



## ***Project-Museum***

*Project Museum* è un'applicazione web full-stack sviluppata per la gestione informatizzata di un museo. Il sistema consente la visualizzazione, l'inserimento e l'amministrazione di mostre, opere d'arte, clienti e utenti.

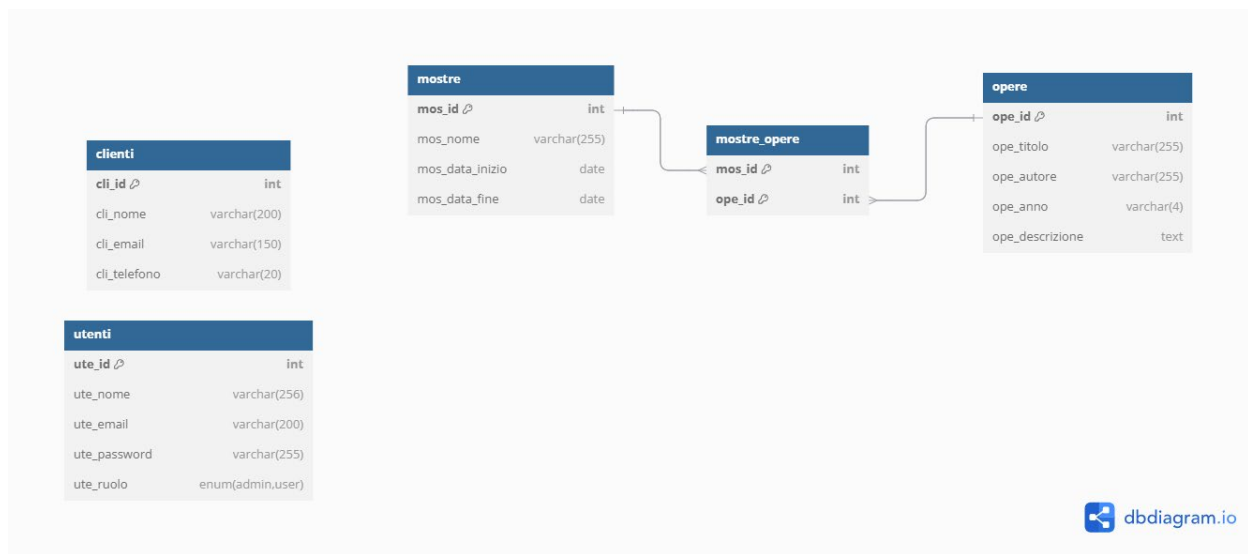
Include funzionalità avanzate come l'autenticazione con ruoli, l'utilizzo di API RESTful per l'interscambio dati, e una dashboard responsive pensata per offrire un'interfaccia semplice ed efficace sia all'amministratore che all'utente finale. Il progetto è stato pubblicato online su InfinityFree per dimostrare l'accessibilità e il corretto funzionamento in ambiente di produzione.

### ✓ **Tecnologie Utilizzate**

Il progetto è stato realizzato con un approccio full stack basato principalmente su tecnologie open source e linguaggi standard del web:

- Frontend
  - HTML5 e CSS3 per la struttura e lo stile delle pagine
  - Bootstrap 5.3 per il layout responsive e i componenti UI
  - JavaScript per la logica interattiva del client, senza l'uso di librerie esterne
- Backend
  - PHP 8.x come linguaggio server-side
  - PDO per la comunicazione sicura con il database
- Database
  - MySQL, gestito in locale tramite phpMyAdmin e pubblicato poi su InfinityFree
- Hosting
  - InfinityFree hosting service per la pubblicazione online del progetto
  - Dominio di produzione: <http://projectmuseum.infinityfreeapp.com/>
- Strumenti
  - VSCode come IDE principale di progettazione
  - XAMPP per setup Apache e SQL
  - Git per controllo di versione e documentazione workflow
  - Web DevTools per debug e testing

## ✓ Progettazione del Database



Il database è stato progettato per modellare una realtà museale le cui tabelle principali sono:

- Utenti (gestione interna di accessi alla dashboard, solo gli admin vedono tutti i dati)
- Clienti (utenti registrati per fruizione)
- Opere (archivio delle opere d'arte)
- Mostre (eventi museali di esposizione opere)
- Mostre-Opere (relazione N:N tra mostre e opere)

*N.B.: per garantire la sicurezza delle informazioni degli utenti, essi non presenti nel file di costruzione del DB .sql, per importarli è necessario visitare la pagina*

*“[http://projectmuseum.infinityfreeapp.com/utills/populate\\_users.php](http://projectmuseum.infinityfreeapp.com/utills/populate_users.php)” se si è online e*

*“/progettomuseo/utills/populate\_users.php” se si esegue il progetto in ambiente locale.*

## ✓ Funzionalità Implementate

Il sistema offre un set completo e organizzato di funzionalità, pensate per garantire un'esperienza utente fluida e una gestione efficace dei contenuti museali.





Di seguito una panoramica dettagliata:

### **Interfaccia e Navigazione**

- **Dashboard dinamica** con riepilogo dei dati principali (conteggio clienti, mostre, opere)
- **Sidebar contestuale persistente**, visibile in tutte le pagine, per accesso rapido alle funzionalità
- **Interfaccia responsive** realizzata con **Bootstrap 5**, ottimizzata per dispositivi mobili e tablet

## Gestione dei Dati (CRUD)

Sistema completo di gestione (Create, Read, Update, Delete) per le seguenti entità:

-  **Clienti**: anagrafica con nome, email e numero di telefono
-  **Opere**: titolo, autore, anno e descrizione dell'opera
-  **Mostre**: nome mostra, data inizio/fine e collegamento a opere esposte
-  **Utenti**: accesso riservato agli admin per gestione credenziali e ruoli

Tutte le operazioni CRUD sono accompagnate da:

- Form validati lato client
- Messaggi di conferma/successo
- Comandi per **modifica rapida**, **eliminazione sicura** (con conferma) e inserimento

## Ricerca, Filtri e Navigazione Dati

- **Campo di ricerca live** su nome/titolo per ciascuna entità
- **Filtro testuale** per affinare i risultati senza ricaricare la pagina
- **Paginazione automatica** per caricare e visualizzare grandi volumi di dati in modo efficiente (con offset e limit configurabili)

## Sicurezza e Autenticazione

- Sistema di **login con sessione PHP** per accesso protetto
- **Gestione ruoli** tramite campo ute\_ruolo (admin / user)
- Restrizione automatica dell'accesso alle sezioni di amministrazione e ai webservice riservati
- Salvataggio delle password **hashate in SHA-256**

## Web Services REST

- API RESTful per accesso remoto ai dati (descritti nel dettaglio nel capitolo dedicato)
- Interfacce separate per testare i webservice (wsCliente.php, wsMostra.php)
- Autenticazione API tramite APIKEY e header personalizzati

Tutte le funzionalità sono state testate in **ambiente locale** e **remoto**, ove possibile, garantendo **robustezza**, **usabilità** e **coerenza dei dati**.

## ✓ Web Services & API

Sono stati sviluppati diversi endpoint RESTful in /api/index.php, protetti da autenticazione tramite API-KEY. I metodi supportati sono GET e PUT.

API documentate:

- /mostre/attive
- /mostre
- /mostra/{id}
- /opere/random
- /opere/recenti
- /opere/autori
- /clienti
- /utenti (solo admin)

Due web service interattivi sono stati implementati nella sezione “Web Services” della dashboard:

- Tracking mostra: visualizza nome e opere di una mostra tramite GET /mostra/{id}
- Aggiorna cliente: modifica numero di telefono con PUT /clienti/{id}

Tutte le API sono descritte nella pagina apidoc.php, con struttura dettagliata, esempi, parametri e changelog.

## ✓ Pubblicazione Online

Il progetto è stato pubblicato online tramite il servizio di hosting gratuito **InfinityFree**, accessibile al seguente indirizzo:

 <http://projectmuseum.infinityfreeapp.com>

Per rendere funzionante l'intero progetto sul server remoto, è stato necessario intervenire in diversi punti del codice per **garantire compatibilità e operatività**, tenendo conto delle restrizioni imposte dalla piattaforma.

Di seguito vengono dettagliate le principali operazioni effettuate:

### 1. **Sostituzione della funzione getallheaders()**

InfinityFree non supporta la funzione getallheaders() su ambienti non Apache. È stata quindi implementata una versione alternativa per recuperare manualmente gli header della richiesta:

```

$headers = [];
foreach ($_SERVER as $name => $value) {
    if (str_starts_with($name, 'HTTP_')) {
        $key = str_replace('_', '-', ucwords(strtolower(str_replace('_', ' ', substr($name, 5))));
        $headers[$key] = $value;
    }
}

```

Questo garantisce la compatibilità su InfinityFree preservando la sicurezza e l'autenticazione tramite APIKEY

## 2. Parsing case-insensitive dell'APIKEY

Poiché il nome degli header può variare in maiuscolo/minuscolo o formato (Apikey, APIKEY, apikey), è stato implementato un controllo multiplo per leggere correttamente la chiave:

```
$receivedKey = $headers['Apikey'] ?? $headers['APIKEY'] ?? $headers['apikey'] ?? null;
```

## 3. Gestione dei percorsi nei file .js

Nei file JS come wsMostra.js, wsCliente.js e index.js, sono stati sostituiti i path locali hardcoded con percorsi dinamici basati su hostname:

```

const BASE_URL = location.hostname.includes('localhost')
? 'http://localhost/laboratorio/progettoMuseo/api/'
: 'http://projectmuseum.infinityfreeapp.com/api/';

```

In questo modo il progetto funziona correttamente sia in ambiente **locale** che su **server remoto** senza modificare ogni volta le URL.

## ⚠ Limitazioni di hosting: PUT & CORS

Durante la fase di testing online su hosting gratuito **InfinityFree**, è emersa una limitazione importante che **impedisce il corretto funzionamento delle chiamate HTTP PUT** (e, per estensione, anche DELETE, PATCH, ecc.).

In particolare, quando si tenta di eseguire un aggiornamento dei dati via fetch() con metodo PUT (es. aggiornare il numero di telefono di un cliente), il browser esegue una richiesta *preflight* OPTIONS come previsto dal protocollo **CORS**. InfinityFree **blocca** tali richieste e reindirizza verso una pagina di errore 403 (<https://errors.infinityfree.net/errors/403/>), impedendo l'effettivo invio della richiesta PUT al server.

La conseguenza diretta è un errore in console del tipo:

```
Access to fetch at 'http://projectmuseum.infinityfreeapp.com/api/clienti/1'
```

from origin 'http://projectmuseum.infinityfreeapp.com' has been blocked by CORS policy:

No 'Access-Control-Allow-Origin' header is present on the requested resource.

N.B.: Questo problema **non dipende dal codice JavaScript o PHP**, ma è causato da una **restrizione server-side** sul piano gratuito.

### ✓ **Soluzione consigliata**

Per testare il webservice `PUT /clienti/{id}` e verificare la corretta logica di aggiornamento, è **consigliato eseguire il progetto in locale**, in quanto tutte le funzionalità del progetto sono pienamente funzionanti in ambiente locale.

### **Note aggiuntive**

- L'autenticazione alle API è protetta da una APIKEY da includere nei headers di ogni richiesta.
- Le richieste GET alle API funzionano anche online (es. `/mostra/{id}`, `/opere/random`, ecc.)
- Le API sono documentate nella pagina `apidoc.php` con esempi, parametri e risposte attese.

### ✓ **Possibili Estensioni Future**

- Aggiunta di un sistema di prenotazione visite e biglietteria
- Upload immagini per opere e mostre (gallery dinamica)
- Sistema di commenti/recensioni per utenti
- Migrazione su server professionale con certificato HTTPS

### ✓ **Considerazioni Finali**

Il progetto rappresenta un esempio concreto di sviluppo full-stack moderno con particolare attenzione alla **modularità del codice**, **manutenibilità**, e **interoperabilità tra frontend, backend e API**.

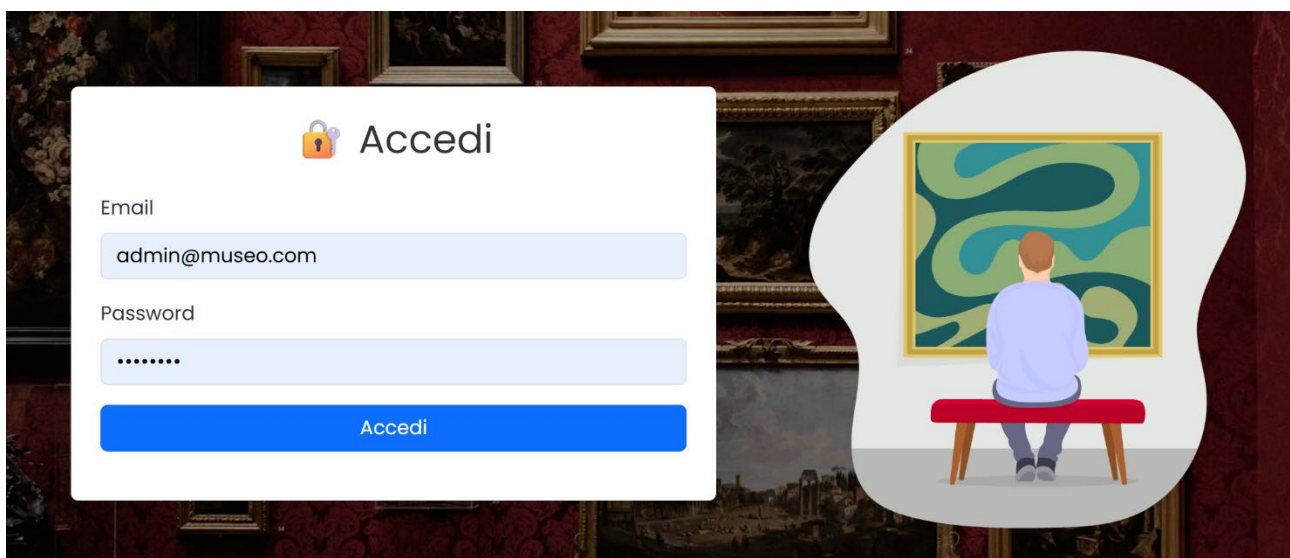
L'esperienza di pubblicazione ha inoltre fornito un'utile occasione per affrontare e superare problematiche reali di compatibilità tra ambienti di sviluppo e ambienti di produzione.

È stato sviluppato un sistema completo, funzionante, documentato e capace di adattarsi a scenari futuri di espansione funzionale.

✓ **Galleria**




*Landing page*



*Login*






Admin

Admin


- Dashboard
- Clienti
- Opere
- Mostre
- Utenti
- Test API
- Logout

## Benvenuto, Admin 🖐️




Clienti

4




Opere

10




### Web Services



#### Tracking Mostra

Visualizza i dettagli e le opere di una mostra inserendo il suo ID.


Dashboard



Admin

Admin

- Dashboard
- Clienti
- Opere
- Mostre
- Utenti
- Test API
- Logout



## Test API Museo

Seleziona API da testare:

Mostre attive


Invia Richiesta

Risultato:

```
{
  "oggi": "2025-05-04",
  "result": [
    {
      "mos_id": 1,
      "mos_nome": "Rinascimento Italiano",
      "mos_data_inizio": "2025-04-01",
      "mos_data_fine": "2025-06-15",
      "numero_opere": 2
    }
  ]
}
```

Test API





Admin

Admin

Dashboard

Clieni

Opere

Mostre

Utenti

Test API

Logout

Documentazione API

Autenticazione

Ogni richiesta deve includere l'APIKEY nell'header della richiesta.

Header	Valore
APIKEY	b4st0I5868HdCLAdetokrSPGdeT1Df9ixpeQpWgD
Content-Type	application/json

Mostre attive pubblica

Endpoint: /mostre/attive

Metodo: GET

Descrizione: restituisce l'elenco di tutte le mostre attualmente in corso, ovvero quelle la cui data di inizio è

Parametri:

Nome	Tipo	Obbligatorio	Descrizione
(nessuno)	-	-	Questa API non richiede parametri.

Risposta:

[

Documentazione API