Musicson - Piattaforma RESTful Stateless per la prenotazione online di lezioni musicali

Marco Barbera - 555979

Gennaio 2026

Indice

1 Introduzione al problema

Nella **programmazione web** e' ormai molto diffuso il paradigma **REST** per la realizzazione dei sistemi, la realizzazione di tali sistemi coinvolge la gestione di **connessioni stateless**, l'**autenticazione per-request** e l'**implementazione di un protocollo di comunicazione standardizzato** basato su HTTP/JSON.

Questa relazione descrive l'implementazione di una **Single Page Application(SPA)** per la prenotazione di lezioni musicali online per cui e' stato inventato il nome **Musicson**. Il sistema è stato realizzato seguendo il paradigma **RESTful** quanto piu' fedelmente possibile, implementando un'architettura **Stateless** in cui il server non mantiene alcuna informazione di stato tra le richieste.

Le principali problematiche affrontate includono:

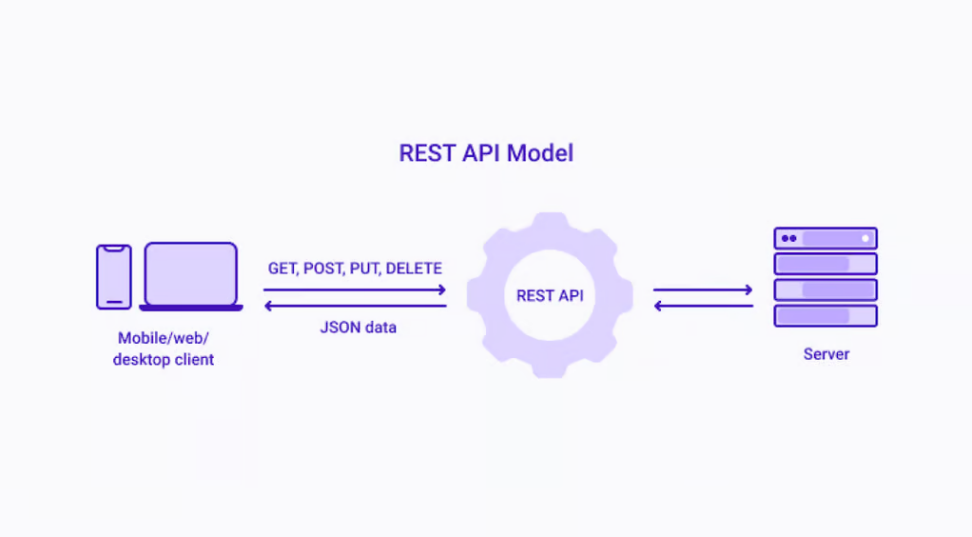
- **Architettura Stateless**: Necessità di gestire l'autenticazione e l'autorizzazione senza l'ausilio di sessioni lato server, in conformità ai principi REST.
- **Realizzazione di un backend solido**: Realizzare un backend che possa gestire efficacemente le richieste **HTTP** per gestire le risorse.
- **Dinamicita' della pagina**: Garantire un'interazione fluida tramite caricamento asincrono delle risorse (Fetch) e aggiornamento dei contenuti a schermo per ottimizzare l'esperienza utente in una Single Page Application (SPA) grazie al linguaggio di frontend **JavaScript**.

2 Stato dell'arte e architettura del sistema

Il sistema implementato segue lo stile architetturale **REST** (REpresentational State Transfer). L'architettura adottata è di tipo client-server, dove il frontend (Client) e il backend (Server) sono completamente disaccoppiati e comunicano esclusivamente tramite API interfaciate in formato JSON.

2.1 Struttura del programma

Il sistema presenta una struttura modulare suddivisa in tre livelli logici: Presentation Layer (UI.js), Application Layer (API PHP) e Data Layer (MySQL).



2.2 Struttura del codice

Il codice sorgente è organizzato secondo la seguente struttura gerarchica:

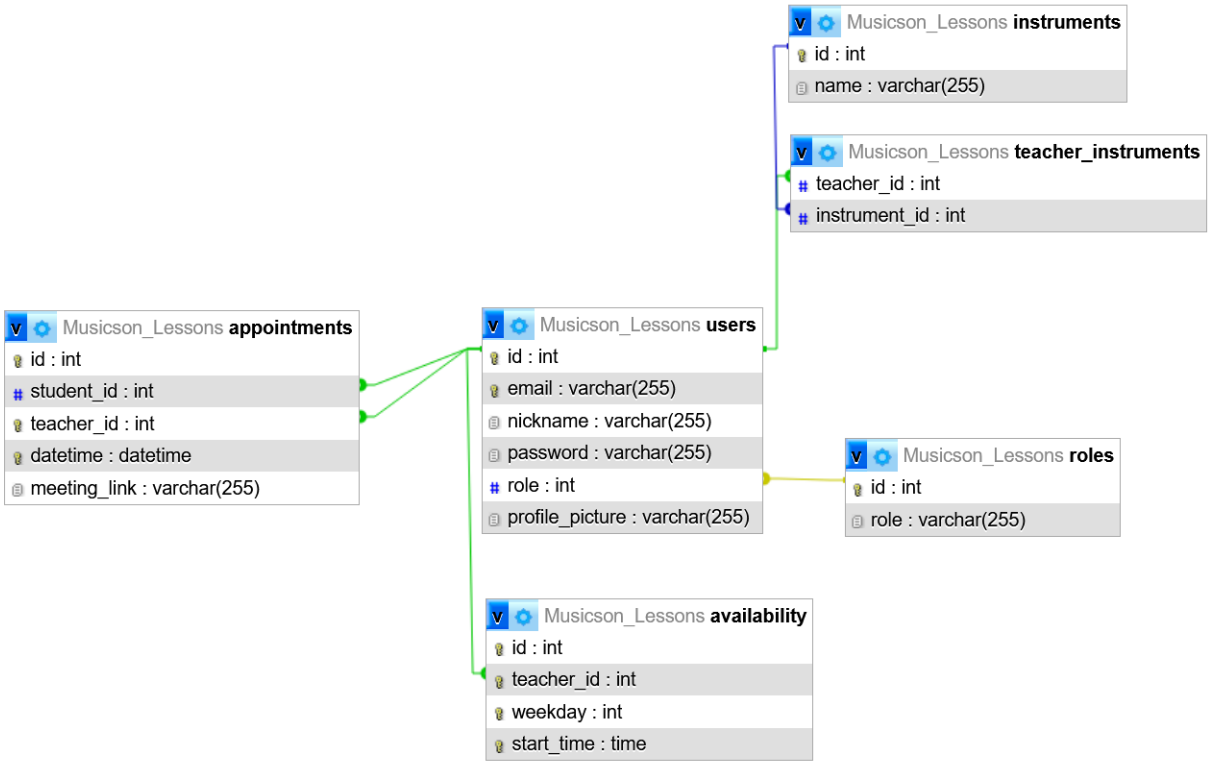
```
root/
├── index.html
├── docker-compose.yml
├── README.md
├── Musicson_Lessons.sql
├── Musicson_Lessons_data.sql
├── css/
│   └── style.css
├── js/
│   ├── main.js ...  (Entry point client)
│   ├── api.js ...  (Gestione networking e auth)
│   └── ui.js ...  (Manipolazione DOM)
└── php/
    ├── config/
    │   └── db.php
    └── api/
        ├── appointments.php
        ├── availability.php
        ├── instruments.php
        ├── login.php
        ├── profile.php
        └── teachers.php
```

3 Metodologia di implementazione

Il progetto è stato implementato utilizzando PHP per la logica server-side e JavaScript per la logica client-side. Per la persistenza dei dati è stato utilizzato un database relazionale MySQL.

3.1 Database

Il database contiene la struttura base per un'implementazione funzionale del sistema. Sono presenti le tabelle dedicate agli **strumenti**, **utenti** che possono avere un **ruolo** (1 = studente, 2 = professore), agli utenti possono essere assegnati degli **strumenti** e delle **disponibilita'** che pero' saranno accessibili solo dai professori tramite frontend, infine degli **appuntamenti** che vengono anche chiamati **Lezioni** all'interno della piattaforma in maniera analoga.



3.2 Tecnologie e Librerie Utilizzate

Sono state utilizzate librerie standard, senza dipendenze esterne (nessun framework) per mettere in pratica il paradigma REST.

3.2.1 Lato Backend (PHP)

Il backend si affida a funzioni native di PHP per la gestione del protocollo HTTP e la manipolazione dei dati JSON:

- **PDO (PHP Data Objects)**: Interfaccia di astrazione per l'accesso al database. Utilizzata per prevenire SQL Injection tramite prepared statements e garantire la portabilità del codice SQL.
- **header()**: Funzione essenziale per impostare gli header HTTP della risposta. Viene usata principalmente per definire il `Content-Type: application/json` e per inviare codici di stato HTTP specifici (es. 401, 500) in caso di errore prima che venga generato output.
- **http_response_code()**: Permette di impostare il codice di stato HTTP (es. `201 Created`, `404 Not Found`) in modo semantico, informando il client sull'esito dell'operazione.
- **json_encode()**: Converte gli array associativi PHP o gli oggetti in stringhe JSON, formato standard per la risposta delle API.
- **json_decode()**: Utilizzata per parsare il corpo delle richieste POST/PUT (ricevuto come raw JSON string) e convertirlo in array associativi PHP manipolabili.
- **file_get_contents('php://input')**: Permette di leggere il flusso di input raw della richiesta HTTP, necessario per recuperare i dati JSON inviati dal client (poiché `$_POST` non supporta nativamente il JSON).

3.2.2 Lato Frontend (JavaScript)

Il frontend sfrutta le moderne API del browser per gestire l'asincronia e lo stato dell'applicazione:

- **Fetch API**: Interfaccia moderna per eseguire richieste HTTP asincrone verso il server. Sostituisce il vecchio `XMLHttpRequest`, offrendo una sintassi più pulita basata sulle Promise.
- **Async / Await**: Per la gestione delle Promise, permette di scrivere codice asincrono (chiamate di rete) in stile procedurale, migliorando notevolmente la leggibilità rispetto alle catene di `.then()`.
- **sessionStorage**: Meccanismo di storage web che permette di salvare coppie chiave-valore nel browser. Nel progetto è utilizzato per persistere le credenziali dell'utente e il ruolo per la durata della sessione del browser, garantendo la natura stateless del server.

3.3 Connessione al Database

Il file **config/db.php** è responsabile della creazione della connessione con il database MySQL. Viene utilizzata l'estensione **PDO**(PHP Data Objects), che fornisce un'interfaccia consistente per l'accesso ai database e previene attacchi di SQL Injection tramite l'uso di prepared statements.

In ottica **RESTful**, la gestione degli errori è fondamentale: se la connessione fallisce, lo script non interrompe l'esecuzione con un errore fatale HTML, ma restituisce un codice di stato HTTP **500 Internal Server Error** e un payload JSON descrittivo.

```php
<?php
function getDbConnection() {
    $host = "mysql_proj_web";
    $db_name = "Musicson_Lessons";
    $username = "musicson_user";
    $password = "musicson_password";

    try {
        $conn = new PDO("mysql:host=$host;dbname=$db_name;charset=utf8", $username, $password);
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return $conn;
    } catch(PDOException $exception) {
        header('Content-Type: application/json');
        header('HTTP/1.1 500 Internal Server Error');
        echo json_encode(["error" => "Connessione al database fallita"]);
        exit;
    }
}
?>
```

3.4 Autenticazione Stateless

Il file **api/login.php** gestisce l'ingresso dell'utente nel sistema. Coerentemente con l'architettura Stateless, non viene avviata alcuna sessione PHP(`session_start()` non viene usata).

Lo script verifica le credenziali inviate tramite **Basic Auth**(decodificando gli header `PHP_AUTH_USER`) e confronta la password con l'hash della password memorizzato nel database tramite **password_verify()**. In caso di successo, restituisce un JSON con i dati utente che il client dovrà memorizzare per le richieste future.

```php
<?php
header('Content-Type: application/json');
require_once '../config/db.php';

// Verifica presenza header Basic Auth
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('HTTP/1.0 401 Unauthorized');
    echo json_encode(["error" => "Autenticazione richiesta"]);
    exit;
}

$user_email = $_SERVER['PHP_AUTH_USER'];
$user_pass  = $_SERVER['PHP_AUTH_PW'];
$db = getDbConnection();

try {
    $stmt = $db->prepare("SELECT * FROM users WHERE email = ?");
    $stmt->execute([$user_email]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    // Verifica hash della password
    if ($user && password_verify($user_pass, $user['password'])) {
        echo json_encode([
            "id" => $user['id'],
            "nickname" => $user['nickname'],
            "role" => $user['role'],
            "status" => "Authenticated"
        ]);
    } else {
        header('HTTP/1.0 401 Unauthorized');
        echo json_encode(["error" => "Credenziali non valide"]);
    }
} catch (PDOException $e) {
    header('HTTP/1.1 500 Internal Server Error');
    echo json_encode(["error" => "Errore interno"]);
}
```

3.5 Gestione delle Risorse e Metodi HTTP Backend

Il cuore dell'applicazione risiede negli endpoint dell' **API** che gestiscono le risorse (Strumenti, Appuntamenti, Disponibilità).

Ogni file PHP (come `instruments.php` o `appointments.php`) agisce come un controller per una specifica risorsa, implementando la logica per i diversi metodi HTTP supportati (GET, POST, DELETE).

Tutti questi endpoint condividono delle azioni comuni:

- Impostano l'header **Content-Type: application/json** per dire al client che il tipo di dato restituito e' JSON.
- Verificano l'**autenticazione** dell'utente prima di eseguire qualsiasi logica.
- Utilizzano uno if basato su **$_SERVER['REQUEST_METHOD']** per determinare l'azione da compiere.

3.5.1 Endpoint Strumenti (instruments.php)

Questo endpoint gestisce la risorsa *Strumenti*. Attualmente supporta solo il metodo **GET**, permettendo al client di scaricare la lista degli strumenti disponibili per popolare i menu a tendina o le barre di ricerca.

```php
$method = $_SERVER['REQUEST_METHOD'];   // prendere il metodo della richiesta
if ($method === 'GET') {
    try {
        // Query per prendere gli strumenti.
        $sql = "SELECT id, name FROM instruments ORDER BY name ASC";
        $stmt = $db->query($sql);
        $instruments = $stmt->fetchAll(PDO::FETCH_ASSOC);
        // codificare gli strumenti in json e mandarli al client
        echo json_encode($instruments);
    } catch (PDOException $e) {
        header('HTTP/1.1 500 Internal Server Error');
        echo json_encode(["error" => "Errore strumenti: " . $e->getMessage()]);
    }
} else {
    header('HTTP/1.1 405 Method Not Allowed');
    echo json_encode(["error" => "Metodo non supportato"]);
}
```

3.5.2 Endpoint Appuntamenti (appointments.php)

Questo è uno degli endpoint più complessi, in quanto gestisce gli appuntamenti/lezioni. A seconda del metodo HTTP ricevuto, il file esegue operazioni diverse, garantendo sempre che l'utente abbia i permessi necessari (ad esempio, un utente può cancellare solo le proprie lezioni).

- **GET**: Recupera la lista delle lezioni. La query SQL si adatta dinamicamente in base al ruolo dell'utente (se è uno studente, recupera i dati del professore; se è un professore, recupera i dati dello studente).
- **POST**: Crea una nuova prenotazione. Riceve `teacher_id` e `datetime`, genera un link univoco per il meeting e inserisce il record nel database. Gestisce il codice **409 Conflict** se lo slot è già occupato.
- **DELETE**: Cancella una prenotazione esistente, verificando prima che l'ID appartenga all'utente richiedente.

```php
$method = $_SERVER['REQUEST_METHOD'];   // prendere il metodo della richiesta
if ($method === 'GET') {
    // Logica differenziata per Studente/Professore
    if ($role == 2) { $sql = "SELECT ... FROM appointments WHERE teacher_id = :my_id ...";}
    else {$sql = "SELECT ... FROM appointments WHERE student_id = :my_id ...";}

    $stmt = $db->prepare($sql);
    $stmt->bindParam(':my_id', $userId);
    $stmt->execute();
    $appointments = $stmt->fetchAll(PDO::FETCH_ASSOC);

    // codificare gli strumenti in json e mandarli al client
    echo json_encode($appointments);
} elseif ($method === 'POST') {
    // Lettura dati in arrivo
    $input = json_decode(file_get_contents('php://input'), true);

    try {
        $sql = "INSERT INTO appointments ...";
        http_response_code(201); // Created

        echo json_encode(["message" => "Prenotazione confermata!", "link" => $meetingLink]);
    } catch (PDOException $e) {
        // Gestione duplicati
        if ($e->getCode() == 23000) header('HTTP/1.1 409 Conflict');
    }

} elseif ($method === 'DELETE') {
    // Leggiamo il body della richiesta
    $input = json_decode(file_get_contents('php://input'), true);

    $appointmentId = $input['id'];

    // Verifichiamo che questa prenotazione appartenga davvero all'utente loggato
    $checkSql = "SELECT id FROM appointments WHERE id = ? AND (student_id = ? OR teacher_id = ?)";
    $stmtCheck = $db->prepare($checkSql);
    $stmtCheck->execute([$appointmentId, $userId, $userId]);

    if (!$stmtCheck->fetch()) {
        header('HTTP/1.1 403 Forbidden');
         echo json_encode(["error" => "Non hai i permessi per cancellare questa lezione."]);
        exit;
    }

    // CANCELLAZIONE
    $sql = "DELETE FROM appointments WHERE id = :id";
    $stmt = $db->prepare($sql);
    $stmt->execute([':id' => $appointmentId]);
    echo json_encode(["message" => "Lezione cancellata con successo."]);
}else {
    header('HTTP/1.1 405 Method Not Allowed');
    echo json_encode(["error" => "Metodo non supportato"]);
}
```

3.6 Interazione Client-Server (api.js)

Il file **js/api.js** costituisce il livello di astrazione per la comunicazione di rete. Questo modulo centralizza tutte le chiamate HTTP, garantendo che il frontend rimanga disaccoppiato dalla logica di basso livello e mantenendo il codice pulito e riutilizzabile.

3.6.1 Wrapper di Autenticazione (fetchAuth)

La funzione wrapper **fetchAuth** rappresenta il componente chiave per supportare l'architettura **Stateless** lato client. Poiché il server non mantiene alcuna sessione, il client ha la responsabilità di inviare le credenziali ad ogni singola richiesta. Questa funzione agisce come un *wrapper* sulla funzione nativa `fetch`: recupera le credenziali codificate in Base64 dal `sessionStorage` e le inietta automaticamente nell'header `Authorization`.

```
async function fetchAuth(url, options = {}) {
    const auth = sessionStorage.getItem('user_auth');
    if (auth) {
        // Spread operator (...) per mantenere header esistenti
        // e aggiungere l'Authorization Basic
        options.headers = { ...options.headers, 'Authorization': `Basic ${auth}` };
    }
    return fetch(url, options);
}
```

La funzione **logout()** si occupa di rimuovere dal browser le credenziali dal session storage e il ruolo dell'utente(salvato per comodita' ai fini di alcune query).

```
function logout() {
    sessionStorage.removeItem('user_auth');      // pulizia credenziali
    sessionStorage
    sessionStorage.removeItem('user_role');      // pulizia ruolo sessionStorage
    location.reload();                           // ricarica pagina per resettare lo
    stato dell'applicazione
}
```

3.6.2 Metodi CRUD Helper

Per facilitare l'interazione con gli endpoint PHP descritti in precedenza, sono state implementate funzioni helper che rispecchiano i metodi **HTTP**:

- **getData**: Gestisce le richieste `GET`. Si occupa di trasformare un oggetto JavaScript in una Query String (es. `?strumento=Piano`) e gestisce centralmente l'errore **401 Unauthorized**, forzando il logout se le credenziali non sono più valide.
- **postData**: Gestisce le richieste `POST` per la creazione di nuove risorse. Imposta automaticamente l'header `Content-Type: application/json` e serializza il payload dati in formato JSON prima dell'invio.
- **deleteData**: Gestisce le richieste `DELETE` per la rimozione di risorse. Utilizza la stessa logica di serializzazione della POST per inviare i dati necessari all'eliminazione (es. l'ID della risorsa) nel corpo della richiesta.

```
async function getData(resource, params = null) {
    let url = `php/api/${resource}.php`;

    // Costruzione dinamica della Query String
    if (params) {
        const queryString = new URLSearchParams(params).toString();
        url += `?${queryString}`;
    }

    const response = await fetchAuth(url);

    // Gestione centralizzata errore 401 (Logout forzato)
    if (response.status === 401) {
        sessionStorage.removeItem('user_auth');
        location.reload();
        return null;
    }

    return await response.json();
}
```

```
async function postData(resource, data) {
    const response = await fetchAuth(`php/api/${resource}.php`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
    });
    return await response.json();
}
```

```
async function deleteData(resource, data) {
    const response = await fetchAuth(`php/api/${resource}.php`, {
        method: 'DELETE',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
    });
    return await response.json();
}
```

3.7 Logica di Presentazione (ui.js)

Il file **js/ui.js** gestisce la manipolazione del DOM e il rendering dinamico dell'HTML. A differenza di un sito tradizionale dove l'HTML è generato dal server, in questa SPA il server invia solo dati (JSON) e il client costruisce l'interfaccia "al volo".

3.7.1 Gestione Stato Interfaccia (updateUI)

La funzione `updateUI` è il cuore della logica di presentazione. Viene richiamata ogni volta che lo stato dell'autenticazione cambia (login, logout o refresh della pagina). Il suo compito è duplice:

1. Alternare la visibilità tra il form di Login e il contenuto principale.
2. Iniettare dinamicamente i controlli utente (Nome, Bottoni di navigazione) nell'header, personalizzando l'esperienza in base ai dati dell'utente loggato.

```
/**
 * Gestisce lo stato globale dell'interfaccia (Loggato vs Non Loggato).
 * Mostra/Nasconde i container principali e aggiorna l'header.
 */
function updateUI(isAuthenticated, userData = null) {
    const loginBox = document.getElementById('login-container');
    const contentBox = document.getElementById('content-container');
    const statusBox = document.getElementById('user-status');

    if (isAuthenticated) {
        // UTENTE LOGGATO: Nasconde login, mostra contenuto
        loginBox.style.display = 'none';
        contentBox.style.display = 'block';

        // Iniezione dinamica dei controlli utente nell'header
        statusBox.innerHTML = `
            <div class="header-controls">
                <span>Ciao <strong>${userData.nickname}</strong></span>
                <button onclick="goHome()">Home</button>
                <button onclick="showMyAppointments()">Mie Lezioni</button>
                <button onclick="openProfileModal()">Profilo</button>
                <button onclick="logout()" class="btn-logout">Esci</button>
            </div>
        `;

        loadInstruments(); // Carica dati necessari
        goHome();
    } else {
        // UTENTE NON LOGGATO: Mostra solo il login
        loginBox.style.display = 'block';
        contentBox.style.display = 'none';
        statusBox.innerHTML = "";
    }
}
```

3.7.2 Altre funzionalità di UI

Il resto del file si occupa di generare componenti specifici in risposta alle azioni dell'utente:

- **Rendering Card**: Le funzioni come `createTeacherCard` generano stringhe HTML per visualizzare i risultati della ricerca.
- **Gestione Modali**: Funzioni per aprire e chiudere le finestre di prenotazione e profilo (es. `openBookingModal`).

- **Gestione Slot**: Logica visiva per l'aggiunta e la rimozione degli orari di disponibilità nel profilo del docente.

3.8 Entry Point e Inizializzazione (main.js)

Il file **js/main.js** funge da punto di ingresso dell'applicazione. Il suo compito è collegare la logica di business (API) con l'interfaccia utente (UI) e gestire il ciclo di vita dell'applicazione al caricamento della pagina.

Quando l'utente effettua il primo accesso, questo script intercetta l'evento di click sul bottone di login. Qui avvengono i seguenti eventi:

1. Le credenziali (email e password) vengono lette dal form.
2. Vengono codificate in formato Base64 tramite la funzione nativa `btoa()`, creando la stringa standard per l'**HTTP Basic Auth**.
3. Se il server risponde positivamente, questa stringa viene salvata nel `sessionStorage`. Da questo momento in poi, sarà il "passpartout" per tutte le chiamate future.

```
document.getElementById('login-button').addEventListener('click', async () => {
    const email = document.getElementById('login-email').value;
    const pass = document.getElementById('login-password').value;

    // Codifica credenziali in Base64 per Basic Auth standard
    const credentials = btoa(`${email}:${pass}`);

    // Chiamata diretta (senza wrapper) per il primo accesso
    const response = await fetch('php/api/login.php', {
        headers: { 'Authorization': `Basic ${credentials}` }
    });

    if (response.ok) {
        const data = await response.json();
        // Salvataggio token nel browser (Persistenza Client-Side)
        sessionStorage.setItem('user_auth', credentials);
        sessionStorage.setItem('user_role', data.role);
        updateUI(true, data);
    } else {
        document.getElementById('login-error').innerText = "Credenziali errate.";
    }
});
```

3.8.1 Ripristino dello Stato (onLoad)

Poiché il server è stateless, non ricorda se l'utente ha visitato la pagina 5 secondi fa. Per evitare che l'utente debba fare login ad ogni refresh (F5), all'evento `window.onload` il sistema controlla se esistono credenziali salvate nel browser.

Se le credenziali sono presenti nel sessionStorage, viene effettuata una chiamata di verifica (`fetchAuth`) verso il server. Questo garantisce sicurezza: se la password è cambiata nel database nel frattempo, la verifica fallirà e l'utente verrà riportato alla schermata di login.

```
window.onload = async () => {
    let isAuthenticated = false;
    let userData = null;

    // Controllo persistenza locale
    if (sessionStorage.getItem('user_auth')) {
        // Verifica validit  credenziali col server
        const response = await fetchAuth('php/api/login.php');

        if (response.ok) {
            isAuthenticated = true;
            userData = await response.json();
        }
    }

    // Aggiornamento UI in base all'esito
    updateUI(isAuthenticated, userData);
};
```

4 Risultati sperimentali

E' stata testata l'API RESTful indipendentemente dall'interfaccia utente tramite dei test diretti sugli endpoint API utilizzando lo strumento da riga di comando **curl**.

Questi test confermano che l'architettura Stateless funziona correttamente, richiedendo l'autenticazione ad ogni chiamata e gestendo correttamente i metodi HTTP e i payload JSON.

4.1 Autenticazione e Accesso

Verifica delle credenziali tramite Basic Auth. Il server risponde con i dati dell'utente e lo stato di autenticazione, confermando che il meccanismo di login (senza sessione server-side) è funzionante.

```
curl.exe -u "marco@gmail.com:marco123" http://localhost:8080/php/api/login.php
{"id":1,"nickname":"Marco","role":1,"status":"Authenticated"}
```

4.2 Recupero Risorse Pubbliche (richiedono comunque l'autenticazione

Richiesta della lista completa degli strumenti disponibili per il popolamento delle interfacce di ricerca.

```
curl.exe -u "marco@gmail.com:marco123" http://localhost:8080/php/api/instruments.php
[{"id":3,"name":"Basso"},{"id":4,"name":"Batteria"},{"id":1,"name":"Chitarra"},
{"id":5,"name":"Flauto"},{"id":2,"name":"Pianoforte"}]
```

Recupero della lista completa dei professori registrati nella piattaforma.

```
curl.exe -u "marco@gmail.com:marco123" http://localhost:8080/php/api/teachers.php
[{"id":2,"nickname":"Bono","profile_picture":"default.png",
"instruments":"Chitarra, Pianoforte"},{"id":3,"nickname":"PaulMcCartney",
"profile_picture":"default.png","instruments":"Basso, Pianoforte, Chitarra"},
{"id":6,"nickname":"JhonLennon","profile_picture":"default.png",
"instruments":"Chitarra"},{"id":7,"nickname":"RingoStar",
"profile_picture":"default.png","instruments":"Batteria"},
{"id":8,"nickname":"JimmyPage","profile_picture":"default.png",
"instruments":"Chitarra"},{"id":9,"nickname":"JackWhite",
"profile_picture":"default.png","instruments":"Chitarra"}]
```

4.3 Filtraggio Risorse (Query String)

Test della funzionalità di filtro tramite parametri GET. In questo caso, vengono richiesti solo i docenti abilitati all'insegnamento del "Pianoforte".

```
curl.exe -u "marco@gmail.com:marco123"
http://localhost:8080/php/api/teachers.php?strumento=Pianoforte

[{"id":2,"nickname":"Bono","profile_picture":"default.png",
"instruments":"Chitarra, Pianoforte"},{"id":3,"nickname":"PaulMcCartney",
"profile_picture":"default.png","instruments":"Basso, Pianoforte, Chitarra"}]
```

4.4 Gestione Appuntamenti (CRUD)

Verifica del ciclo di vita completo di una prenotazione: visualizzazione, creazione (POST) e cancellazione (DELETE).

1. Visualizzazione appuntamenti esistenti:

```
curl.exe -u "marco@gmail.com:marco123" http://localhost:8080/php/api/appointments.php
[{"id":21,"datetime":"2026-01-26 17:00:00",
"meeting_link":"https:\/\/meet.google.com\/9e856fcee5","partner_name":"Bono",
"partner_image":"default.png"},{"id":24,"datetime":"2026-01-27 15:00:00",
"meeting_link":"https:\/\/meet.google.com\/be81e8963f",
"partner_name":"PaulMcCartney","partner_image":"default.png"},
{"id":23,"datetime":"2026-01-28 14:00:00",
"meeting_link":"https:\/\/meet.google.com\/d6d9b0b29f",
"partner_name":"PaulMcCartney","partner_image":"default.png"}]
```

2. Creazione di un nuovo appuntamento (POST): Si noti l'utilizzo della sintassi `-%` per la corretta gestione del JSON in ambiente PowerShell.

```
curl.exe --% -X POST -u "marco@gmail.com:marco123"
-H "Content-Type: application/json"
-d "{ \"teacher_id\": 2, \"datetime\": \"2026-02-10 15:00:00\" }"
http://localhost:8080/php/api/appointments.php

{"message":"Prenotazione confermata!",
"link":"https:\/\/meet.google.com\/6d2599dcf8"}
```

3. Cancellazione dell'appuntamento (DELETE): Rimozione della risorsa appena creata tramite il suo ID.

```
curl.exe --% -X DELETE -u "marco@gmail.com:marco123"
-H "Content-Type: application/json" -d "{ \"id\": 22 }"
http://localhost:8080/php/api/appointments.php

{"message":"Lezione cancellata con successo."}
```

5 Conclusioni e sviluppi futuri

Il progetto ha raggiunto l'obiettivo di implementare una Single Page Application funzionale basata sul paradigma **RESTful**, utilizzando tecnologie native senza l'ausilio di framework esterni.

Sebbene il sistema costituisca una solida base, esistono margini di miglioramento per rendere l'applicazione pronta per un ambiente di produzione:

- **Sicurezza delle Credenziali**: Attualmente, per supportare l'architettura stateless, le credenziali vengono salvate nel `sessionStorage` con una semplice codifica Base64 (che è facilmente decodificabile). Per aumentare la sicurezza, sarebbe opportuno evitare di salvare le password lato client, implementando un sistema di autenticazione basato su **Token** (come JWT - JSON Web Token) con scadenza temporale e refresh automatico.
- **Espansione delle Funzionalità**: La piattaforma potrebbe essere arricchita con numerose feature aggiuntive, un sistema di messaggistica e un pannello di amministrazione avanzato.
- **Gestione Errori**: Migliorare la gestione degli errori lato frontend per fornire feedback visivi più precisi all'utente in caso di problemi di rete o validazione.