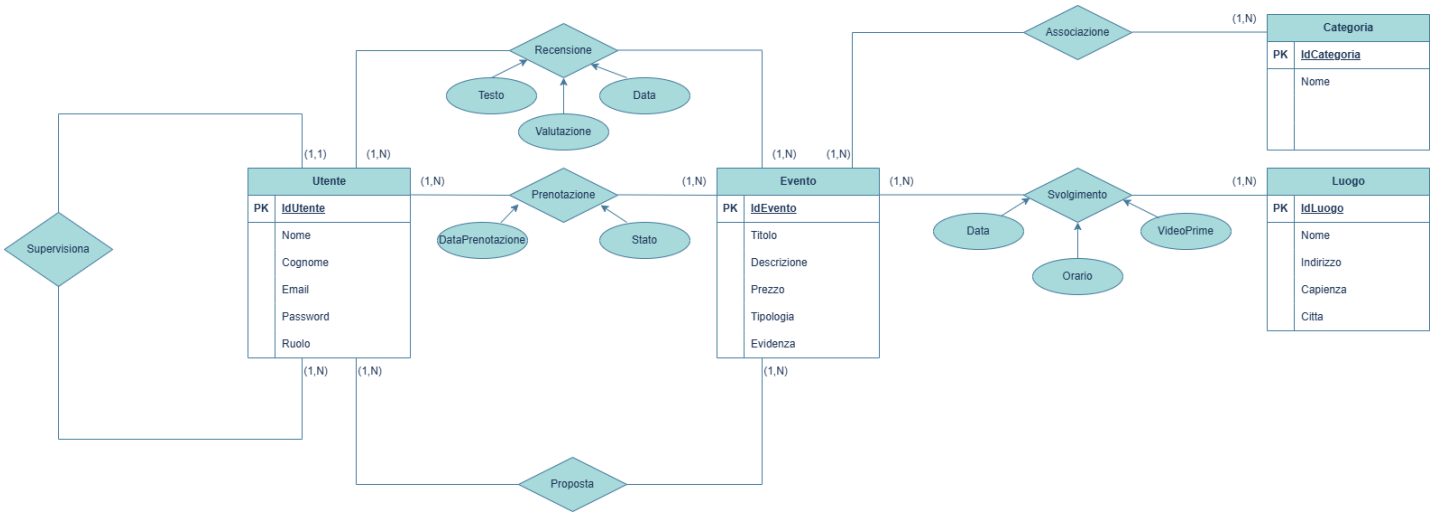
Vantaggi:

- Nessun valore NULL;
- Maggiore chiarezza nella gestione dei dati specifici per ciascun ruolo.

Svantaggi:

- Maggior numero di tabelle;
- Complessità nelle operazioni comuni tra utenti (es. login, gestione credenziali).

Modello E/R Ottimizzato



Nel passaggio al modello E/R ottimizzato e successivamente a quello relazionale, si è optato per abbandonare la generalizzazione in favore di una struttura unificata.

Strategia adottata: una sola entità ‘Utente’ con campo ‘Ruolo’

Tutti gli utenti sono modellati tramite un’unica entità ‘Utente’, arricchita da un attributo ruolo che assume valori all’interno del dominio {Cittadino, Organizzatore, Amministratore}.

Gli attributi specifici dei vari ruoli vengono mantenuti nella stessa tabella, con l’accettazione di valori NULL per quelli non pertinenti al ruolo assegnato.

Struttura finale:

- Utente(id_utente, nome, cognome, email, password, ruolo [Cittadino, Organizzatore, Amministratore])

Motivazioni della scelta progettuale:

1. Semplificazione logica e gestionale
    - Tutti gli utenti vengono gestiti all'interno di una sola tabella;
    - Si riducono le join tra sottotipi, semplificando query e manutenzione.
2. Uniformità nei dati anagrafici
    - Le informazioni base (nome, email, telefono) sono comuni a tutti i ruoli;
    - Non è necessario replicare dati condivisi su più entità.
3. Maggiore flessibilità ed estensibilità
    - Il sistema può supportare facilmente l'introduzione di nuovi ruoli o funzionalità;
    - La distinzione dei ruoli può essere facilmente gestita lato applicativo.

2. Progettazione logica

La progettazione logica del database è stata realizzata sulla base del modello concettuale E/R precedentemente descritto, adottando un approccio unificato tramite l'attributo discriminante 'Ruolo' per l'entità 'Utente', evitando quindi l'uso di generalizzazioni.

Utente (IdUtente, Nome, Cognome, Email (UNIQUE), Password, Ruolo [Cittadino, Organizzatore, Amministratore]):

- L'entità 'Utente' contiene i dati anagrafici comuni a tutte le tipologie di utenti del sistema.
- Il campo 'Ruolo' discrimina tra le tre categorie logiche previste: cittadini (partecipanti), organizzatori (proponenti e gestori eventi) e amministratori (supervisori del sistema).
- 'Email' è unica e funge da riferimento per l'autenticazione.

Evento (IdEvento, Titolo, Descrizione, Prezzo, Evidenza (BOOLEAN)):

- L'entità 'Evento' rappresenta un'attività culturale.
- Il campo 'Evidenza' indica se l'evento è messo in risalto nella sezione promozionale.
- Ogni evento può avere più svolgimenti in luoghi differenti, ed essere associato a più categorie tematiche.

Prenotazione (IdUtente : Utente, IdEvento : Evento, Stato [richiesta, confermata, annullata], DataPrenotazione):

- I cittadini prenotano eventi, con tracciamento dello stato della prenotazione.
- Relazione molti-a-molti tra cittadini ed eventi, tramite la tabella 'Prenotazione'.

Recensione (IdUtente : Utente, IdEvento : Evento, Testo, Valutazione (1–5)):

- I cittadini possono recensire gli eventi a cui hanno partecipato.
- È possibile consultare le recensioni pubblicate da altri utenti.

Proposta (IdUtente : Utente, IdEvento : Evento):

- Gli organizzatori propongono eventi, che potranno poi essere approvati o supervisionati dagli amministratori.
- La relazione 'Proposta' permette la gestione dei diritti di modifica o aggiornamento eventi da parte dell'organizzatore.

Luogo (IdLuogo, Nome, Indirizzo, Capienza, Città):

- I luoghi possono ospitare eventi. La capienza è usata per verificare la disponibilità dei posti.

Svolgimento (IdEvento : Evento, IdLuogo : Luogo, Data, Orario, VideoPrime (URL)):

- Rappresenta il calendario effettivo dell'evento, inclusi gli orari e il luogo.
- Permette anche il collegamento a contenuti video condivisi dopo l'evento.

Categoria (IdCategoria, Nome):

- Le categorie tematiche servono a organizzare gli eventi per interessi (es. musica, arte, scienza...).

Associazione (IdEvento : Evento, IdCategoria : Categoria):

- Relazione molti-a-molti tra eventi e categorie.
- Un evento può appartenere a più categorie, e ogni categoria può includere più eventi.

Supervisiona (IdSupervisore : Utente, IdSupervisionato : Utente):

- Relazione gerarchica tra amministratori e utenti (cittadini o organizzatori).
- Ogni utente può essere supervisionato da un solo amministratore.
- Un amministratore può supervisionare più utenti.

Entità e Relazioni

Utente

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdUtente	Identificativo univoco dell'utente	INT	-	PK, NOT NULL
Nome	Nome dell'utente	VARCHAR	50	NOT NULL
Cognome	Cognome dell'utente	VARCHAR	50	NOT NULL
Email	Email dell'utente	VARCHAR	100	NOT NULL, UNIQUE
Password	Password utente (hashata)	VARCHAR	255	NOT NULL
Ruolo	Ruolo dell'utente	ENUM	-	Valori ammessi: 'Cittadino', 'Organizzatore', 'Amministratore'; NOT NULL

Evento

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdEvento	Identificativo evento	INT	-	PK, NOT NULL
Titolo	Titolo dell'evento	VARCHAR	100	NOT NULL
Descrizione	Descrizione dettagliata	TEXT	-	NOT NULL

Data	Data dell'evento	DATE	-	NOT NULL
Evidenza	Se evento è in evidenza	BOOLEAN	-	DEFAULT FALSE

Luogo

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdLuogo	ID del luogo	INT	-	PK, NOT NULL
Nome	Nome del luogo	VARCHAR	100	NOT NULL
Indirizzo	Indirizzo completo	VARCHAR	200	NOT NULL
Città	Città del luogo	VARCHAR	100	NOT NULL
Capienza	Numero posti	INT	-	CHECK > 0

Categoria

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdCategoria	Identificativo categoria	INT	-	PK, NOT NULL
Nome	Nome categoria	VARCHAR	50	NOT NULL, UNIQUE

Prenotazione

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdUtente	Utente che prenota	INT	-	PK, FK → UTENTE(IdUtente), NOT NULL
IdEvento	Evento prenotato	INT	-	PK, FK → EVENTO(IdEvento), NOT NULL
Stato	Stato della prenotazione	ENUM	-	Valori: 'richiesta', 'confermata', 'annullata'; NOT NULL

Recensione

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdUtente	Autore della recensione	INT	-	PK (composita), FK, NOT NULL
IdEvento	Evento recensito	INT	-	PK (composita), FK, NOT NULL
Testo	Testo della recensione	TEXT	-	NOT NULL
Valutazione	Valutazione numerica	INT	-	CHECK (1 ≤ Valutazione ≤ 5), NOT NULL

Proposta

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdUtente	Organizzatore	INT	-	PK, FK → UTENTE(IdUtente), NOT NULL, ruolo='Organizzatore'
IdEvento	Evento proposto	INT	-	PK, FK → EVENTO(IdEvento), NOT NULL

Associazione

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdEvento	Evento	INT	-	PK, FK → EVENTO(IdEvento)
IdCategoria	Categoria	INT	-	PK, FK → CATEGORIA(IdCategoria)

Svolgimento

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdEvento	Evento	INT	-	PK, FK → EVENTO(IdEvento)

IdLuogo	Luogo	INT	-	PK, FK → LUOGO(IdLuogo)
Data	Data dello svolgimento	DATE	-	NOT NULL
Orario	Orario	TIME	-	NOT NULL

Supervisiona

Nome Campo	Descrizione	Tipo Dati	Lunghezza	Vincoli
IdSupervisore	Supervisore (Amministratore)	INT	-	FK → UTENTE(IdUtente), ruolo='Amministratore', NOT NULL
IdSupervisionato	Supervisionato (Citt/Org)	INT	-	PK, FK → UTENTE(IdUtente), NOT NULL

Descrizione dei Vincoli

- **Integrità Referenziale**

Tipo: Vincolo di Interrelazione

Descrizione: Le tabelle che rappresentano relazioni tra entità principali (specialmente quelle molti-a-molti) includono vincoli di integrità referenziale tramite chiavi esterne (foreign key), che garantiscono la coerenza tra le tabelle. Ogni chiave esterna punta a una chiave primaria esistente, impedendo riferimenti a entità inesistenti. In particolare, i vincoli di integrità referenziale sono presenti nelle seguenti tabelle: ‘Prenotazione’, ‘Recensione’, ‘Supervisiona’, ‘Associazione’, ‘Svolgimento’.

- **Integrità delle Chiavi Primarie**

Tipo: Vincolo di Chiave

Descrizione: Ogni tabella del modello presenta una chiave primaria (PK), la quale garantisce l’unicità dei record. Le chiavi primarie non possono contenere valori nulli, in conformità alle regole di integrità del database relazionale. Questo assicura che ogni entità o relazione sia identificabile in modo univoco

- **Vincolo sul Ruolo dell’Utente**

Tipo: Vincolo di Dominio

Descrizione: L'attributo 'Ruolo' della tabella 'Utente' può assumere esclusivamente uno dei seguenti valori:{Cittadino, Organizzatore, Amministratore}

- **Capienza del Luogo > 0**

Tipo: Vincolo di Dominio

Descrizione: L'attributo Capienza della tabella 'Luogo' deve rispettare la condizione: Capienza > 0. Viene così evitata la definizione di spazi non utilizzabili o logicamente errati (capienza nulla o negativa).

- **Valutazione nella Recensione (1–5)**

Tipo: Vincolo di Dominio + Tupla

Descrizione: L'attributo 'Valutazione' nella tabella 'Recensione' deve assumere un valore compreso nell'intervallo [1, 5].

- **Evento in Evidenza**

Tipo: Vincolo di Interrelazione (semantico)

Descrizione: Un evento può essere contrassegnato come in evidenza (Evidenza = TRUE) solo se è effettivamente supervisionato da almeno un amministratore, cioè se esiste una corrispondente entry nella tabella 'Supervisiona'.

- **Prevenzione Prenotazioni Duplicate**

Tipo: Vincolo di Chiave

Descrizione: La tabella 'Prenotazione' utilizza una chiave primaria composta: (IdUtente, IdEvento). Questo impedisce che uno stesso utente effettui più prenotazioni per lo stesso evento, garantendo l'unicità del binomio utente-evento.

- **Condizione per 'Recensione'**

Tipo: Vincolo di Interrelazione

Descrizione: Un utente può inserire una recensione per un evento solo se lo ha prenotato e ha partecipato (indicato da un attributo Stato = 'Confermato' nella prenotazione).
Questo vincolo semantico impedisce recensioni arbitrarie da parte di utenti non coinvolti.

Se un utente ha ruolo ‘Amministratore’, può inserire una recensione solo se è anche classificato come ‘Cittadino’.

- **Validazione dell’Organizzatore**

Tipo: Vincolo di Interrelazione + Dominio

Descrizione: Ogni riferimento a IdUtente nella relazione ‘Proposta’ deve puntare a un utente il cui Ruolo = 'Organizzatore'.
Questo garantisce che solo utenti autorizzati possano proporre eventi.

- **Validazione della Supervisione**

Tipo: Vincolo di Interrelazione + Tupla (semantico)

Descrizione: Nella tabella ‘Supervisiona’, l’utente che esercita la supervisione deve avere Ruolo = 'Amministratore', mentre il supervisionato deve essere un Organizzatore.
Si tratta di una relazione ricorsiva ma vincolata ai ruoli, che riflette una gerarchia interna tra utenti. L’attributo IdSupervisore può essere NULL quando un utente non è supervisionato. Questo consente la presenza di utenti non legati gerarchicamente ad altri (es. un organizzatore indipendente).

Vincolo	**Tipo**	**Descrizione**
IdSupervisore IS NOT NULL	Tupla (se supervisiona)	Un'organizzazione che supervisiona deve avere un amministratore. Se la tupla esiste, non può avere IdSupervisore = NULL.
Ruolo(IdSupervisore) = 'Amministratore'	Interrelazione semantica	Il supervisore deve essere un utente con ruolo "Amministratore".
Ruolo(IdSupervisionato) = 'Organizzatore'	Interrelazione semantica	Il supervisionato deve essere un utente con ruolo "Organizzatore".
IdSupervisore ≠ IdSupervisionato	Tupla	Un utente non può supervisionare sé stesso.
PK composta (IdSupervisore, IdSupervisionato)	Chiave	Impedisce supervisione doppia tra le stesse due persone.
FK su entrambi gli ID verso UTENTE	Interrelazione	Garantisce integrità referenziale.

- **Unicità della Categoria**

Tipo: Vincolo di Chiave (Unique)

Descrizione: L'attributo 'Nome' nella tabella 'Categoria' deve essere univoco. Questo previene la duplicazione di categorie con denominazioni identiche o ambigue.

- **Appartenenza a più Categorie**

Tipo: Vincolo di Interrelazione + Chiave

Descrizione: Gli eventi possono appartenere a più categorie tematiche grazie alla relazione 'Associazione', che implementa una relazione molti-a-molti (N:N). Questo consente un sistema di classificazione flessibile e ricco.

- **Utilizzo di Chiavi Composte**

Tipo: Vincolo di Chiave

Descrizione: Le tabelle relazionali derivate da relazioni N:N o da entità deboli utilizzano chiavi primarie composte.

- **Attributi NOT NULL**

Tipo: Vincolo di Tupla

Descrizione: Oltre alle chiavi primarie (che per definizione non possono contenere valori nulli), anche altri attributi essenziali come Nome, Cognome, Email, Titolo, Descrizione, Data, Orario, Stato ecc. devono essere definiti come NOT NULL, al fine di garantire la completezza e la coerenza informativa del database.

Semplificazioni e Scelte Progettuali

- Materiali Promozionali: sono gestiti in modo parziale attraverso l'attributo VideoPrime, che rappresenta eventuali materiali promozionali o documentativi (locandine, video, brochure).
- Archiviazione Storica: è implementata implicitamente tramite attributi come Data (per eventi, prenotazioni, svolgimenti) e Stato (es. "Confermato", "Annullato"), che consentono il tracciamento delle modifiche senza tabelle di log separate.

3. Implementazione Sistema Informativo

L'implementazione del sistema informativo è stata realizzata utilizzando il framework Django, che consente di sviluppare un'applicazione web scalabile, sicura e facilmente mantenibile. Il sistema è progettato per gestire i dati secondo il modello logico precedentemente definito e per offrire accesso a un insieme di funzionalità chiave, rivolte a diverse categorie di utenti: cittadini, organizzatori e amministratori. L'applicazione supporta almeno quattro funzionalità principali, selezionate in base alle esigenze cliniche e organizzative, garantendo una gestione efficace e intuitiva.

Tra le funzionalità realizzabili, si possono includere:

- Visualizzazione del calendario eventi con dettaglio.
- Registrazione e autenticazione degli utenti. I cittadini possono registrarsi autonomamente creando un account personale tramite un modulo dedicato e successivamente possono effettuare il login per accedere alle funzionalità riservate. Gli organizzatori e l'amministratore comunale accedono solo tramite login, con credenziali già definite e riportate nel database.
- Creazione, modifica e approvazione di eventi.
- Prenotazione di posti e generazione biglietti.
- Inserimento e consultazione di recensioni degli utenti (con ruolo 'Cittadino'). Modifica, aggiornamento e inserimento degli eventi da parte dell'utente con ruolo 'Organizzatore'. L'utente con ruolo 'Amministratore' può supervisionare eventi organizzati da utenti con ruolo "Organizzatore" e può contrassegnare un evento come "in evidenza" solo se è stato effettivamente supervisionato.
- Archivio degli eventi passati con materiali allegati.
- Ricerca eventi per categoria, parola chiave o luogo.
- Visualizzazione delle sedi con mappa e disponibilità.
- Area personale con storico recensioni.

Homepage

La homepage rappresenta la vetrina pubblica della piattaforma **Agora Cultura**, accessibile anche senza autenticazione.

Contenuti principali:

- **Presentazione della piattaforma**: messaggio di benvenuto che introduce l'obiettivo del portale: facilitare l'accesso e la partecipazione ad eventi culturali locali.
- **Accesso Area Riservata**: pulsanti che indirizzano l'utente verso il login o la registrazione.
- **Calendario Eventi**: accesso rapido alla sezione contenente il calendario completo degli eventi disponibili.
- **Recensioni e Area Personale**: indicazioni sulle funzionalità riservate agli utenti autenticati, come lasciare recensioni e consultare lo storico personale.

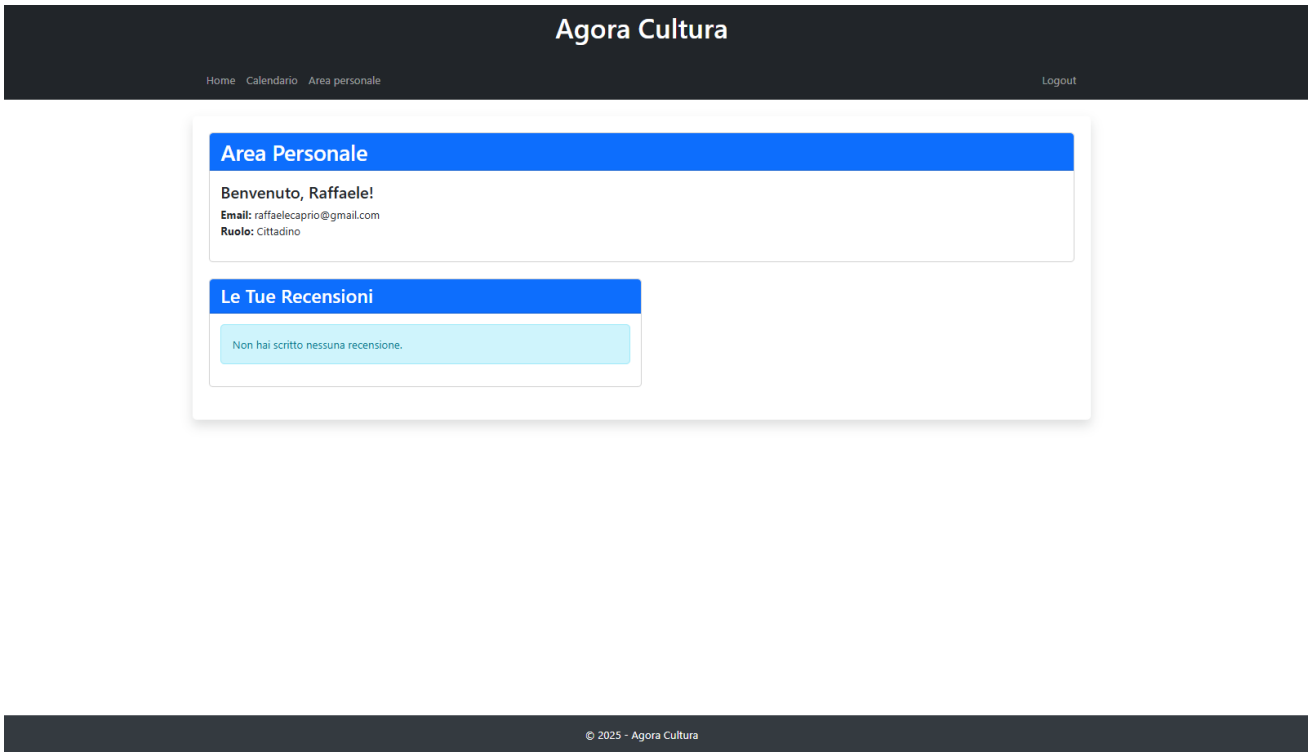
Area riservata – Gestione Autenticata (con sessioni)

La piattaforma integra il sistema di autenticazione e gestione delle sessioni di Django, differenziando i permessi in base al ruolo dell'utente. I cittadini possono registrarsi autonomamente creando un account personale tramite un modulo dedicato. Successivamente possono effettuare il login per accedere alle funzionalità riservate.
L'organizzatore e l'amministratore comunale accede solo tramite login, con credenziali già definite e riportate nel database.

Login Cittadino

L'utente cittadino ha accesso a un'area personale dedicata, con funzionalità mirate all'esperienza culturale:

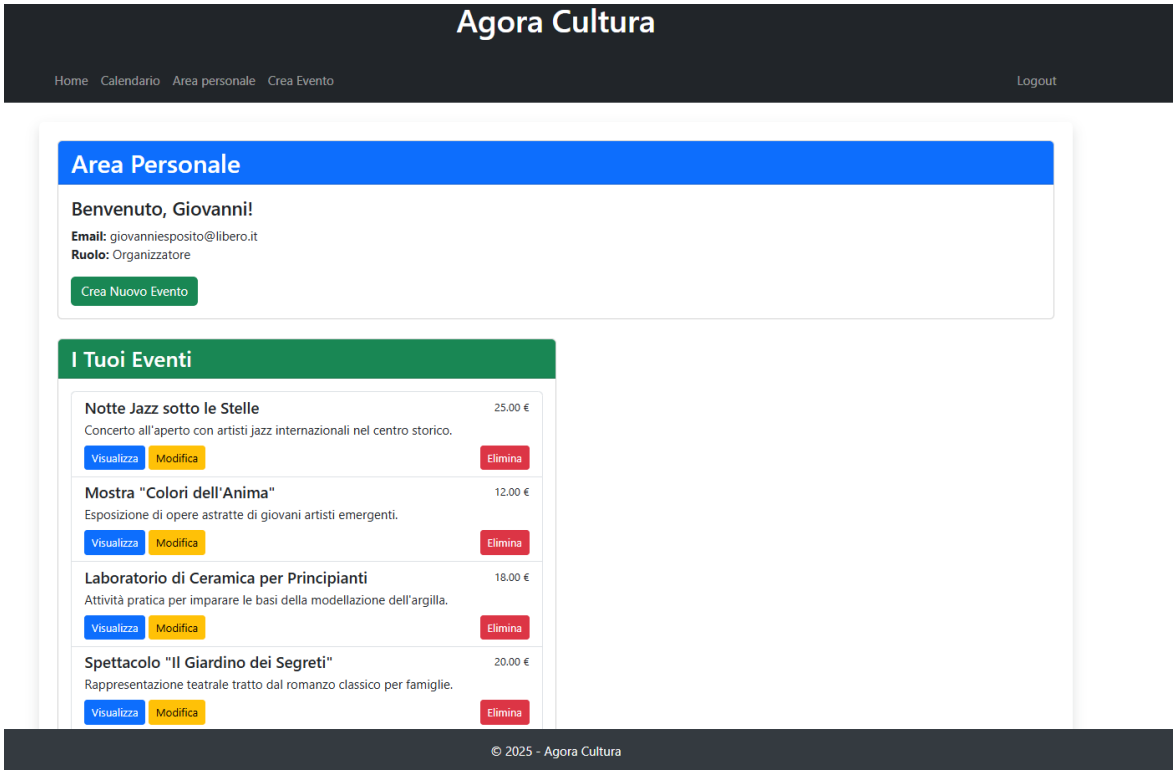
- Visualizzazione calendario eventi: può esplorare eventi culturali, visionare orari e dettagli.
- Inserimento recensioni: può lasciare recensioni per eventi a cui ha partecipato.
- Storico recensioni: consultazione e gestione delle recensioni già inserite.



Login Organizzatore

L’organizzatore ha accesso a strumenti di gestione degli eventi:

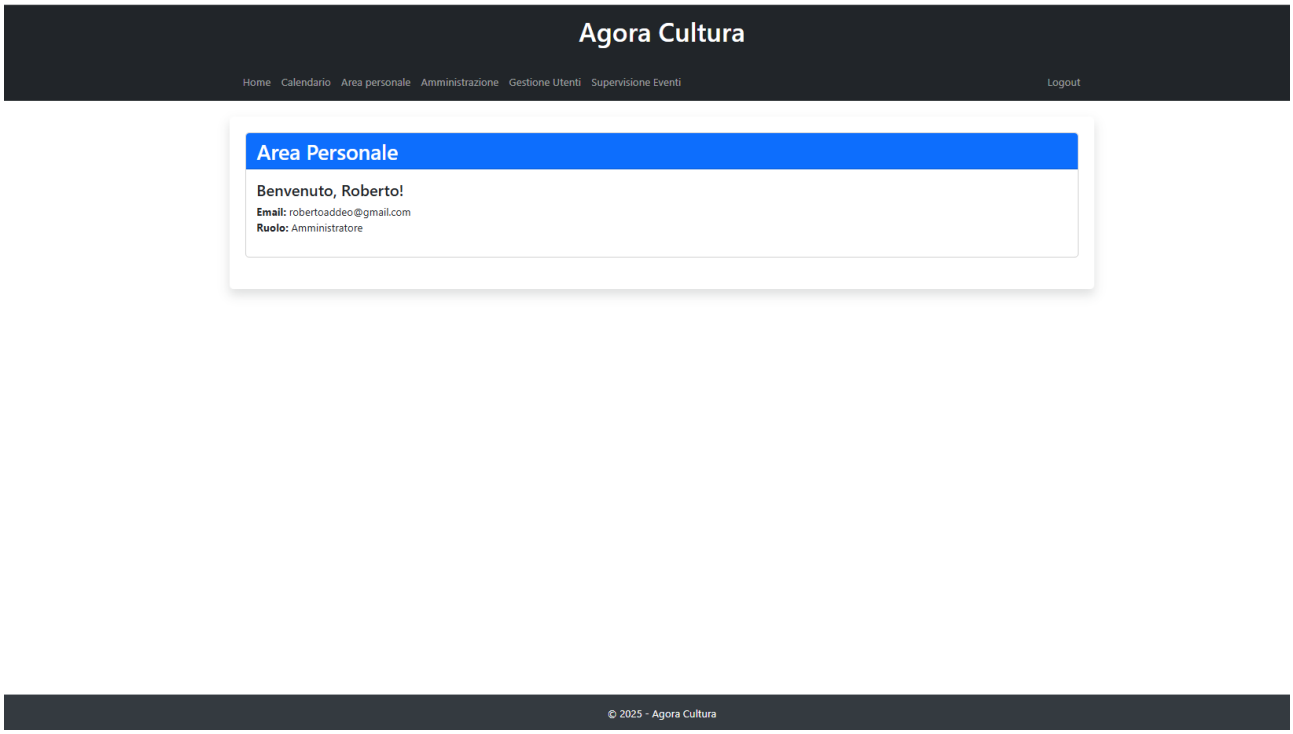
- Proposta nuovi eventi: possibilità di creare e proporre un nuovo evento culturale.
- Modifica eventi esistenti: può aggiornare titolo, descrizione, luogo e data degli eventi proposti.



Login Amministratore

L’amministratore ha accesso completo alle funzioni di supervisione e controllo qualità:

- Supervisione eventi: può supervisionare eventi organizzati da utenti con ruolo ‘Organizzatore’.
- Evidenziazione eventi: può contrassegnare un evento come "in evidenza" solo se è stato effettivamente supervisionato.



Gestione delle sessioni

Nel progetto, le sessioni svolgono un ruolo fondamentale nella gestione dell'autenticazione e dell'autorizzazione degli utenti, ovvero nel mantenere lo stato di login durante la navigazione tra le varie pagine dell'applicazione web. Questo è particolarmente importante perché HTTP è un protocollo stateless, quindi senza un meccanismo come le sessioni, ogni richiesta sarebbe trattata come indipendente e anonima.

Django fornisce un sistema di autenticazione integrato che gestisce login, logout, registrazione, controllo dei permessi e gestione delle credenziali. Tuttavia la scelta di non usare il sistema *auth_user* di Django è motivata dalla necessità di gestire tre ruoli distinti con attributi, metodi e interfacce differenti. Mantenere modelli separati per ciascun tipo di utente (Cittadino, Organizzatore, Amministratore) consente un’organizzazione del codice più chiara.

```
            if user.ruolo == 'Cittadino':
                request.session['cittadino_email'] = user.email
            elif user.ruolo == 'Organizzatore':
                request.session['organizzatore_email'] = user.email
            elif user.ruolo == 'Amministratore':
                request.session['amministratore_email'] = user.email
            return redirect('home')
```

In questo modo, ogni volta che l'utente farà una nuova richiesta, il sistema riconoscerà la sua sessione attiva tramite questa informazione, ciò impedisce che un utente autenticato come 'Cittadino' possa accedere per errore o malintenzionatamente alle funzioni dedicate all''Organizzatore'.

Esempio login

```
def autenticazione_cittadino(email, password):  2 usages  new *
    try:
        user = Utente.objects.get(email=email, ruolo='Cittadino')
        password_md5 = hashlib.md5(password.encode()).hexdigest()
        if password_md5 == user.password:
            return user
    except Utente.DoesNotExist:
        pass
    return None
```

- Con questa funzione viene verificato se esiste un cittadino nel database con l'email e la password specificate, essa viene ripresa nella funzione di login a seguire.

```
def login_cittadino(request):  1 usage  new *
    if request.method == 'POST':
        email = request.POST.get('username')  # L'email viene inserita nel campo username
        password = request.POST.get('password')
        user = autenticazione_cittadino(email, password)

        # Se esiste un cittadino con email e password corrispondenti
        if user is not None:
            # Crea una sessione per l'utente inserendo l'email nella chiave 'cittadino_email'
            login(request, user)
            request.session['cittadino_email'] = user.email
            # Reindirizza all'area riservata cittadino
            return redirect('area_riservata_cittadino')
    else:
        form = AuthenticationForm()
    return render(request, template_name: 'login.html', context: {'form': form})
```

- Gestione del processo di login del cittadino, verificando le credenziali e creando la sessione. La funzione accetta solo richieste di tipo POST per il login. Se la richiesta non è POST (es. accesso diretto tramite URL), mostra la pagina di login senza effettuare alcuna verifica. Prende l'email inserita dall'utente tramite *request.POST.get('email').* Prende la password inserita tramite *request.POST.get('password').*
  Se esiste un cittadino con email e password corrispondenti viene creata una sessione per l'utente inserendo l'email nella chiave 'cittadino_email'. L'utente viene reindirizzato all'area riservata paziente *redirect('area_riservata_cittadino').* Altrimenti viene mostrata nuovamente la pagina di login.

```
def area_riservata_cittadino(request):  1 usage  new *
    # Controlla che l'utente è ancora in sessione
    if 'cittadino_email' not in request.session:
        # Se la chiave non è presente, significa che l'utente non è autenticato
        return redirect('login_cittadino')
```

- Se la chiave non è presente, significa che l'utente non è autenticato e viene reindirizzato alla pagina di login. Questo evita accessi non autorizzati.

- Quando l'utente decide di uscire (logout), la sessione viene completamente svuotata tramite:

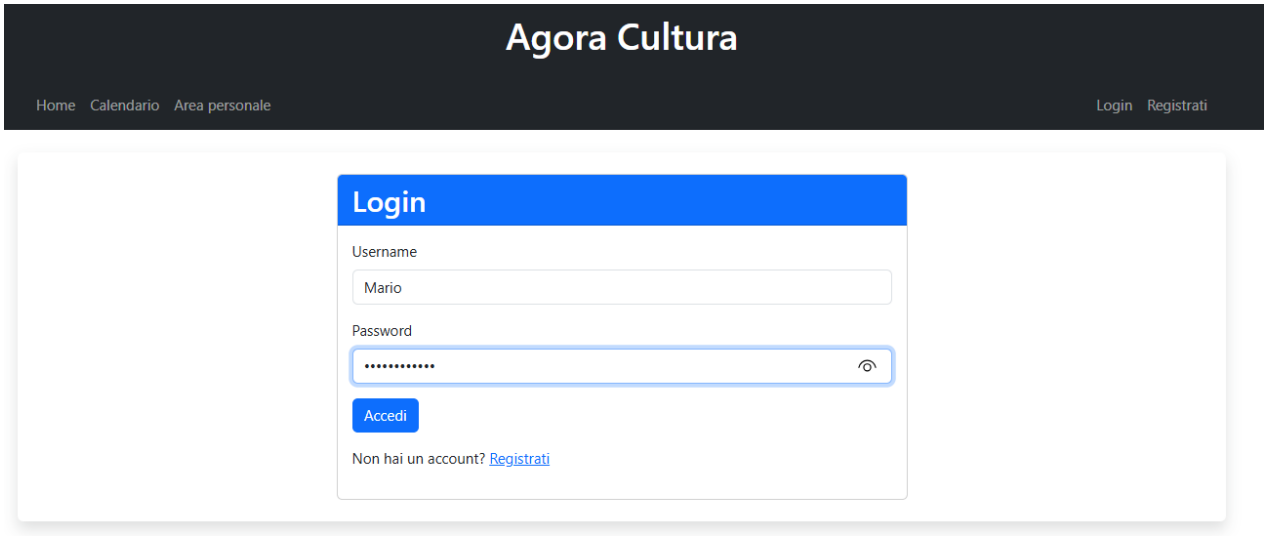
```
def logout_cittadino(request):
    request.session.flush()

    # Disconnetti l'utente
    logout(request)
    return redirect('home')
```

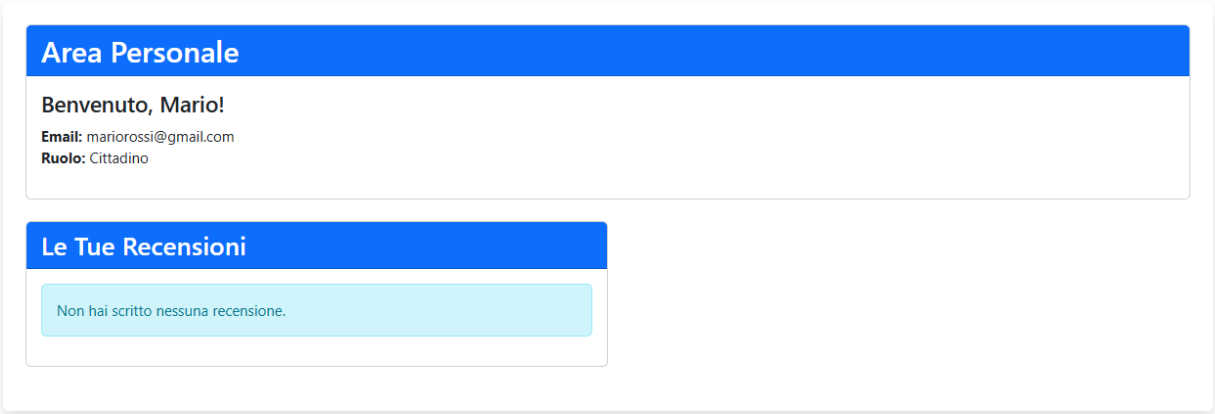
- Ciò comporta la cancellazione di tutte le informazioni di sessione memorizzate, invalidando la sessione attuale.

Esempio pratico

Accedo come cittadino:



Ho già un account valido, le credenziali inserite sono corrette, vengo reindirizzato nella mia area personale:



Implementazione SQL Injection

La SQL Injection (SQLi) è una delle vulnerabilità di sicurezza web più diffuse e pericolose. Consiste nell'inserimento di comandi SQL malevoli all'interno dei campi di input dell'applicazione, con l'obiettivo di manipolare le query inviate al database. Uno degli scopi più comuni di questo attacco è l'estrazione non autorizzata di dati sensibili. L'attacco SQLi si basa spesso sull'interruzione anticipata di una stringa SQL seguita da un'iniezione di codice. Per ignorare la parte restante della query, si utilizza il carattere di commento --, che fa sì che tutto ciò che segue venga ignorato dal database durante l'esecuzione.

Per simulare una SQL injection nel progetto, è stata creata una funzione di login da sostituire a quella originale per testarla:

```python
@csrf_exempt  1 usage  new *
def login_vulnerabile_cittadino(request):
    if request.method == "POST":
        email = request.POST.get('email')
        password = request.POST.get('password')
        #query SQL vulnerabile
        query = f"SELECT * FROM Agora_Cultura_cittadino WHERE email = '{email}' AND password = '{password}'"
        with connection.cursor() as cursor:
            cursor.execute(query)
            row = cursor.fetchone()
            if row:
                # Converte il risultato in un dizionario per visualizzarlo nel template
                columns = [col[0] for col in cursor.description]
                user_dict = dict(zip(columns, row))
                return render(request, template_name: 'area_riservata_cittadino.html', context: {'cittadino': user_dict})
            else:
                return render(request, template_name: 'login_vulnerabile_cittadino.html', context: {'error': 'Credenziali non valide.'})
    else:
        return render(request, template_name: 'login_vulnerabile_cittadino.html')
```

Le variabili email e password, ricevute da una richiesta POST, sono inserite direttamente nella query SQL senza alcun filtro, rendendo il codice vulnerabile a SQL injection.

- SELECT * FROM Agora_Cultura_cittadino WHERE email = '{email}' AND password = '{password}'.

Il decoratore *@csrf_exempt* disattiva la protezione CSRF di Django, consentendo l'invio di richieste POST anche da strumenti automatizzati come sqlmap.

Cursor ci permette di puntare al database, tramite cui possiamo eseguire i comandi SQL e recuperare dati. Viene instaurata la connessione, per eseguire la query è usato *execute*, in seguito i dati sono recuperati con *fetchone* che prende la prima riga del risultato. Se è trovato un risultato valido dopo l'esecuzione della query *row* conterrà i dati attesi e si accederà alla pagina prevista.

Per analizzare la vulnerabilità della pagina è stato utilizzato "Sqlmap", esso ha confermato in modo definitivo che il parametro email del modulo di login è vulnerabile a SQL Injection.

Come input nel prompt dei comandi è stato inviato questo:

- python sqlmap.py -u "http://127.0.0.1:8000/login_vulnerabile_paziente/" --data "email=test&password=test" --batch –dbs

L'analisi ha rivelato che il parametro email è vulnerabile a questi tipi di SQL Injection:

1) **Boolean (blind)-based:** Modifica la logica della query per ottenere risposte vere o false. Esempio:' OR 1=1- -

```
[15:19:08] [INFO] POST parameter 'email' appears to be 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)' injectable
```

2) **Error-based:** Estrae dati sfruttando i messaggi di errore del database.
   Esempio: ' OR (SELECT 1 FROM (SELECT COUNT(*), CONCAT(..., FLOOR(RAND()))) x)- -

```
[15:19:09] [INFO] POST parameter 'email' is 'MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
```

3) **Stacked queries**: Permette l'esecuzione di più query nello stesso comando, separandole con il ;.
   Esempio: '; SELECT SLEEP(5)#

```
[15:19:19] [INFO] POST parameter 'email' appears to be 'MySQL >= 5.0.12 stacked queries (comment)' injectable
```

4) **Time-based blind:** Deduce informazioni basandosi sul ritardo nella risposta del server, utilizzando comandi come SLEEP().
   Esempio: ' AND SLEEP(5)—

```
[15:19:29] [INFO] POST parameter 'email' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
```

È stata creata anche una pagina HTML dedicata per testare il login vulnerabile. Per facilitare l'iniezione, il campo email è stato reso un campo di tipo testo semplice, e si è usato l'attributo *novalidate* per disattivare i controlli automatici del browser (come il *required* o la validazione dell'email):

```
{% extends 'base.html' %}

{% block title %}Login Vulnerabile - Agora Cultura{% endblock %}

{% block content %}
<div class="row justify-content-center">
    <div class="col-md-6">
        <div class="card">
            <div class="card-header bg-danger text-white">
                <h2 class="mb-0">Login Vulnerabile (SQL Injection)</h2>
            </div>
            <div class="card-body">
                {% if error %}
                    <div class="alert alert-danger">
                        {{ error }}
                    </div>
                {% endif %}

                <div class="alert alert-warning">
                    <h4 class="alert-heading">ATTENZIONE: Pagina Vulnerabile!</h4>
```

```
                    </ul>
                    <p>Questo bypassa l'autenticazione e accede come primo utente nel database.</p>
                </div>

                <form method="post" novalidate>
                    <div class="mb-3">
                        <label for="id_email" class="form-label">Email</label>
                        <input type="text" name="email" class="form-control" id="id_email">
                    </div>
                    <div class="mb-3">
                        <label for="id_password" class="form-label">Password</label>
                        <input type="password" name="password" class="form-control" id="id_password">
                    </div>
                    <button type="submit" class="btn btn-danger">Accedi (Vulnerabile)</button>
```

```
                    <button type="submit" class="btn btn-danger">Accedi (Vulnerabile)</button>
                </form>

                <div class="mt-3">
                    <a href="{% url 'login' %}" class="btn btn-sm btn-primary">
                        <i class="fas fa-lock"></i> Torna al login sicuro
                    </a>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

- L'uso di *novalidate* consente l'invio del form anche con campi vuoti o valori non conformi.

Esempio di attacco SQL Injection

Selezionando il login vulnerabile del cittadino, possiamo provare ad entrare nel sistema usando come email: *' OR '1'='1'* -- e come password un campo qualsiasi.

Cliccando su *Accedi vulnerabile* saremo indirizzati verso l'area personale di un cittadino.

Passando il campo *' OR '1'='1'* - - nell'email la query si trasforma in:

- SELECT * FROM Agora_Cultura_cittadino WHERE email = '' OR '1'='1' AND password = '' OR '1'='1';

Poiché '1'='1' è sempre vera, il database restituirà comunque dei risultati e l'utente verrà autenticato senza credenziali valide, accedendo così all'area riservata.

SQL Injection Soluzione

Nel progetto di Agora Cultura, realizzato con Django, il rischio di SQL Injection è mitigato in modo nativo grazie all'utilizzo dell'ORM (Object-Relational Mapping) di Django, che genera query SQL in modo sicuro, escludendo automaticamente input potenzialmente pericolosi.

Inoltre per accedere all'area riservata la funzione viene imposta in questo modo:

```
def area_riservata_cittadino(request):
    if 'cittadino_email' in request.session:
        email = request.session['paziente_email']
        paziente = Cittadino.objects.get(email=email)
        return render(request, 'area_riservata_cittadino.html', {'cittadino': cittadino})
    else:
        return redirect('login_cittadino')
```

- L'uso di Cittadino.objects.get(email=email) genera una query sicura e parametrizzata. Inoltre viene fatto un controllo di sessione come specificato sopra.

Per la protezione delle credenziali degli utenti, le password non vengono mai memorizzate in chiaro nel database. Durante il login, la password inserita viene trasformata in una stringa hash mediante l'algoritmo MD5, aumentando ulteriormente la sicurezza:

```
password = request.POST.get('password')
hashed_password = hashlib.md5(password.encode()).hexdigest()
```

Sommario

1. **Introduzione al progetto**
    - Obiettivi del progetto
2. **Progettazione del database**
    - Analisi e progettazione concettuale: modello E/R
    - Strategie di generalizzazione
    - Modello E/R ottimizzato
3. **Progettazione logica**
    - Entità e relazioni
    - Vincoli
4. **Implementazione del sistema informativo**
    - Struttura tecnica e framework (Django)
    - Funzionalità implementate
    - Area pubblica e homepage
    - Area riservata:
        - Login Cittadino
        - Login Organizzatore
        - Login Amministratore
    - Gestione delle sessioni
5. **Sicurezza del sistema**
    - Esempio di SQL Injection
    - Simulazione di attacco
    - Contromisure adottate