Documentazione Approfondita: Web Service Gestione Timbrature

*Fatto da: Picoco Mattia 5infB*

*Tomcat 8.5*

*Apache netbeans 26*

*Jdk 22*

*Maven 3.1*

*Java ee 7 web*

Indice

[1. Scopo del Sistema 3](#_Toc199204805)

[1.1. Obiettivi e Contesto 3](#_Toc199204806)

[1.2. Problemi Risolti e Benefici 3](#_Toc199204807)

[1.3. Utenti e Sistemi Target 3](#_Toc199204808)

[1.4. Confini e Limitazioni (Attuali) 3](#_Toc199204809)

[2. Utilizzo delle API (Web Service REST) 4](#_Toc199204810)

[2.1. Principi e Architettura API 4](#_Toc199204811)

[2.2. URL di Base e Versioning 4](#_Toc199204812)

[2.3. Flussi di Lavoro Tipici 4](#_Toc199204813)

[2.4. Gestione degli Errori e Codici di Stato 5](#_Toc199204814)

[2.5. Esempi di Interazione (curl) 5](#_Toc199204815)

[3. Dettagli di Implementazione 6](#_Toc199204816)

[3.1. Architettura Software 6](#_Toc199204817)

[3.2. Struttura del Database 6](#_Toc199204818)

[3.3. Componenti Chiave del Backend (Java) 6](#_Toc199204819)

[3.3.1. TimbraturaServlet (Controller/Endpoint) 6](#_Toc199204820)

[3.3.2. TimbraturaDAO (Data Access & Business Logic) 6](#_Toc199204821)

[3.3.3. DatabaseManager (Connessione DB) 6](#_Toc199204822)

[3.3.4. Modelli Dati 6](#_Toc199204823)

[3.4. Logica di Business e Validazione 6](#_Toc199204824)

[3.5. Considerazioni sulla Sicurezza (Attuale e Futura) 7](#_Toc199204825)

[3.6. Possibili Estensioni Future 7](#_Toc199204826)

[Appendice codice SQL: 7](#_Toc199204827)

# 1. Scopo del Sistema

## 1.1. Obiettivi e Contesto

L’obiettivo primario di questo sistema è fornire una soluzione **centralizzata, affidabile e automatizzata** per la registrazione e la gestione delle presenze del personale aziendale. Nasce dall’esigenza di superare i limiti dei sistemi manuali o frammentati, offrendo una piattaforma digitale che si integra con l’infrastruttura esistente (lettori RFID) per monitorare gli ingressi e le uscite dei dipendenti in tempo reale.

## 1.2. Problemi Risolti e Benefici

* **Eliminazione Errori Manuali:** Riduce drasticamente le imprecisioni associate alla registrazione manuale degli orari.
* **Dati in Tempo Reale:** Fornisce una visione aggiornata delle presenze, utile per la gestione operativa e la sicurezza.
* **Efficienza Amministrativa:** Semplifica il lavoro dell’ufficio Risorse Umane per il calcolo delle ore lavorate e la gestione delle presenze/assenze.
* **Tracciabilità:** Crea un archivio digitale e immutabile (salvo correzioni autorizzate) degli accessi.
* **Flessibilità:** L’architettura basata su API permette l’integrazione con diversi tipi di lettori e potenzialmente con altri software gestionali (es. paghe).
* **Validazione Coerenza:** Implementa regole (es. ingresso-uscita) per garantire la logicità dei dati registrati.

## 1.3. Utenti e Sistemi Target

* **Sistemi di Lettura RFID:** Sono i *client primari* delle API, inviando dati ogni volta che un tag viene letto.
* **Ufficio Risorse Umane:** Utilizzatori (tramite interfacce dedicate come quella web o altre) per monitorare, correggere e (potenzialmente) estrarre dati per l’elaborazione delle paghe.
* **Manager/Supervisori:** Per verificare la presenza dei propri team.
* **Sviluppatori/Integratori:** Coloro che devono interfacciare nuovi lettori o sistemi esterni con questa piattaforma.

## 1.4. Confini e Limitazioni (Attuali)

Il sistema, nella sua implementazione attuale:

* **Fa:** Registra ingressi/uscite, valida la sequenza, elenca le timbrature odierne, elimina l’ultima timbratura per un utente.
* **Non Fa:** Gestione anagrafica completa dei dipendenti via API, calcolo automatico delle ore lavorate/straordinari, gestione ferie/permessi, reportistica avanzata, autenticazione/autorizzazione API robusta.

# 2. Utilizzo delle API (Web Service REST)

## 2.1. Principi e Architettura API

Il web service è progettato seguendo i principi REST (Representational State Transfer), anche se in modo semplificato:

**Risorsa:** L’entità principale è la “timbratura”, accessibile tramite l’endpoint /timbrature.

**Metodi HTTP:** Utilizza i verbi HTTP standard per definire le azioni:

GET: Per recuperare informazioni (lettura).

POST: Per creare nuove risorse (registrazione).

DELETE: Per rimuovere risorse (eliminazione).

**Stateless:** Ogni richiesta dal client al server deve contenere tutte le informazioni necessarie per essere compresa, senza fare affidamento su sessioni precedenti.

**Interfaccia Uniforme:** L’uso di HTTP e URL standard garantisce un’interfaccia accessibile da una vasta gamma di client.

## 2.2. URL di Base e Versioning

L’URL di base è http://<server>:<porta>/<nome\_applicazione>/api. Attualmente **non è implementato un sistema di versioning** delle API (es. /api/v1/timbrature). Per future evoluzioni, sarebbe consigliabile introdurlo.

## 2.3. Flussi di Lavoro Tipici

Oltre all’ingresso/uscita standard, si consideri la correzione di un errore:

* Mario (RFID0001) esce alle 17:30 (POST ... rfid=RFID0001&totem=USCI0001 -> 201 Created).
* Si accorge di aver dimenticato qualcosa e rientra subito, ma per errore passa di nuovo davanti al lettore di uscita (POST ... rfid=RFID0001&totem=USCI0001).
* Il server risponde 409 Conflict con “Errore: Il dipendente è già registrato come USCITO.”.
* Mario capisce l’errore e passa davanti al lettore di ingresso (POST ... rfid=RFID0001&totem=INGR0001).
* Il server risponde 201 Created.
* Mario esce correttamente (POST ... rfid=RFID0001&totem=USCI0001 -> 201 Created).
* *Alternativa:* Se Mario avesse timbrato l’uscita (USCI0001) e si fosse accorto *subito* dell’errore (doveva restare), avrebbe potuto chiedere all’ufficio HR di cancellare l’ultima timbratura (DELETE ... ?rfid=RFID0001 -> 200 OK), riportando il suo stato a “entrato”.

## 2.4. Gestione degli Errori e Codici di Stato

È fondamentale che i client interpretino correttamente i codici di stato HTTP:

**200 OK:** Richiesta (GET, DELETE) completata con successo.

**201 Created:** Risorsa (timbratura) creata con successo (POST).

**400 Bad Request:** Richiesta malformata (mancano parametri, codici non validi). Il client non dovrebbe ritentare senza modificare la richiesta.

**409 Conflict:** La richiesta è valida ma viola una regola di business (es. doppio ingresso). Il client deve capire la logica e non ritentare ciecamente. Il corpo della risposta (text/plain) contiene dettagli.

**500 Internal Server Error:** Errore lato server (es. DB non raggiungibile). Il client può ritentare la richiesta dopo un intervallo.

I client dovrebbero sempre leggere il corpo della risposta (text/plain) in caso di errore per ottenere un messaggio descrittivo.

## 2.5. Esempi di Interazione (curl)

curl -X GET http://localhost:8080/TimbraturaWebService/api/timbrature

curl -X POST -d "rfid=RFID0001&totem=INGR0001" http://localhost:8080/TimbraturaWebService/api/timbrature

curl -X DELETE http://localhost:8080/TimbraturaWebService/api/timbrature?rfid=RFID0001

# 3. Dettagli di Implementazione

## 3.1. Architettura Software

Il sistema segue un’architettura a layer (strati) semplificata:

* **Presentation/Client Layer:** L’interfaccia web (index.html) o sistemi esterni.
* **Web/Controller Layer:** TimbraturaServlet.
* **Data Access Layer:** TimbraturaDAO.
* **Data Layer:** Database MySQL.

Il tutto è eseguito da **Apache Tomcat**.

## 3.2. Struttura del Database

* **lavoratori:** Anagrafica base, valida gli RFID.
* **tornelli:** Anagrafica lettori, definisce ingresso/uscita.
* **timbrature:** Tabella transazionale degli eventi.

## 3.3. Componenti Chiave del Backend (Java)

### 3.3.1. TimbraturaServlet (Controller/Endpoint)

* Gestisce le richieste HTTP (doGet, doPost, doDelete).
* Interpreta parametri e formatta risposte.
* Delega al TimbraturaDAO.

### 3.3.2. TimbraturaDAO (Data Access & Business Logic)

* Interagisce con il DB tramite JDBC (PreparedStatement).
* Implementa la logica di validazione ingresso/uscita.

### 3.3.3. DatabaseManager (Connessione DB)

* Fornisce connessioni JDBC.
* Centralizza i dettagli di connessione.

### 3.3.4. Modelli Dati

* Timbratura.java: Gestisce i dati.

## 3.4. Logica di Business e Validazione

La validazione in TimbraturaDAO.registraAccesso() controlla:

1. Esistenza RFID.
2. Esistenza Totem.
3. Ultima postazione RFID.
4. Coerenza della sequenza (In->Out, Out->In).

## 3.5. Considerazioni sulla Sicurezza (Attuale e Futura)

* **Attuale:** **Nessuna sicurezza implementata.**
* **Futura:** È **essenziale** aggiungere HTTPS, Autenticazione (API Key/OAuth) e Autorizzazione.

## 3.6. Possibili Estensioni Future

* Reportistica.
* Gestione anagrafica via API.
* Timbrature causali.
* Interfaccia web avanzata.
* Notifiche.
* Sicurezza.

# Appendice codice SQL:

CREATE DATABASE IF NOT EXISTS timbratura\_azienda;

USE timbratura\_azienda;

CREATE TABLE lavoratori (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    nome VARCHAR(50),

    cognome VARCHAR(50),

    mansione ENUM('dipendente', 'magazziniere', 'dirigente'),

    codice\_rfid CHAR(8) UNIQUE,

    timbratura\_odierna BOOLEAN DEFAULT FALSE

);

CREATE TABLE tornelli (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    codice\_totem CHAR(8) UNIQUE,

    postazione ENUM('ingresso', 'uscita')

);

CREATE TABLE timbrature (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    codice\_rfid\_utente CHAR(8),

    codice\_totem CHAR(8),

    giorno DATE,

    ora TIME,

    FOREIGN KEY (codice\_rfid\_utente) REFERENCES lavoratori(codice\_rfid),

    FOREIGN KEY (codice\_totem) REFERENCES tornelli(codice\_totem)

);