Software Requirements Specification

for

<Sound Sanctuary>

**Version 1.5 approved**

**Prepared by <Alessandro Trasacco>**

**<Università degli studi della Campania Luigi Vanvitelli>**

**<15/07/2025>**

**Sommario**

**Sommario [ii](#_7aj7hd6jo1qr)**

**Cronologia Revisioni [ii](#_pvnrwcgfmht3)**

**1. Introduzione [1](#_xjjkhrfq2ixz)**

1.1 Idea di Progetto [1](#_bh8kevty784c)

1.2 Pubblico e Suggerimenti di Lettura [1](#_y7h3ltfj5fw4)

**2. Specifica dei Requisiti [2](#_aa7zztg8sy26)**

2.1 Requisiti Funzionali [2](#_s5wwwrhboqhy)

2.2 Requisiti Non Funzionali [2](#_e4wy05kh33b)

**3. Descrizione Generale [3](#_cclzh9xlqhr2)**

3.1 Funzionalità di Sistema [3](#_9vov0rdurav)

3.2 Diagramma dei Casi D'Uso [3](#_asm041sgifo1)

3.3 Scenari [3](#_562e3f438v34)

3.4 Diagramma della Classi [3](#_v2tqytsjpux8)

**4. Progettazione Logica e Concettuale [4](#_pl91p09fkcdh)**

4.1 Progettazione Logica [4](#_a10t4pmxqpv3)

4.2 Progettazione Concettuale [4](#_45skl4r0035e)

**5. Tests 5**

5.1 Login test 5

**Cronologia delle revisioni**

| **Nome** | **Data** | **Motivo delle modifiche** | **Versione** |
| --- | --- | --- | --- |
| Introduzione | 01/06/2025 | Descrizione dell’introduzione del documento e Specifica dei Requisiti | 1.0 |
| Descrizione | 04/06/2025 | Descrizione generale e Casi d’Uso | 1.1 |
| Scenari | 06/06/2025 | Scenari e Diagramma delle Classi | 1.2 |
| Progettazione Concettuale e Logica | 09/06/2025 | Aggiunta Diagramma E-R con Entità, attributi e Relazioni | 1.3 |
| Finalizzazione | 30/06/2025 | Test e descrizioni finali del documento | 1.4 |
| Modifica | 02/07/2025 | Modifica parti approssimative | 1.5 |

**1. Introduzione**

**1.1 Idea di progetto**

Sound Sanctuary è una piattaforma web pensata per semplificare e ottimizzare la gestione di uno studio di registrazione professionale. L’applicazione consente ai musicisti di prenotare sessioni in sala specificando la strumentazione vintage o di backline necessaria, mentre ai produttori è garantito l’accesso remoto per la revisione dei progetti audio.

**1.2 Pubblico e Suggerimenti di Lettura**

Questo documento è destinato a:

* Team di sviluppo che implementeranno il sistema.
* Project manager per il coordinamento complessivo.
* Ingegneri della qualità per le attività di testing.
* Stakeholder aziendali per la validazione dei requisiti funzionali.

Il documento è organizzato in sezioni che descrivono la panoramica generale del sistema, i requisiti, le funzionalità principali, la Progettazione sia Concettuale che Logica e i Test finali. Si suggerisce ai lettori di iniziare con la descrizione generale e le funzionalità del sistema, per poi proseguire con i dettagli della Progettazione e dei Test.

**2. Specifica Dei Requisiti**

**2.1 Requisiti Funzionali**

* + 1. **Registrazione Musicista**

Il sistema consente agli utenti di registrarsi come musicisti, inserendo dati personali (nome, cognome, username, email, data di nascita, genere musicale, ecc.). La registrazione è necessaria per accedere alle funzionalità riservate del portale.

* + 1. **Login**

Gli utenti registrati possono effettuare l’accesso al sistema tramite username e password. Il sistema verifica le credenziali per garantire l’autenticità dell’utente.

* + 1. **Logout**

Gli utenti possono terminare la sessione corrente attraverso la funzione di logout, che invalida la sessione e li riporta alla schermata pubblica.

* + 1. **Prenotazione Sala**

I musicisti possono visualizzare la disponibilità delle sale di registrazione e prenotare una sessione selezionando data, orario.

* + 1. **Prenotazione Equipment**

Durante la prenotazione di una sala, i musicisti possono selezionare strumenti musicali e attrezzature specifiche di cui necessitano per la sessione**.**

* + 1. **Prenotazione Sessioni**

Ogni prenotazione definisce una sessione di registrazione con informazioni su orari di inizio/fine.

* + 1. **Gestione Catalogo Strumenti**

Il sistema gestisce un catalogo di strumenti musicali (vintage e non) disponibili nelle diverse sale.

* + 1. **Editor Audio**I produttori possono accedere a un editor audio web-based per modificare i file registrati dai musicisti, con funzionalità come visualizzazione della forma d’onda ed accesso ad un editor online.
    2. **Upload**

I produttori possono caricare file audio che verranno associati a una sessione, musicista, ecc.. e poi salvati nell’archivio sia del produttore che del musicista.

* + 1. **Gestione File Audio**

I file audio caricati vengono gestiti tramite l’archivio (per produttori o musicisti), con possibilità di ascolto,scaricamento(per produttore e musicista), modifica, eliminazione(solo per produttore).

* + 1. **Visualizzazione Archivio Produttore**

Il produttore può visualizzare tutte le registrazioni associate alle sessioni di cui è stato responsabile. Ogni file ha dettagli come nome utente musicista, data strumenti,ecc…

* + 1. **Visualizzazione Archivio Musicista**

Il musicista può accedere al proprio archivio per rivedere i file audio delle sessioni effettuate, con possibilità di download o ascolto.

* + 1. **Monitoraggio Ore Utilizzo Strumenti**

Il sistema tiene traccia delle ore di utilizzo di ciascuno strumento in base alle sessioni in cui è stato selezionato. Superata una soglia (es. 20 ore), viene richiesta una revisione.

* + 1. **Segnalazione Revisione Strumenti**

Quando uno strumento supera la soglia di utilizzo, compare una notifica per i produttori che indica la necessità di effettuare una revisione. Il produttore può segnare la revisione come completata.

* + 1. **Gestione Prenotazioni (Produttore)**

I produttori possono visualizzare, accettare o rifiutare le richieste di prenotazione ricevute dai musicisti, con un’interfaccia per il monitoraggio dello stato.

* + 1. **Gestione Prenotazioni (Musicista)**

I musicisti possono gestire le proprie prenotazioni attive, verificarne lo stato (in attesa, approvata, rifiutata). Potrà cancellare le prenotazioni in attessa.

**2.2 Requisiti Non Funzionali**

* + 1. **Sicurezza**

Il software dovrà garantire il trattamento di dati

personali e delle credenziali di accesso dei propri utenti con riservatezza, i dati sono salvati tramite crittografia.

* + 1. **Affidabilità e Disponibilità**

Il software dovrà essere disponibile agli utenti 24/7 per almeno l’85% di un anno solare.

* + 1. **Portabilità e Scalabilità**

Il software dovrà funzionare e operare correttamente sui seguenti sistemi operativi: Windows, Mac IOS, Linux.

* + 1. **Performance**

Il software dovrà garantire l’esecuzione delle varie operazioni presenti in modo rapido, senza tempi di attesa significativi.

**3. Descrizione Generale**

* 1. **Funzionalità di sistema**

Lo scopo del software è quello di offrire una piattaforma completa per la gestione di tutte le attività operative di uno studio di registrazione, suddividendo le funzionalità in base al ruolo dell’utente (musicista o produttore).

Dopo aver effettuato la registrazione o il login, i musicisti possono:

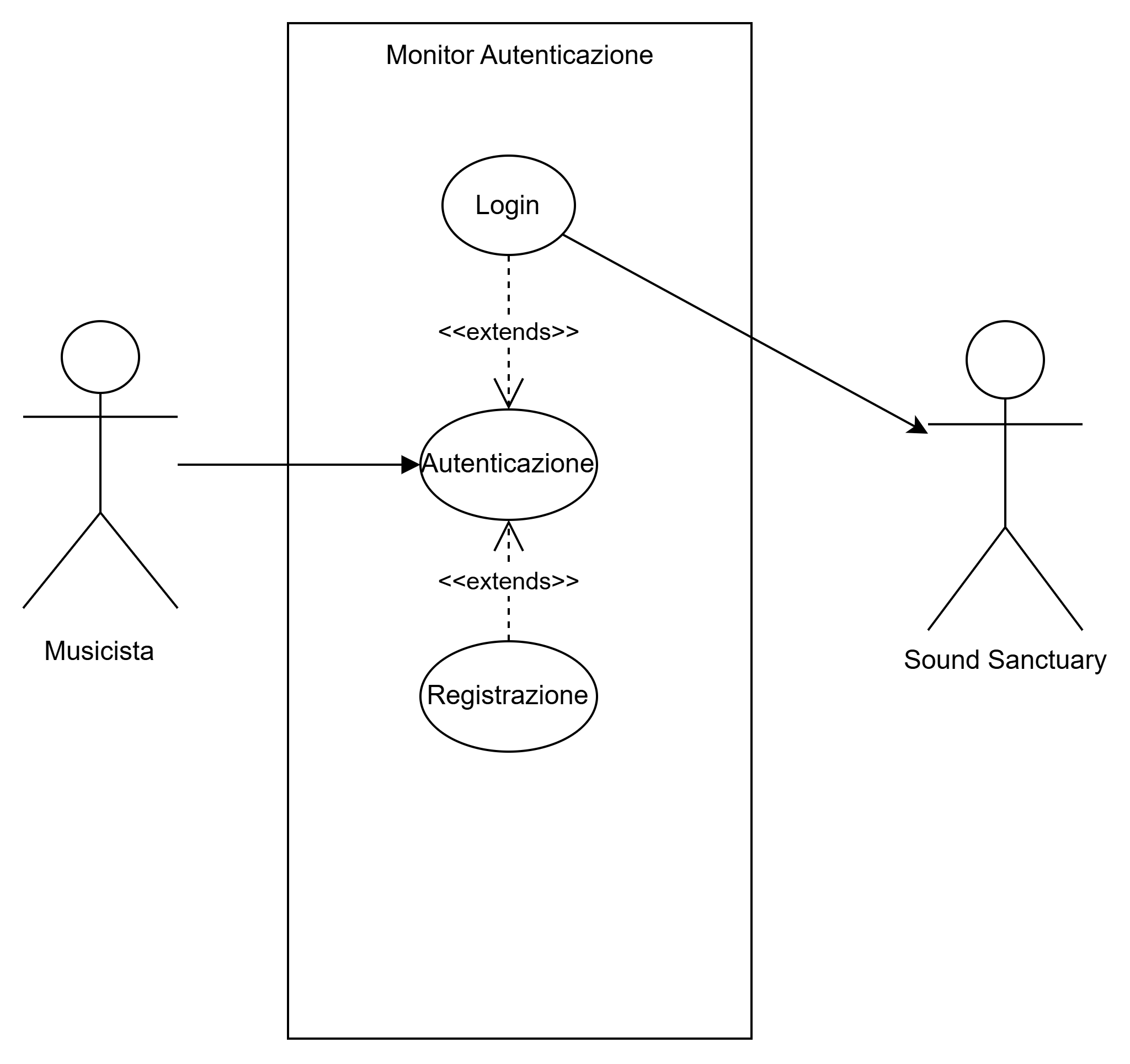
* Prenotare sessioni in studio indicando data, orario e strumentazione necessaria.
* Consultare le proprie prenotazioni, ricevere notifiche su eventuali esiti (accettazione o rifiuto) e accedere alle registrazioni audio effettuate.

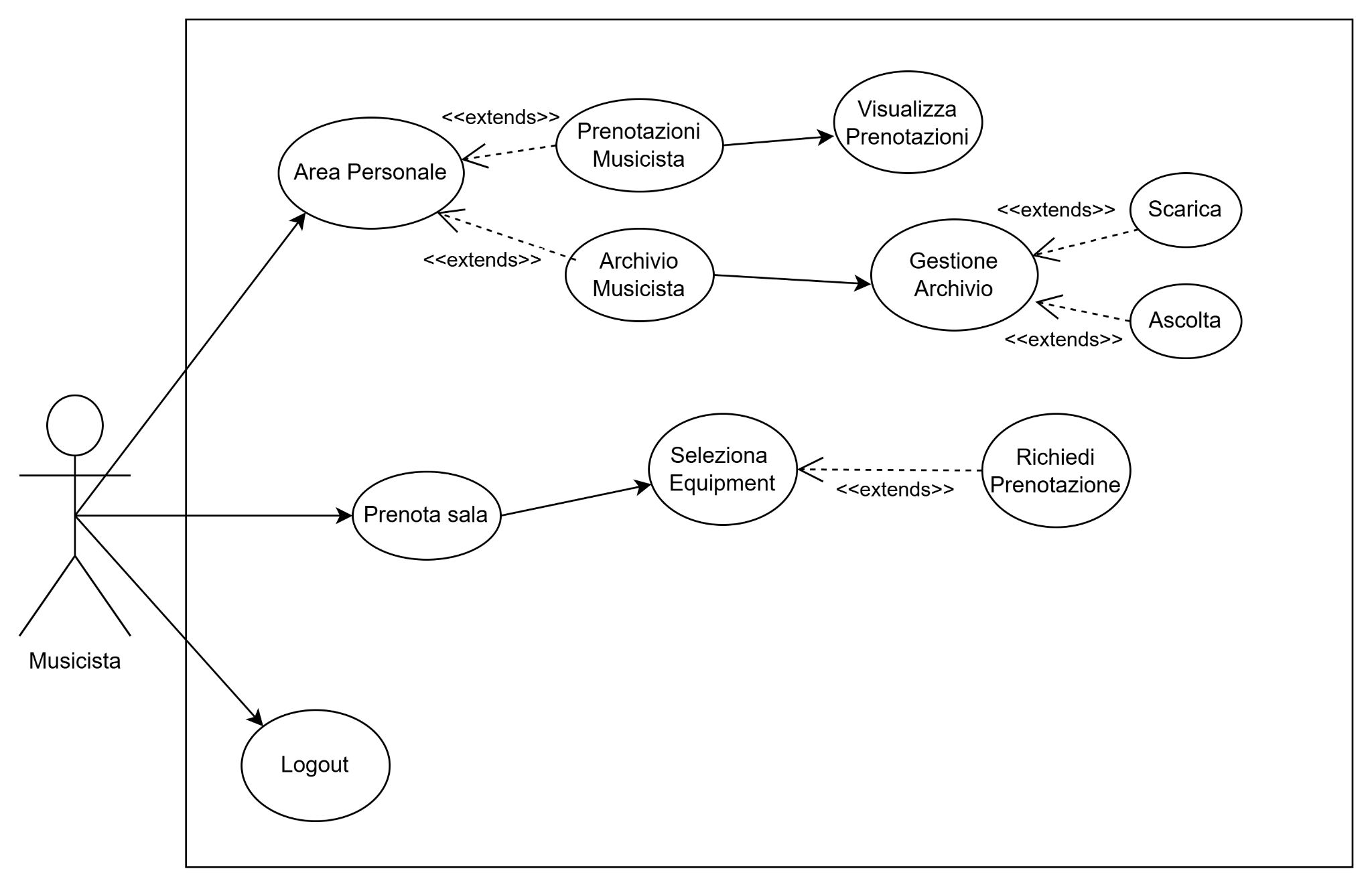
I produttori, invece, hanno accesso a:

* Un pannello dedicato per gestire le prenotazioni ricevute, assegnare o negare conferme, e controllare la disponibilità della sala.
* Una sezione per monitorare lo stato di utilizzo degli strumenti musicali, dove viene automaticamente tracciato il numero di ore d’impiego.
* Un sistema di manutenzione programmata, che segnala quando uno strumento supera le 20 ore di utilizzo e richiede revisione.
  1. **Diagrammi dei Casi d’Uso**

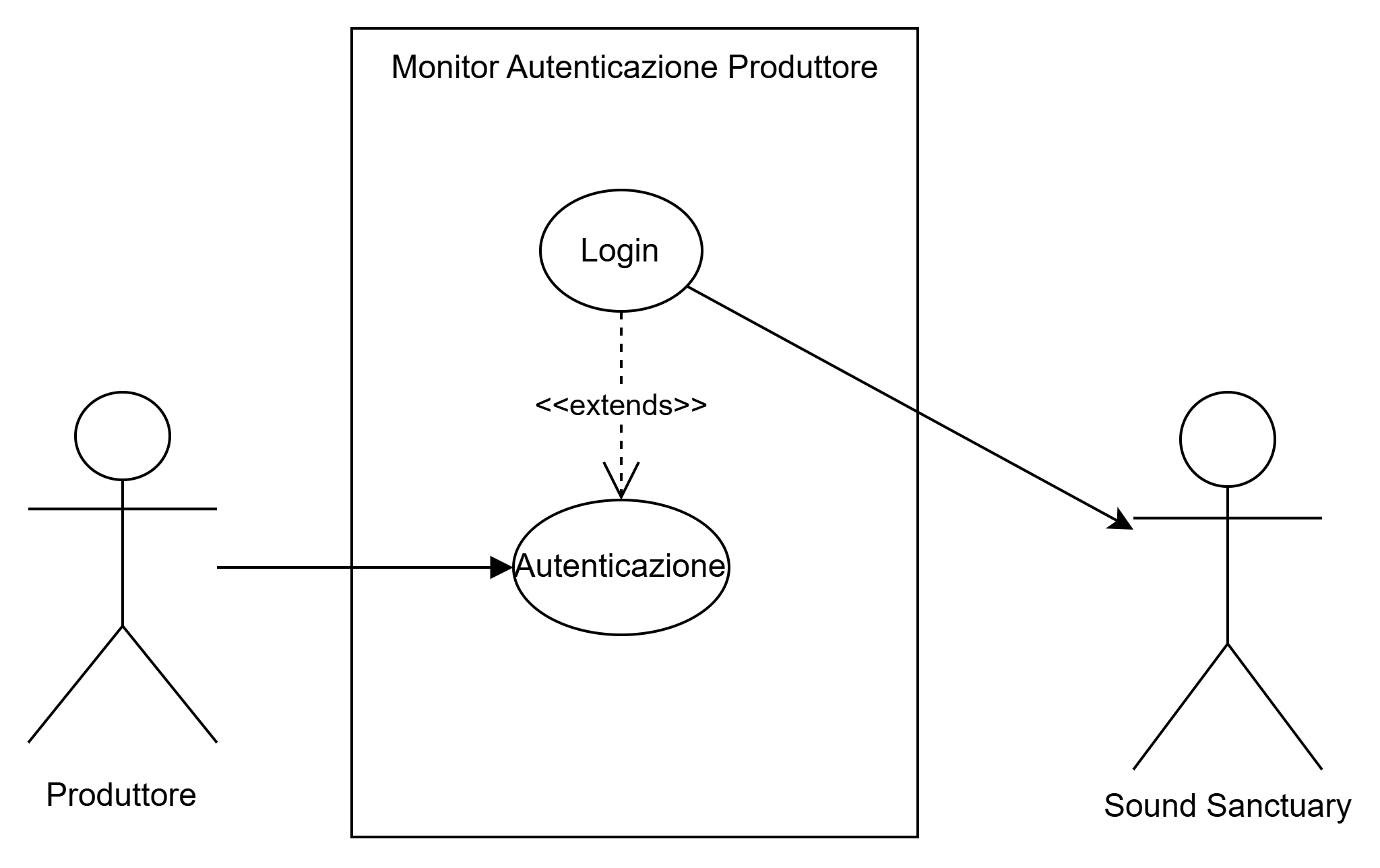
La descrizione dettagliata di tutti i requisiti con lo schema dei casi d’uso.

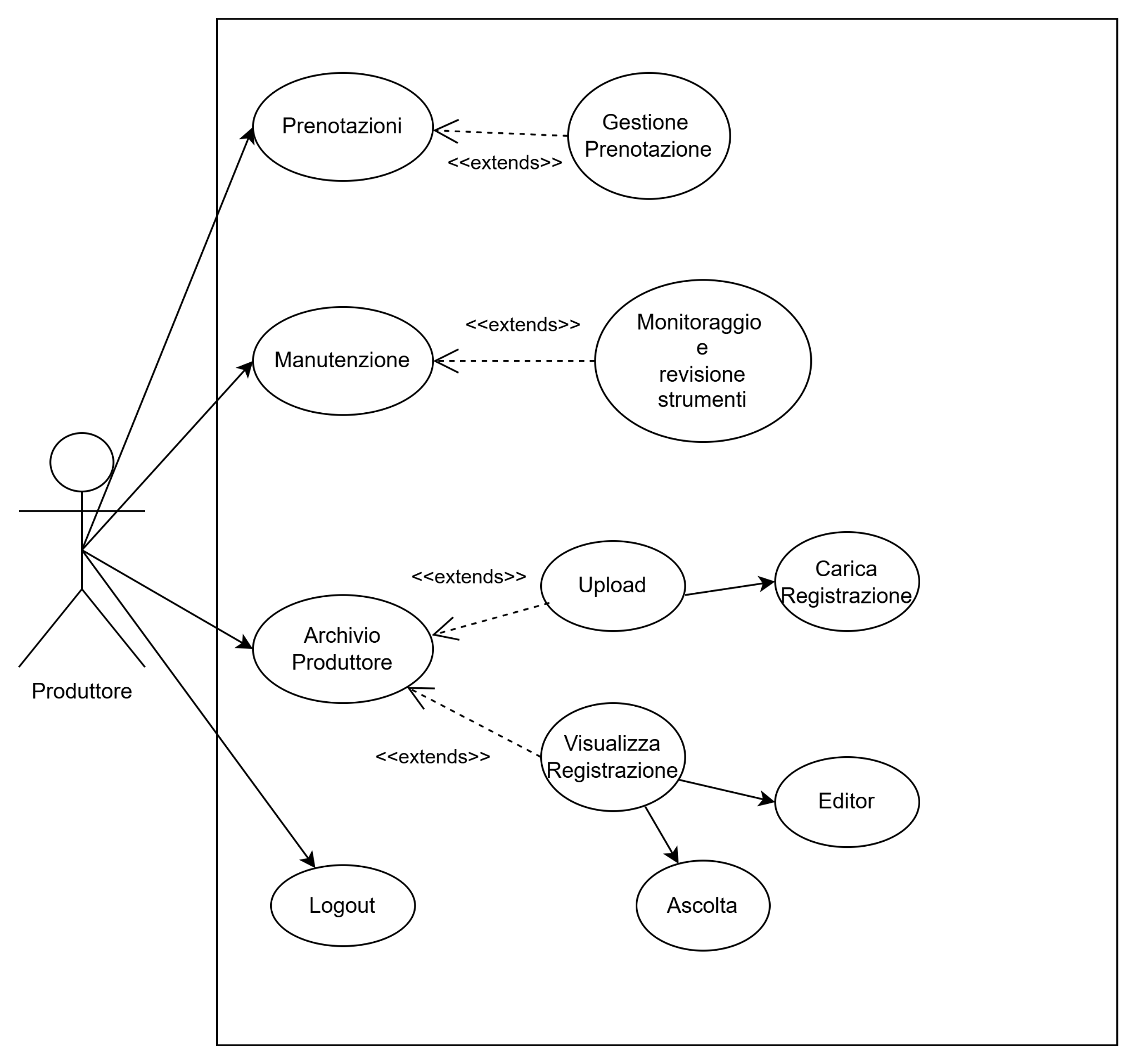
* + 1. **Casi d’Uso Musicista**





* + 1. **Casi d’Uso Produttore**

****

****

* 1. **Scenari**

Definizione più precisa di alcuni casi d’uso attraverso gli scenari.

| 3.3.1Per caso d’uso: “Registrazione” |
| --- |
| Attore: Musicista |
| Precondizioni: l’utente non deve risultare già registrato |
| Sequenza degli Eventi:   1. L’utente sceglie di registrarsi e clicca su “Musicista”; 2. L’utente inserisce i dati necessari (nome, cognome, username,data di nascita, luogo di nascita, email, genere musicale,password, conferma password); 3. Clicca su “Registrati”. |
| Postcondizioni: il musicista è registrato. |
| Scenario Alternativo 1: Se l’utente inserisce credenziali già registrate nel database, apparirà il messaggio “Questa e-mail è già registrata”. |

| *3.3.2 Per caso d’uso: “Login”* |
| --- |
| Attore: Musicista, Produttore |
| Precondizioni: l’utente deve risultare già registrato |
| Scenario:   1. L’utente clicca su “Login”; 2. L’utente inserisce usernamel e password; 3. Clicca su “Login”. |
| Scenario Alternativo 1: Se l’utente inserisce l’username sbagliato, apparirà il messaggio di errore “Utente non trovato”.  Scenario Alternativo 2: Se l’utente inserisce la password sbagliata apparirà il messaggio di errore “Password errata”. |

| *3.3.3 Per caso d’uso: “Logout”* |
| --- |
| Attore: Utente |
| Precondizioni: l’utente deve aver già effettuato l’accesso |
| Scenario:   1. L’utente clicca su “Logout”. |
| Postcondizioni: L’utente non è più loggato all’interno del suo account |

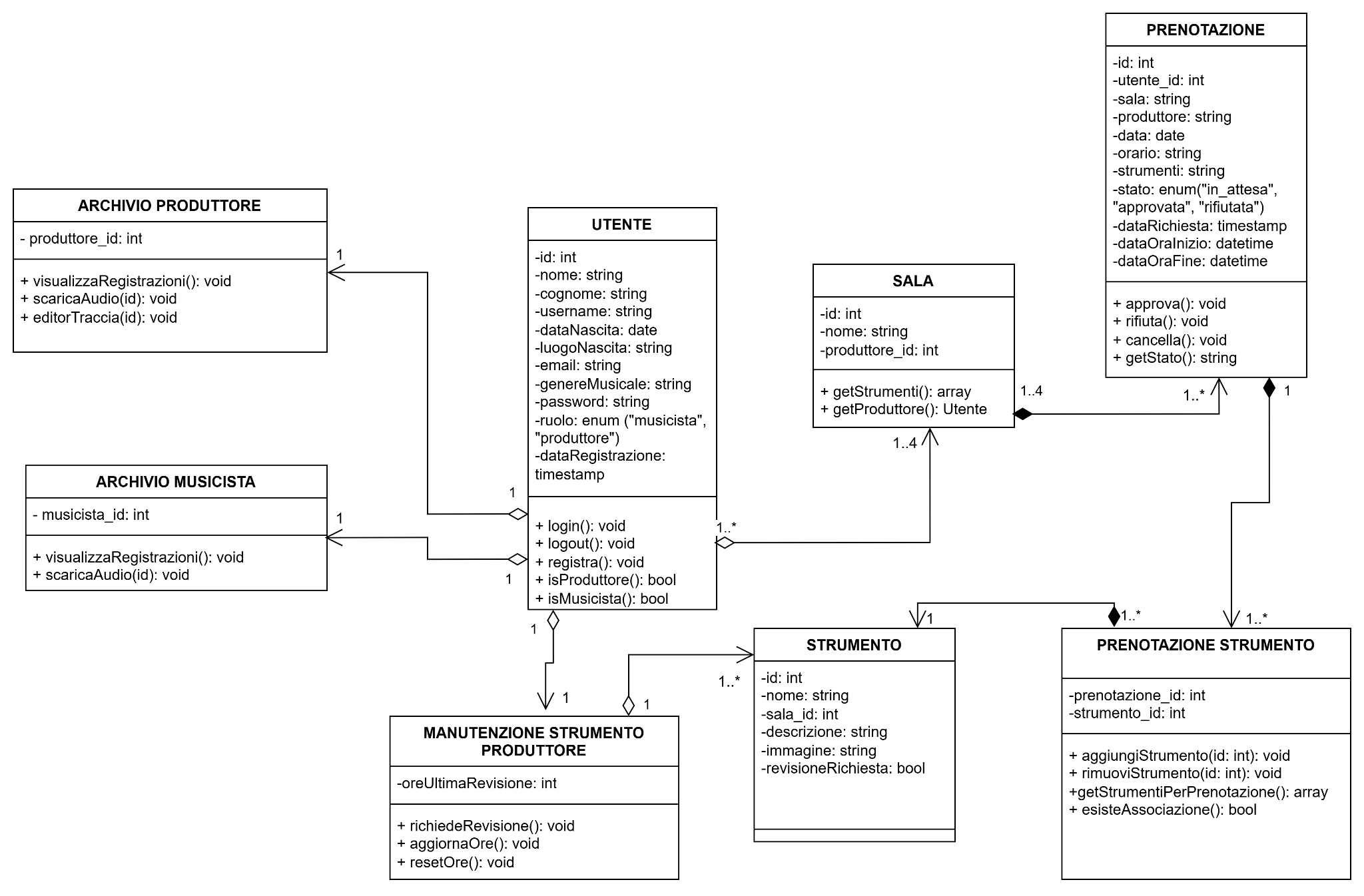
| *3.3.4 Per caso d’uso: “Prenotazione sala”* |
| --- |
| Attore: Musicista |
| Precondizioni: Il musicista deve aver già effettuato l’accesso. |
| Scenario:   1. Il musicista seleziona la sala 2. Seleziona la data 3. Seleziona orario sessione 4. Seleziona equipment 5. Richiede prenotazione |
| Postcondizioni: Il musicista effettua con successo la prenotazione. |
| Scenario Alternativo: c’è conflitto di sessione e apparirà un messaggio di warning che avvisa il musicista del conflitto di orario. |

| *3.3.5 Per caso d’uso: “Gestione Prenotazione Produttore”* |
| --- |
| Attore: Produttore |
| Precondizioni: il produttore deve aver già effettuato l’accesso. |
| Scenario:   1. Il produttore clicca su prenotazioni 2. Visualizza le prenotazioni 3. Accetta o rifiuta una Prenotazione |
| Postcondizioni: Saranno inviati l’accettazione o il rifiuto al musicista. |
| Scenario Alternativo 1: Se non ci sono prenotazioni, apparirà la scritta nessuna prenotazione effettuata. |

| *3.3.6 Per caso d’uso: “Gestione Prenotazione Musicista”* |
| --- |
| Attore: Musicista |
| Precondizioni: Il musicista deve aver già effettuato l’accesso. |
| Scenario:   1. Il musicista clicca su “Area Personale”; 2. Il musicista clicca su “Le tue Prenotazioni” |
| Postcondizioni: Il musicista visualizza le sue prenotazioni. |
| Scenario Alternativo 1: Se non ha effettuato prenotazioni, apparirà la scritta nessuna prenotazione effettuata.  Scenario Alternativo 2: Se la prenotazione è in attesa, potrà cancellarla. |

| *3.3.7 Per caso d’uso: “*Segnalazione Revisione Strumenti*”* |
| --- |
| Attore: Produttore |
| Precondizioni: il produttore deve aver già effettuato l’accesso. |
| Scenario:   1. Il produttore clicca su “Manutenzione” 2. Si apre la tabella |
| Postcondizioni: il Produttore visualizza con successo la tabella e compie delle azioni. |
| Scenario Alternativo 1: Se non sono state effettuate sessioni, non apparirà la sezione della revisione degli strumenti  Scenario Alternativo 2: Superate le 20 ore di utilizzo dello strumento apparirà la scritta “Revisione necessaria”, una volta effettuata la revisione il produttore potrà cliccare sulla scritta e azzerare lo stato, ogni 20 ore apparirà la scritta. |

* 1. **Diagramma delle Classi**



1. **Progettazione Logica e Concettuale**
   1. **Progettazione Logica**

La progettazione logica di un database si concentra sulla creazione di tabelle che rappresentano le entità e le relazioni della progettazione concettuale, evidenziando i tipi di dati per gli attributi, le chiavi primarie, le chiavi esterne, i vincoli e le relazioni tra le tabelle.

1. **Tabella utenti**

Contiene tutti gli utenti registrati, siano essi **musicisti** o **produttori**. Gestisce anche i dati anagrafici e di accesso.

CREATE TABLE `utenti` (

`id` int(10) UNSIGNED NOT NULL,

`nome` varchar(50) NOT NULL,

`cognome` varchar(50) NOT NULL,

`username` varchar(50) NOT NULL,

`data\_nascita` date NOT NULL,

`luogo\_nascita` varchar(100) NOT NULL,

`email` varchar(100) NOT NULL,

`genere\_musicale` varchar(50) NOT NULL,

`password` varchar(255) NOT NULL,

`ruolo` enum('musicista','produttore') NOT NULL,

`data\_registrazione` timestamp NOT NULL DEFAULT current\_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

1. **Tabella strumenti**

Contiene gli strumenti musicali disponibili nelle varie sale. Ogni strumento è associato a una sala e può avere uno stato di revisione.

CREATE TABLE `strumenti` (

`id` int(11) NOT NULL,

`nome` varchar(100) NOT NULL,

`sala\_id` int(11) NOT NULL,

`sala` varchar(50) DEFAULT NULL,

`descrizione` text DEFAULT NULL,

`immagine` varchar(255) DEFAULT NULL,

`revisione\_richiesta` tinyint(1) DEFAULT 0,

`ore\_ultima\_revisione` int(11) DEFAULT 0

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

1. **Tabella registrazione**

CREATE TABLE `registrazioni` (

`id` int(11) NOT NULL,

`titolo` varchar(255) NOT NULL,

`musicista\_id` int(10) UNSIGNED NOT NULL,

`sala` varchar(100) NOT NULL,

`data\_registrazione` timestamp NOT NULL DEFAULT current\_timestamp(),

`file\_path` varchar(255) NOT NULL,

`note` text DEFAULT NULL,

`inserito\_da` int(10) UNSIGNED NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

1. **Tabella prenotazioni strumenti**

Tabella di relazione many-to-many tra prenotazioni e strumenti. Permette di associare più strumenti a una singola prenotazione.

CREATE TABLE `prenotazione\_strumenti` (

`prenotazione\_id` int(11) NOT NULL,

`strumento\_id` int(11) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

1. **Tabella prenotazioni**

Rappresenta le prenotazioni effettuate dai musicisti. Ogni prenotazione include la sala, il produttore assegnato, gli strumenti richiesti, lo stato e le date/ore di inizio e fine.

CREATE TABLE `prenotazioni` (

`id` int(11) NOT NULL,

`utente\_id` int(11) NOT NULL,

`sala` varchar(100) DEFAULT NULL,

`produttore` varchar(100) DEFAULT NULL,

`data` date DEFAULT NULL,

`orario` varchar(20) DEFAULT NULL,

`strumenti` text DEFAULT NULL,

`stato` enum('in\_attesa','approvata','rifiutata') DEFAULT 'in\_attesa',

`data\_richiesta` timestamp NOT NULL DEFAULT current\_timestamp(),

`data\_ora\_inizio` datetime DEFAULT NULL,

`data\_ora\_fine` datetime DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

1. **Tabella sale**

Questa tabella rappresenta le sale di registrazione presenti nello studio e associate a ciascun produttore.

CREATE TABLE `sale` (

`id` int(11) NOT NULL,

`nome` varchar(100) NOT NULL,

`produttore\_id` int(11) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

1. **Tabelle Test**

Questa tabelle sono identiche alle precedenti ma vengono utilizzate solo nei diversi test, non intaccando il database.

CREATE TABLE `prenotazioni` (

`id` int(11) NOT NULL,

`utente\_id` int(11) NOT NULL,

`sala` varchar(100) DEFAULT NULL,

`produttore` varchar(100) DEFAULT NULL,

`data` date DEFAULT NULL,

`orario` varchar(20) DEFAULT NULL,

`strumenti` text DEFAULT NULL,

`stato` enum('in\_attesa','approvata','rifiutata') DEFAULT 'in\_attesa',

`data\_richiesta` timestamp NOT NULL DEFAULT current\_timestamp(),

`data\_ora\_inizio` datetime DEFAULT NULL,

`data\_ora\_fine` datetime DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

CREATE TABLE `registrazioni` (

`id` int(11) NOT NULL,

`titolo` varchar(255) NOT NULL,

`musicista\_id` int(10) UNSIGNED NOT NULL,

`sala` varchar(100) NOT NULL,

`data\_registrazione` timestamp NOT NULL DEFAULT current\_timestamp(),

`file\_path` varchar(255) NOT NULL,

`note` text DEFAULT NULL,

`inserito\_da` int(10) UNSIGNED NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

CREATE TABLE `utenti` (

`id` int(10) UNSIGNED NOT NULL,

`nome` varchar(50) NOT NULL,

`cognome` varchar(50) NOT NULL,

`username` varchar(50) NOT NULL,

`data\_nascita` date NOT NULL,

`luogo\_nascita` varchar(100) NOT NULL,

`email` varchar(100) NOT NULL,

`genere\_musicale` varchar(50) NOT NULL,

`password` varchar(255) NOT NULL,

`ruolo` enum('musicista','produttore') NOT NULL,

`data\_registrazione` timestamp NOT NULL DEFAULT current\_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

* 1. **Progettazione Concettuale**

Per rappresentare i concetti principali e le loro relazioni, utilizziamo un diagramma ER con le seguenti:

**Entità e Attributi**

**Utente:**

ID, Nome, Cognome, Username, Data\_Nascita, Luogo\_Nascita, Email, Genere\_Musicale, Password, Ruolo, Data\_Registrazione

**Sala:**

ID, Nome, Produttore\_ID

**Strumento:**

ID, Nome, Sala\_ID, Descrizione, Immagine, Revisione\_Richiesta, Ore\_Ultima\_Revisione

**Prenotazione:**

ID, Utente\_ID, Sala, Produttore, Data, Orario, Stato, Strumenti (campo libero), Data\_Richiesta, Data\_Ora\_Inizio, Data\_Ora\_Fine

**Prenotazione\_Strumenti:**

Prenotazione\_ID, Strumento\_ID

**Registrazione:**

ID, Titolo, Musicista\_ID, File\_Path, Note, Inserito\_Da, Sala\_ID

**Editor Audio:**

(Componente logico, non tabellare): File Audio, Funzioni: Caricamento, Taglio, **Salvataggio.**

**Archivio:**

(Logico, diviso in ArchivioMusicista e ArchivioProduttore)

**Relazioni:**

**Un Utente può effettuare più Prenotazioni:**

Relazione: Prenota (1:N)

**Un Utente può essere associato a più Registrazioni:**

Come Musicista (musicista\_id) → È Oggetto Di

Come Produttore (inserito\_da) → Carica

**Ogni Sala è associata a più Strumenti:**

Relazione: Contiene (1:N)

**Una Prenotazione può coinvolgere più Strumenti, e uno Strumento può essere richiesto in più Prenotazioni:**

Relazione tramite Prenotazione\_Strumenti (M:N)

**Ogni Registrazione può essere associata a una Sala:**

Relazione logica: AvvieneIn (1:N)

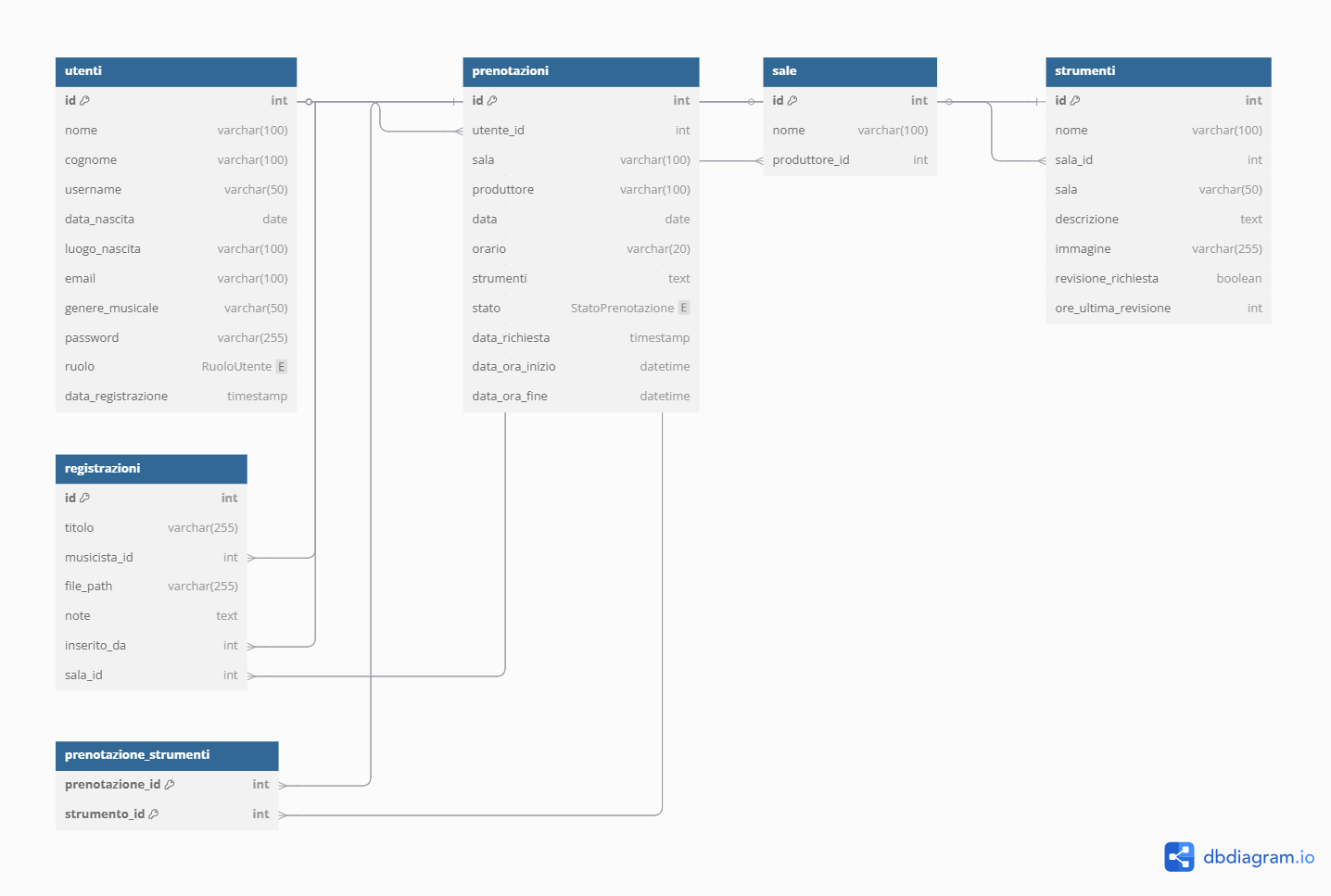
**Ogni Sala è gestita da un Utente Produttore:**

Relazione: Gestisce (1:N)

**Ogni Utente ha accesso al proprio Archivio (musicista o produttore):**

Relazione logica: Possiede (1:1)

**Diagramma E-R**

******

**5. Tests**

Questa è la sezione in cui vengono riportati i vari test eseguiti con phpunit per le verifiche di diverse sezioni della web app.

Ho usato:

PHPUnit (versione 11.x)

PHP 8.2 su ambiente locale XAMPP

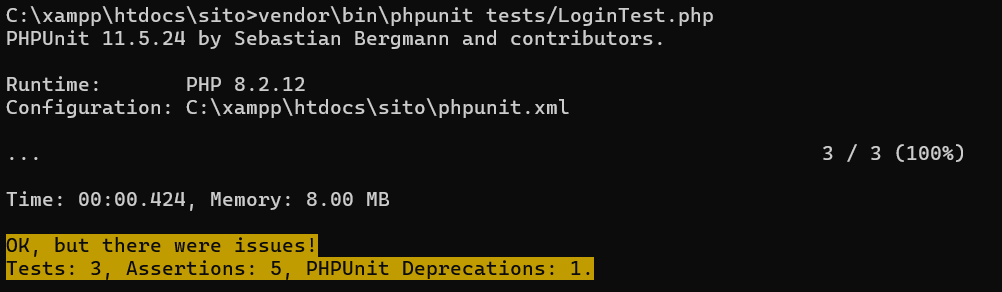
Database separato per i test: studio\_di\_registrazione\_test (replica del DB reale, isolato)

**Fase di Login.**

Obiettivo:  
 Garantire che il sistema di autenticazione funzioni correttamente in diverse situazioni:

* Login con credenziali corrette
* Login con password errata
* Login con un nome utente non registrato

Risultato:



<?php

use PHPUnit\Framework\TestCase;

class LoginTest extends TestCase {

private $conn;

protected function setUp(): void {

$this->conn = new mysqli("localhost", "root", "", "studio\_di\_registrazione\_test");

if ($this->conn->connect\_errno) {

$this->fail("Connessione fallita al DB di test: " . $this->conn->connect\_error);

}

$this->conn->set\_charset("utf8mb4");

$this->conn->query("DELETE FROM utenti WHERE username = 'testuser'");

$password\_hash = password\_hash("password123", PASSWORD\_DEFAULT);

$this->conn->query("

INSERT INTO utenti (

nome, cognome, username, data\_nascita, luogo\_nascita,

email, genere\_musicale, password, ruolo, data\_registrazione

) VALUES (

'Test', 'User', 'testuser', '2000-01-01', 'TestCity',

'testuser@example.com', 'Rock', '$password\_hash', 'musicista', NOW()

)

");

}

protected function tearDown(): void {

$this->conn->query("DELETE FROM utenti WHERE username = 'testuser'");

$this->conn->close();

}

public function testLoginCorretto() {

$username = 'testuser';

$password = 'password123';

$stmt = $this->conn->prepare("

SELECT u.id, u.username, u.password, u.ruolo, s.id AS sala\_id

FROM utenti u

LEFT JOIN sale s ON s.produttore\_id = u.id

WHERE u.username = ?

");

$stmt->bind\_param("s", $username);

$stmt->execute();

$result = $stmt->get\_result();

$user = $result->fetch\_assoc();

$stmt->close();

$this->assertNotNull($user, "L'utente dovrebbe essere trovato nel DB.");

$this->assertTrue(password\_verify($password, $user['password']), "La password deve essere valida.");

$this->assertEquals('musicista', $user['ruolo'], "Il ruolo deve essere 'musicista'.");

}

public function testLoginPasswordErrata() {

$username = 'testuser';

$password = 'sbagliata';

$stmt = $this->conn->prepare("SELECT password FROM utenti WHERE username = ?");

$stmt->bind\_param("s", $username);

$stmt->execute();

$stmt->bind\_result($hash);

$stmt->fetch();

$stmt->close();

$this->assertFalse(password\_verify($password, $hash), "La password sbagliata non deve funzionare.");

}

public function testLoginUtenteNonEsistente() {

$username = 'utente\_inesistente';

$stmt = $this->conn->prepare("SELECT \* FROM utenti WHERE username = ?");

$stmt->bind\_param("s", $username);

$stmt->execute();

$result = $stmt->get\_result();

$stmt->close();

$this->assertEquals(0, $result->num\_rows, "Nessun utente dovrebbe essere trovato.");

}

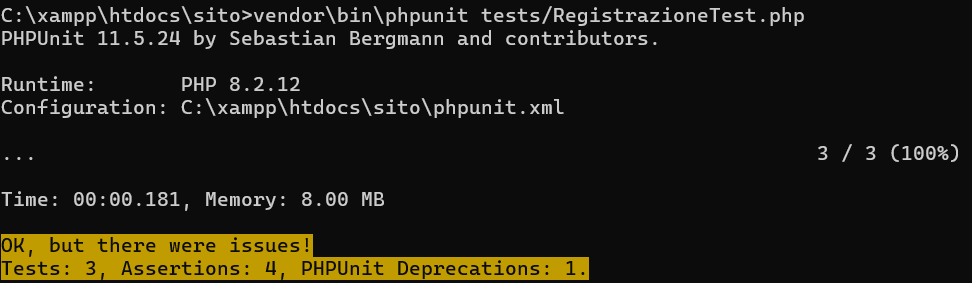
}

**Fase di Registrazione**

Obiettivo:  
Verificare che il sistema consenta la creazione di un nuovo utente in modo sicuro e corretto:

* Registrazione con dati validi e password coincidenti
* Rifiuto della registrazione in caso di email o username già presenti
* Rifiuto della registrazione se le password non coincidono

Risultato:



<?php

use PHPUnit\Framework\TestCase;

class RegistrazioneTest extends TestCase {

private $conn;

protected function setUp(): void {

$this->conn = new mysqli("localhost", "root", "", "studio\_di\_registrazione\_test");

if ($this->conn->connect\_errno) {

$this->fail("Connessione fallita al DB di test: " . $this->conn->connect\_error);

}

$this->conn->set\_charset("utf8mb4");

$this->conn->query("DELETE FROM utenti WHERE username = 'nuovoutente'");

}

protected function tearDown(): void {

$this->conn->query("DELETE FROM utenti WHERE username = 'nuovoutente'");

$this->conn->close();

}

public function testRegistrazioneSuccesso() {

$nome = "Mario";

$cognome = "Rossi";

$username = "nuovoutente";

$data\_nascita = "1990-01-01";

$luogo\_nascita = "Roma";

$email = "mario.rossi@example.com";

$genere = "Rock";

$password = "Password123!";

$ruolo = "musicista";

$password\_hash = password\_hash($password, PASSWORD\_DEFAULT);

$stmt = $this->conn->prepare("

INSERT INTO utenti (

nome, cognome, username, data\_nascita, luogo\_nascita,

email, genere\_musicale, password, ruolo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

");

$stmt->bind\_param("sssssssss", $nome, $cognome, $username, $data\_nascita, $luogo\_nascita, $email, $genere, $password\_hash, $ruolo);

$success = $stmt->execute();

$stmt->close();

$this->assertTrue($success, "La registrazione deve andare a buon fine.");

$stmt2 = $this->conn->prepare("SELECT \* FROM utenti WHERE username = ?");

$stmt2->bind\_param("s", $username);

$stmt2->execute();

$result = $stmt2->get\_result();

$stmt2->close();

$this->assertEquals(1, $result->num\_rows, "L'utente registrato deve esistere nel database.");

}

public function testUsernameDuplicato() {

$this->conn->query("

INSERT INTO utenti (nome, cognome, username, data\_nascita, luogo\_nascita, email, genere\_musicale, password, ruolo)

VALUES ('Mario', 'Rossi', 'nuovoutente', '1990-01-01', 'Roma', 'mario.rossi@example.com', 'Rock', '" . password\_hash("Password123!", PASSWORD\_DEFAULT) . "', 'musicista')

");

$stmt = $this->conn->prepare("SELECT id FROM utenti WHERE email = ? OR username = ?");

$email = 'mario.rossi@example.com';

$username = 'nuovoutente';

$stmt->bind\_param("ss", $email, $username);

$stmt->execute();

$stmt->store\_result();

$this->assertGreaterThan(0, $stmt->num\_rows, "Username o email già presenti devono essere rilevati.");

$stmt->close();

}

public function testPasswordNonCoincide() {

$password = "Password123!";

$conferma = "PasswordDiversa";

$this->assertNotEquals($password, $conferma, "Le password devono risultare diverse.");

}

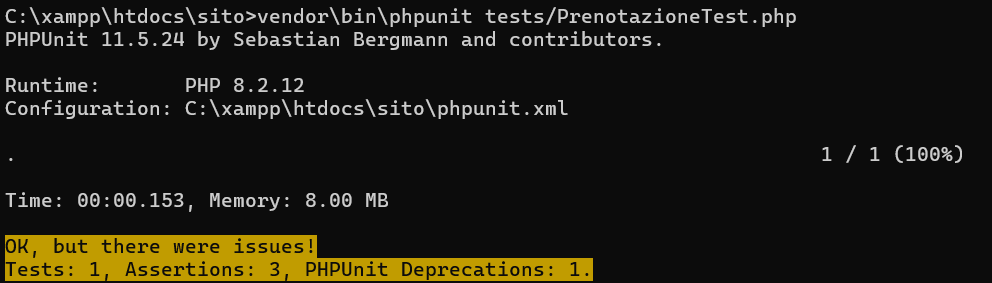
}

### **Fase di Prenotazione**

Obiettivo: Assicurarsi che il sistema consenta al musicista di effettuare una prenotazione e associare uno o più strumenti:

* Inserimento corretto di una prenotazione per una sala disponibile
* Associazione corretta degli strumenti selezionati alla prenotazione
* Verifica della presenza dei dati nel database dopo la prenotazione

Risultato:



<?php

use PHPUnit\Framework\TestCase;

class PrenotazioneTest extends TestCase {

private $conn;

private $utente\_id;

private $produttore\_username = 'testprod';

private $sala\_nome = 'Sala Test';

private $strumento\_id;

protected function setUp(): void {

$this->conn = new mysqli("localhost", "root", "", "studio\_di\_registrazione\_test");

if ($this->conn->connect\_errno) {

$this->fail("Connessione fallita al DB di test: " . $this->conn->connect\_error);

}

$this->conn->set\_charset("utf8mb4");

$this->conn->query("DELETE FROM prenotazione\_strumenti");

$this->conn->query("DELETE FROM prenotazioni");

$this->conn->query("DELETE FROM strumenti WHERE nome = 'Strumento Test'");

$this->conn->query("DELETE FROM sale WHERE nome = 'Sala Test'");

$this->conn->query("DELETE FROM utenti WHERE username IN ('testmusic', 'testprod')");

$this->conn->query("

INSERT INTO utenti (nome, cognome, username, data\_nascita, luogo\_nascita, email, genere\_musicale, password, ruolo, data\_registrazione)

VALUES ('Prod', 'Test', 'testprod', '1980-01-01', 'Città', 'prod@test.it', 'Rock', '" . password\_hash("1234", PASSWORD\_DEFAULT) . "', 'produttore', NOW())

");

$produttore\_id = $this->conn->insert\_id;

$this->conn->query("INSERT INTO sale (nome, produttore\_id) VALUES ('Sala Test', $produttore\_id)");

$this->conn->query("

INSERT INTO utenti (nome, cognome, username, data\_nascita, luogo\_nascita, email, genere\_musicale, password, ruolo, data\_registrazione)

VALUES ('Musicista', 'Test', 'testmusic', '1995-01-01', 'Città', 'music@test.it', 'Jazz', '" . password\_hash("1234", PASSWORD\_DEFAULT) . "', 'musicista', NOW())

");

$this->utente\_id = $this->conn->insert\_id;

$this->conn->query("INSERT INTO strumenti (nome) VALUES ('Strumento Test')");

$this->strumento\_id = $this->conn->insert\_id;

}

protected function tearDown(): void {

$this->conn->query("DELETE FROM prenotazione\_strumenti");

$this->conn->query("DELETE FROM prenotazioni");

$this->conn->query("DELETE FROM strumenti WHERE nome = 'Strumento Test'");

$this->conn->query("DELETE FROM sale WHERE nome = 'Sala Test'");

$this->conn->query("DELETE FROM utenti WHERE username IN ('testmusic', 'testprod')");

$this->conn->close();

}

public function testPrenotazioneInseritaCorrettamente() {

$data = date("Y-m-d", strtotime("+1 day"));

$orario = "14:00-15:00";

$data\_ora\_inizio = $data . " 14:00:00";

$data\_ora\_fine = $data . " 15:00:00";

$stmt = $this->conn->prepare("

INSERT INTO prenotazioni (utente\_id, sala, produttore, data\_ora\_inizio, data\_ora\_fine, stato, data\_richiesta)

VALUES (?, ?, ?, ?, ?, 'in\_attesa', NOW())

");

$stmt->bind\_param("issss", $this->utente\_id, $this->sala\_nome, $this->produttore\_username, $data\_ora\_inizio, $data\_ora\_fine);

$success = $stmt->execute();

$this->assertTrue($success, "La prenotazione deve essere inserita con successo");

$prenotazione\_id = $this->conn->insert\_id;

$stmt2 = $this->conn->prepare("INSERT INTO prenotazione\_strumenti (prenotazione\_id, strumento\_id) VALUES (?, ?)");

$stmt2->bind\_param("ii", $prenotazione\_id, $this->strumento\_id);

$stmt2->execute();

$stmt2->close();

$res = $this->conn->query("SELECT \* FROM prenotazioni WHERE id = $prenotazione\_id");

$this->assertEquals(1, $res->num\_rows, "La prenotazione dovrebbe esistere");

$res2 = $this->conn->query("SELECT \* FROM prenotazione\_strumenti WHERE prenotazione\_id = $prenotazione\_id");

$this->assertEquals(1, $res2->num\_rows, "Lo strumento deve essere associato alla prenotazione");

}

}