

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

Marco Barbera - 555979

Gennaio 2026

Indice

1	Introduzione al problema	3
2	Stato dell'arte e architettura del sistema	4
2.1	Struttura del programma	4
2.2	Struttura del codice	4
3	Metodologia di implementazione	5
3.1	Database	5
3.2	Tecnologie e Librerie Utilizzate	5
3.2.1	Lato Backend (PHP)	5
3.2.2	Lato Frontend (JavaScript)	6
4	Backend	7
4.1	Connessione al Database	7
4.2	Autenticazione Stateless	8
4.3	Gestione delle Risorse e Metodi HTTP Backend	9
4.3.1	Endpoint Strumenti (instruments.php)	9
4.3.2	Endpoint Appuntamenti (appointments.php)	9
5	Frontend	11
5.1	Interazione Client-Server (api.js)	11
5.1.1	Metodi CRUD	12
5.2	Logica di Presentazione (ui.js)	14
5.2.1	Gestione Stato Interfaccia (updateUI)	14
5.2.2	Altre funzionalità di UI	14
5.3	Entry Point e Inizializzazione (main.js)	15
5.3.1	Ripristino dello Stato (onLoad)	16
6	Risultati sperimentali	17
6.1	Autenticazione e Accesso	17
6.2	Recupero Risorse Pubbliche (richiedono comunque l'autenticazione)	17
6.3	Filtraggio Risorse (Query String)	17
6.4	Gestione Appuntamenti (CRUD)	18
7	Web Pages	19
7.1	Schermata di Login	19
7.2	Schermata di Registrazione	19
7.3	Home Page e Ricerca	20

7.4	Risultati della Ricerca	20
7.5	Modale di Prenotazione	21
7.6	Profilo Utente e Gestione Orari	21
7.7	Mie Lezioni	22
8	Conclusioni e sviluppi futuri	23

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:

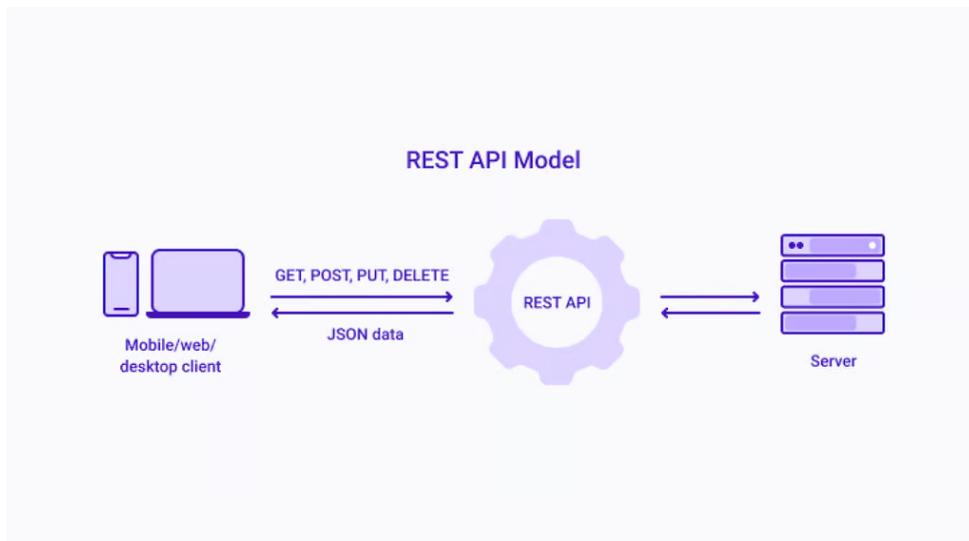
- **Architettura Stateless:** Necessità di gestire l'autenticazione e l'autorizzazione senza l'ausilio di sessioni lato server, in conformità ai principi REST.
- **Realizzazione di un backend solido:** Realizzare un backend che possa gestire efficacemente le richieste **HTTP** per gestire le risorse.
- **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 architettonico **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:

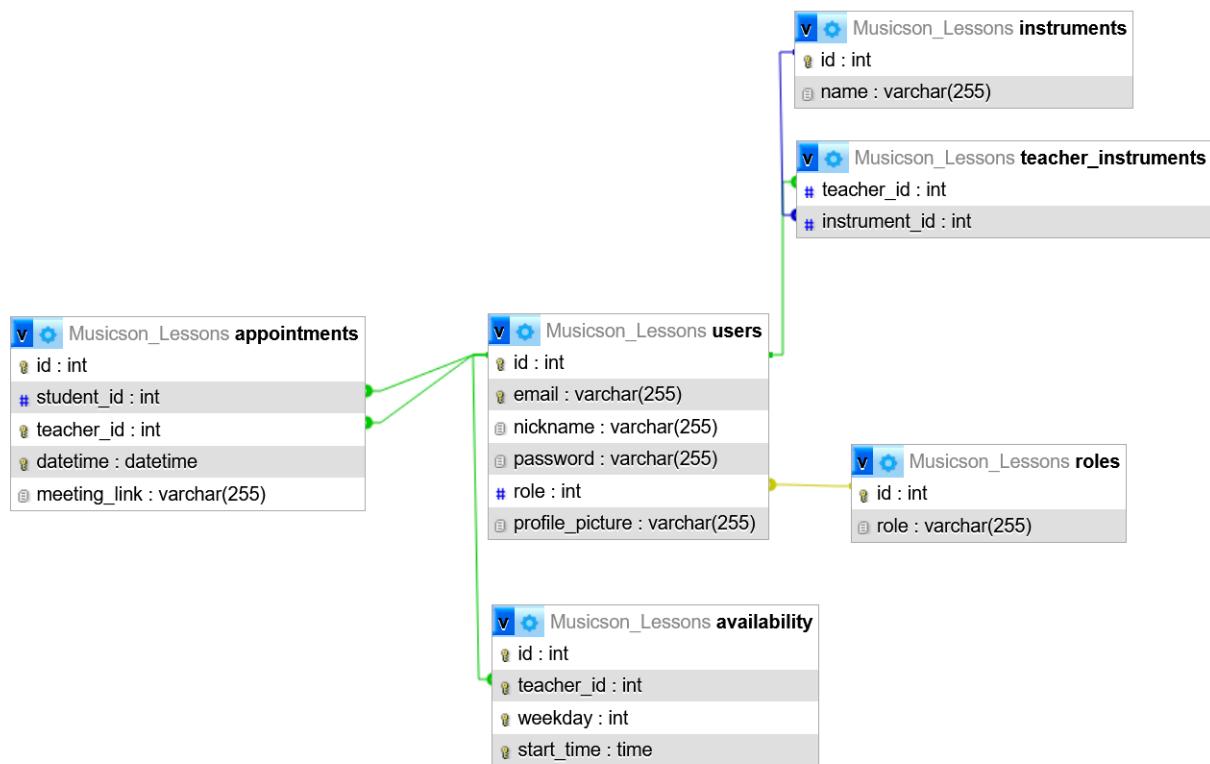
```
root/
  index.html
  docker-compose.yml
  README.md
  Musicson_Lessons.sql
  Musicson_Lessons_data.sql
  css/
    style.css
  js/
    main.js ... (Entry point client)
    api.js ... (Gestione networking e auth)
    ui.js ... (Manipolazione DOM)
  php/
    config/
      db.php
    api/
      appointments.php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

4 Backend

In questa sezione vengono analizzati i dettagli implementativi della logica server-side.

4.1 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.

```
1 <?php
2 function getDbConnection() {
3     $host = "mysql_proj_web";
4     $db_name = "Musicson_Lessons";
5     $username = "musicson_user";
6     $password = "musicson_password";
7
8     try {
9         $conn = new PDO("mysql:host=$host;dbname=$db_name;charset=utf8", $username,
10                     $password);
11         $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
12         return $conn;
13     } catch(PDOException $exception) {
14         header('Content-Type: application/json');
15         header('HTTP/1.1 500 Internal Server Error');
16         echo json_encode(["error" => "Connessione al database fallita"]);
17         exit;
18     }
19 ?>
```

4.2 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.

```
1 <?php
2 header('Content-Type: application/json');
3 require_once '../config/db.php';
4
5 // Verifica presenza header Basic Auth
6 if (!isset($_SERVER['PHP_AUTH_USER'])) {
7     header('HTTP/1.0 401 Unauthorized');
8     echo json_encode(["error" => "Autenticazione richiesta"]);
9     exit;
10}
11
12$user_email = $_SERVER['PHP_AUTH_USER'];
13$user_pass = $_SERVER['PHP_AUTH_PW'];
14$db = getDbConnection();
15
16try {
17    $stmt = $db->prepare("SELECT * FROM users WHERE email = ?");
18    $stmt->execute([$user_email]);
19    $user = $stmt->fetch(PDO::FETCH_ASSOC);
20
21    // Verifica hash della password
22    if ($user && password_verify($user_pass, $user['password'])) {
23        echo json_encode([
24            "id" => $user['id'],
25            "nickname" => $user['nickname'],
26            "role" => $user['role'],
27            "status" => "Authenticated"
28        ]);
29    } else {
30        header('HTTP/1.0 401 Unauthorized');
31        echo json_encode(["error" => "Credenziali non valide"]);
32    }
33} catch (PDOException $e) {
34    header('HTTP/1.1 500 Internal Server Error');
35    echo json_encode(["error" => "Errore interno"]);
36}
```

4.3 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 è 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.

4.3.1 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.

```
1 $method = $_SERVER['REQUEST_METHOD']; // prendere il metodo della richiesta
2 if ($method === 'GET') {
3     try {
4         // Query per prendere gli strumenti.
5         $sql = "SELECT id, name FROM instruments ORDER BY name ASC";
6         $stmt = $db->query($sql);
7         $instruments = $stmt->fetchAll(PDO::FETCH_ASSOC);
8         // codificare gli strumenti in json e mandarli al client
9         echo json_encode($instruments);
10    } catch (PDOException $e) {
11        header('HTTP/1.1 500 Internal Server Error');
12        echo json_encode(["error" => "Errore strumenti: " . $e->getMessage()]);
13    }
14 } else {
15     header('HTTP/1.1 405 Method Not Allowed');
16     echo json_encode(["error" => "Metodo non supportato"]);
17 }
```

4.3.2 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.

```

1 $method = $_SERVER['REQUEST_METHOD']; // prendere il metodo della richiesta
2 if ($method === 'GET') {
3     // Logica differenziata per Studente/Professore
4     if ($role == 2) { $sql = "SELECT ... FROM appointments WHERE teacher_id = :my_id ...";}
5     else {$sql = "SELECT ... FROM appointments WHERE student_id = :my_id ...";}
6
7     $stmt = $db->prepare($sql);
8     $stmt->bindParam(':my_id', $userId);
9     $stmt->execute();
10    $appointments = $stmt->fetchAll(PDO::FETCH_ASSOC);
11
12    // codificare gli strumenti in json e mandarli al client
13    echo json_encode($appointments);
14 } elseif ($method === 'POST') {
15     // Lettura dati in arrivo
16     $input = json_decode(file_get_contents('php://input'), true);
17
18     try {
19         $sql = "INSERT INTO appointments ...";
20         http_response_code(201); // Created
21
22         echo json_encode(["message" => "Prenotazione confermata!", "link" => $meetingLink]);
23     } catch (PDOException $e) {
24         // Gestione duplicati
25         if ($e->getCode() == 23000) header('HTTP/1.1 409 Conflict');
26     }
27
28 } elseif ($method === 'DELETE') {
29     // Leggiamo il body della richiesta
30     $input = json_decode(file_get_contents('php://input'), true);
31
32     $appointmentId = $input['id'];
33
34     // Verifichiamo che questa prenotazione appartenga davvero all'utente loggato
35     $checkSql = "SELECT id FROM appointments WHERE id = ? AND (student_id = ? OR teacher_id = ?)";
36     $stmtCheck = $db->prepare($checkSql);
37     $stmtCheck->execute([$appointmentId, $userId, $userId]);
38
39     if (!$stmtCheck->fetch()) {
40         header('HTTP/1.1 403 Forbidden');
41         echo json_encode(["error" => "Non hai i permessi per cancellare questa lezione."]);
42         exit;
43     }
44
45     // CANCELLAZIONE
46     $sql = "DELETE FROM appointments WHERE id = :id";
47     $stmt = $db->prepare($sql);
48     $stmt->execute([':id' => $appointmentId]);
49     echo json_encode(["message" => "Lezione cancellata con successo."]);
50 } else {
51     header('HTTP/1.1 405 Method Not Allowed');
52     echo json_encode(["error" => "Metodo non supportato"]);
53 }
```

5 Frontend

In questa sezione viene analizzata la logica client-side gestita tramite JavaScript.

5.1 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.

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`.

```
1 /**
2  * Wrapper per la funzione nativa fetch.
3  * Inietta automaticamente l'header 'Authorization' se sono presenti credenziali in
4  * sessione.
5  * @param {string} url - L'endpoint da chiamare.
6  * @param {object} options - Opzioni standard della fetch (method, headers, body).
7  * @returns {Promise<Response>} - Oggetto Response nativo.
8 */
9 async function fetchAuth(url, options = {}) {
10   const auth = sessionStorage.getItem('user_auth');
11   if (auth) {
12     // Spread operator (...) per mantenere header esistenti e aggiungere
13     // Authorization
14     options.headers = { ...options.headers, 'Authorization': `Basic ${auth}` };
15   }
16   return fetch(url, options);
17 }
```

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

```
1 // Procedura di Logout
2 function logout() {
3   sessionStorage.removeItem('user_auth');           // pulizia credenziali
4   sessionStorage.removeItem('user_role');          // pulizia ruolo sessionStorage
5   location.reload();                            // ricarica pagina per resettare lo
6 }
```

5.1.1 Metodi CRUD

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.

```
1 /**
2  * Esegue una richiesta GET autenticata.
3  * Gestisce la costruzione della Query String e il controllo della scadenza
4  * sessione (401).
5  * * @param {string} resource - Il nome della risorsa (es. 'teachers', 'appointments').
6  * @param {object|null} params - Oggetto chiave-valore per i parametri GET (es. {strumento: 'Piano'}).
7  * @returns {Promise<object|null>} - Il JSON di risposta o null in caso di errore
8  * auth.
9  */
10 async function getData(resource, params = null) {
11   let url = 'php/api/${resource}.php';
12
13   // Conversione parametri oggetto -> Query String (es. ?strumento=Piano&min=1)
14   if (params) {
15     const queryString = new URLSearchParams(params).toString();
16     url += `?${queryString}`;
17   }
18
19   const response = await fetchAuth(url);
20
21   // Gestione errore 401 (Unauthorized): Sessione scaduta o invalida
22   if (response.status === 401) {
23     sessionStorage.removeItem('user_auth');
24     location.reload(); // Ricarica forzata per tornare al login
25     return null;
26   }
27
28   return await response.json();
29 }
```

```
1 /**
2  * Esegue una richiesta POST autenticata per creare nuove risorse.
3  * Invia i dati in formato JSON.
4  * * @param {string} resource - Il nome della risorsa.
5  * @param {object} data - Il payload dei dati da inviare.
6  * @returns {Promise<object>} - Il JSON di risposta del server.
7  */
8 async function postData(resource, data) {
9   const response = await fetchAuth('php/api/${resource}.php', {
10     method: 'POST',
11     headers: { 'Content-Type': 'application/json' },
12     body: JSON.stringify(data)
13   }
```

```
13     });
14     return await response.json();
15 }

1 /**
2  * Esegue una richiesta DELETE autenticata per rimuovere risorse.
3  * @param {string} resource - Il nome della risorsa.
4  * @param {object} data - Il payload contenente l'ID da eliminare.
5  * @returns {Promise<object>} - Il JSON di risposta.
6 */
7 async function deleteData(resource, data) {
8     const response = await fetchAuth('php/api/${resource}.php', {
9         method: 'DELETE',
10        headers: { 'Content-Type': 'application/json' },
11        body: JSON.stringify(data)
12    });
13    return await response.json();
14 }
```

5.2 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".

5.2.1 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.

```
1 /**
2  * Aggiorna l'interfaccia in base allo stato di autenticazione.
3  * Mostra la dashboard se l'utente loggato, altrimenti mostra il login.
4 */
5 function updateUI(isAuthenticated, userData = null) {
6     const loginBox = document.getElementById('login-container');
7     const contentBox = document.getElementById('content-container');
8     const statusBox = document.getElementById('user-status');
9
10    if (isAuthenticated) {
11        // Utente Loggato
12        loginBox.style.display = 'none'; // nasconde il box di login
13        contentBox.style.display = 'block'; // mostra il container del contenuto
14
15        // Inserisce i bottoni di controllo nell'header
16        statusBox.innerHTML =
17            <div class="header-controls">
18                <span>Ciao <strong>${userData.nickname}</strong></span>
19                <button onclick="goHome()">Home</button>
20                <button onclick="showMyAppointments()">Mie Lezioni</button>
21                <button onclick="openProfileModal()">Profilo</button>
22                <button onclick="logout()" class="btn-logout">Esci</button>
23            </div>
24    ;
25
26    loadInstruments(); // caricare gli strumenti nella datalist
27    goHome(); // mostra gli elementi della schermata principale
28 } else {
29     // Utente Non Loggato
30     loginBox.style.display = 'block'; // mostrare il box login
31     contentBox.style.display = 'none'; // nascondere contenuti
32     statusBox.innerHTML = ""; // nascondere status utente
33 }
34 }
```

5.2.2 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.

5.3 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.

```

1 // Gestione click sul tasto Login (se login avviene con successo la ui viene
2 // aggiornata da utente autenticato altrimenti viene scritto un errore)
3 document.getElementById('login-button').addEventListener('click', async () => {
4     const email = document.getElementById('login-email').value;
5     const pass = document.getElementById('login-password').value;
6
7     // Codifica le credenziali in Base64
8     const credentials = btoa(`${email}:${pass}`);
9
10    // Esegue la chiamata di login
11    const response = await fetch('php/api/login.php', {
12        headers: { 'Authorization': `Basic ${credentials}` }
13    });
14
15    if (response.ok) {
16        const data = await response.json();
17        sessionStorage.setItem('user_auth', credentials);
18        sessionStorage.setItem('user_role', data.role);
19        updateUI(true, data);
20    } else {
21        document.getElementById('login-error').innerText = "Credenziali errate.";
22    }
23});
24
25 // Gestione click sul tasto Registrazione
26 document.getElementById('register-button').addEventListener('click', async () => {
27     // Recupera i valori rimuovendo gli spazi vuoti
28     const nickname = document.getElementById('reg-nickname').value.trim();
29     const email = document.getElementById('reg-email').value.trim();
30     const password = document.getElementById('reg-password').value.trim();
31
32     const errorBox = document.getElementById('reg-error');
33
34     // Verifica che tutti i campi siano compilati
35     if (!nickname || !email || !password) {
36         errorBox.innerText = "Tutti i campi sono obbligatori!";
37         return;
38     }
39
40     // Esegue la chiamata di registrazione

```

```

40     const response = await fetch('php/api/register.php', {
41         method: 'POST',
42         headers: { 'Content-Type': 'application/json' },
43         body: JSON.stringify({ nickname, email, password })
44     );
45
46     const result = await response.json();
47
48     if (response.ok) {
49         alert("Registrazione avvenuta con successo! Ora fai il login.");
50         toggleAuth('login'); // funzione per rimuovere la ui della registrazione
51     } else {
52         errorBox.innerText = result.error;
53     }
54 });

```

5.3.1 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.

```

1 // Controlla lo stato della sessione al caricamento della pagina
2 window.onload = async () => {
3     let isAuthenticated = false;
4     let userData = null;
5
6     if (sessionStorage.getItem('user_auth')) {
7         const response = await fetchAuth('php/api/login.php');
8
9         if (response.ok) {
10             isAuthenticated = true;
11             userData = await response.json();
12         }
13     }
14
15     // Se l'utente non è autenticato, mostra il form di login
16     if (!isAuthenticated) {
17         toggleAuth('login');
18     }
19
20     updateUI(isAuthenticated, userData);
21 };

```

6 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.

6.1 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.

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

6.2 Recupero Risorse Pubbliche (richiedono comunque l'autenticazione)

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

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

Recupero della lista completa dei professori registrati nella piattaforma.

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

6.3 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".

```
1 curl.exe -u "marco@gmail.com:marco123"
2 http://localhost:8080/php/api/teachers.php?strumento=Pianoforte
3
4 [{"id":2,"nickname":"Bono","profile_picture":"default.png",
5 "instruments":"Chitarra, Pianoforte"}, {"id":3,"nickname":"PaulMcCartney",
6 "profile_picture":"default.png","instruments":"Basso, Pianoforte, Chitarra"}]
```

6.4 Gestione Appuntamenti (CRUD)

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

1. Visualizzazione appuntamenti esistenti:

```
1 curl.exe -u "marco@gmail.com:marco123" http://localhost:8080/php/api/appointments.php
2 [{"id":21,"datetime":"2026-01-26 17:00:00",
3 "meeting_link":"https:\/\/meet.google.com\/9e856fce5","partner_name":"Bono",
4 "partner_image":"default.png"}, {"id":24,"datetime":"2026-01-27 15:00:00",
5 "meeting_link":"https:\/\/meet.google.com\/be81e8963f",
6 "partner_name":"PaulMcCartney","partner_image":"default.png"}, {"id":23,"datetime":"2026-01-28 14:00:00",
7 "meeting_link":"https:\/\/meet.google.com\/d6d9b0b29f",
8 "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.

```
1 curl.exe --% -X POST -u "marco@gmail.com:marco123"
2 -H "Content-Type: application/json"
3 -d "{ \"teacher_id\": 2, \"datetime\": \"2026-02-10 15:00:00\" }"
4 http://localhost:8080/php/api/appointments.php
5
6 {"message": "Prenotazione confermata!",
7 "link": "https:\/\/meet.google.com\/6d2599dcf8"}
```

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

```
1 curl.exe --% -X DELETE -u "marco@gmail.com:marco123"
2 -H "Content-Type: application/json" -d "{ \"id\": 22 }"
3 http://localhost:8080/php/api/appointments.php
4
5 {"message": "Lezione cancellata con successo."}
```

7 Web Pages

In questa sezione vengono presentati gli screenshot delle principali schermate dell'applicazione.

7.1 Schermata di Login

La pagina di ingresso permette all'utente di autenticarsi.

The screenshot shows the 'Accedi a Musicson' (Login) page. At the top left is the 'Musicson' logo. Below it, the title 'Accedi a Musicson' is centered. There are two input fields: 'Email' and 'Password', followed by a blue 'Entra' (Enter) button. Below the fields is a link 'Non hai un account? [Registrati qui](#)' (Don't have an account? [Register here](#)).

Figura 1: Pagina di Login

7.2 Schermata di Registrazione

Il form per creare un nuovo account studente o insegnante.

The screenshot shows the 'Crea Account' (Create Account) page. At the top left is the 'Musicson' logo. Below it, the title 'Crea Account' is centered. There are three input fields: 'Nickname (Nome)', 'Email', and 'Password', followed by a blue 'Registrati' (Register) button. Below the fields is a link 'Hai già un account? [Accedi](#)' (Do you already have an account? [Log in](#)).

Figura 2: Pagina di Registrazione

7.3 Home Page e Ricerca

La dashboard principale dove l'utente può cercare il professore in base allo strumento che si vuole imparare.

The screenshot shows the Musicson home page. At the top, there is a navigation bar with the logo "Musicson", the user's name "Ciao Marco", and links for "Home", "Mie Lezioni", "Profilo", and "Esci". Below the navigation, a search bar contains the text "Chitarra" and a blue button labeled "Cerca Maestri". A large white box labeled "Cosa vuoi imparare oggi?" is centered above a grid of teacher profiles. The grid consists of seven cards arranged in two rows. The first row contains four cards: "Paul McCartney" (Strumenti: Basso, Pianoforte, Chitarra), "Jhon Lennon" (Strumenti: Chitarra), "Jimmy Page" (Strumenti: Chitarra), and "Jack White" (Strumenti: Chitarra). Each card has a "Prenota Lezione" button. The second row contains two cards: "The Edge" (Strumenti: Chitarra) and "Eddie Van Halen" (Strumenti: Chitarra).

Figura 3: Home Page

7.4 Risultati della Ricerca

Lista dei professori che insegnano lo strumento selezionato.

The screenshot shows the search results page for "Chitarra" on the Musicson platform. The layout is identical to the home page, with a search bar at the top containing "Chitarra" and a "Cerca Maestri" button. Below it is a "Cosa vuoi imparare oggi?" box. The main content is a grid of seven teacher profiles in two rows. The first row includes "Paul McCartney" (Strumenti: Basso, Pianoforte, Chitarra), "Jhon Lennon" (Strumenti: Chitarra), "Jimmy Page" (Strumenti: Chitarra), and "Jack White" (Strumenti: Chitarra). The second row includes "The Edge" (Strumenti: Chitarra) and "Eddie Van Halen" (Strumenti: Chitarra). Each profile card features a "Prenota Lezione" button.

Figura 4: Risultati Ricerca Professori (Vista Griglia)

7.5 Modale di Prenotazione

Popup che mostra gli slot orari disponibili per il professore scelto.

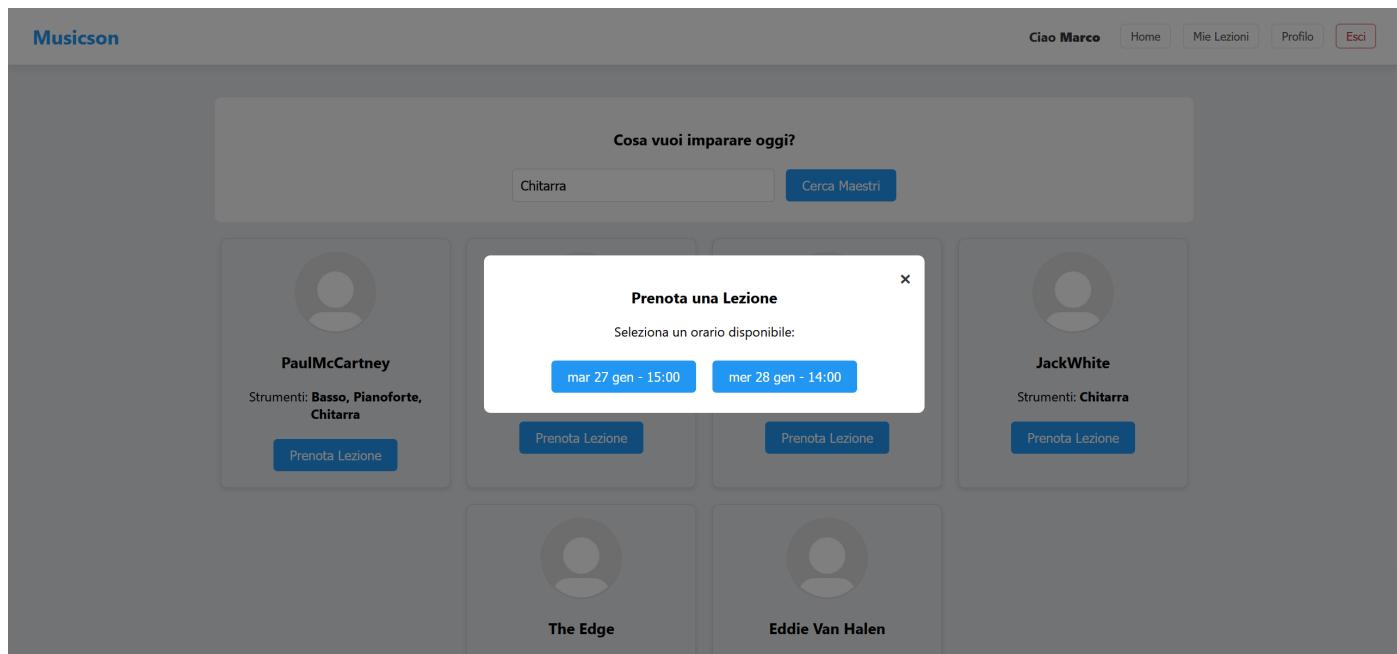


Figura 5: Modale Prenotazione Lezione

7.6 Profilo Utente e Gestione Orari

Sezione per la gestione del profilo e, per i docenti, l'inserimento delle disponibilità.

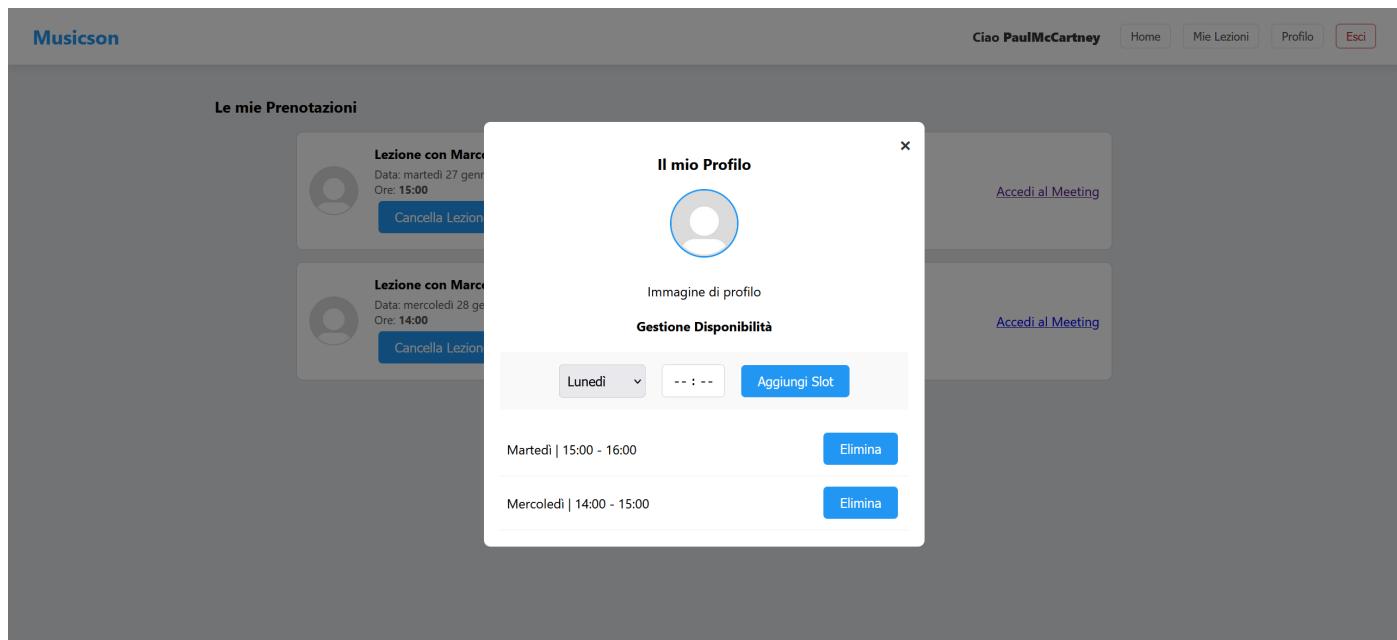


Figura 6: Profilo e Gestione Disponibilità

7.7 Mie Lezioni

Lista degli appuntamenti prenotati con possibilità di cancellazione.

The screenshot displays the 'Mie Lezioni' section of the Musicson application. At the top, there is a navigation bar with the logo 'Musicson' and user information 'Ciao Marco' along with links for 'Home', 'Mie Lezioni', 'Profilo', and 'Esci'. Below the navigation, the heading 'Le mie Prenotazioni' is visible. Two lesson entries are listed:

- Lezione con PaulMcCartney**
Data: martedì 27 gennaio 2026
Ore: 15:00
[Cancella Lezione](#) [Accedi al Meeting](#)
- Lezione con PaulMcCartney**
Data: mercoledì 28 gennaio 2026
Ore: 14:00
[Cancella Lezione](#) [Accedi al Meeting](#)

Figura 7: Pagina Mie Lezioni

8 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.