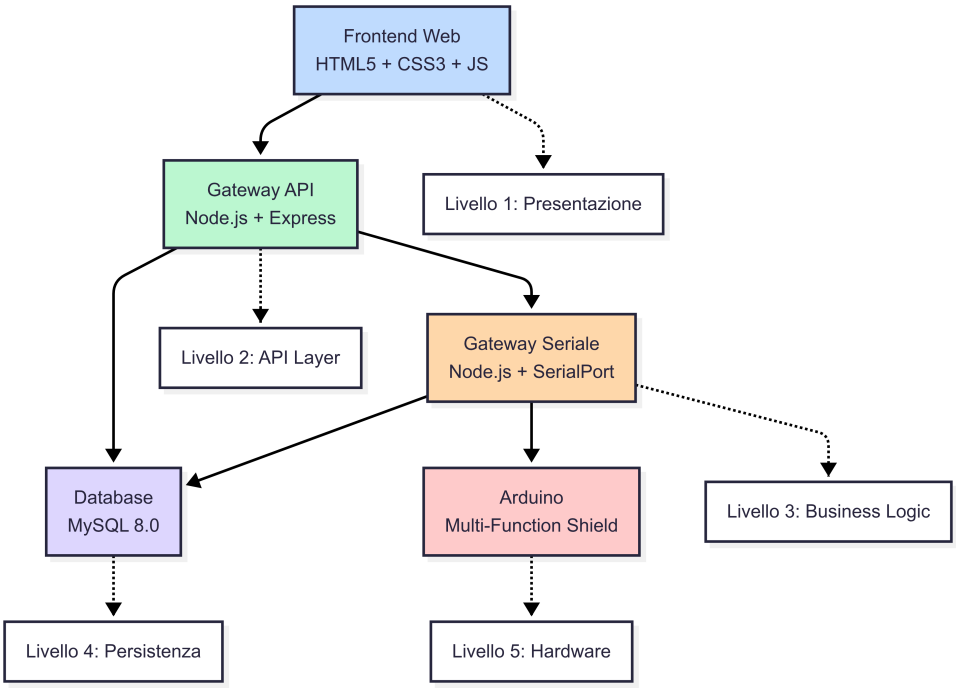
SISTEMA GESTIONE LAVORAZIONI TEMPORIZZATE

Progettazione Architetturale Completa

Compito 1 - Prova d'Esame Finale



1 Introduzione

Il presente documento descrive l'architettura completa del **Sistema Gestione Lavorazioni Temporizzate**, sviluppato come soluzione alla prova d'esame finale del corso TECNICO SUPERIORE DIGITAL SOLUTIONS 4.0.

Il sistema implementa una soluzione distribuita per la gestione temporizzata delle fasi produttive, permettendo la parametrizzazione remota delle lavorazioni e il controllo locale tramite dispositivi Arduino.

1.1 Obiettivi del Sistema

- **Parametrizzazione remota**: Definizione delle caratteristiche di lavorazione da interfaccia web
- **Controllo locale**: Gestione operativa tramite Arduino Multi-Function Shield
- **Monitoraggio real-time**: Visualizzazione stato
- **Storicizzazione**: Registrazione completa di tutte le operazioni
- **Scalabilità**: Architettura modulare per future estensioni

2 Architettura del Sistema

2.1 Panoramica Architetturale

Il sistema adotta un'architettura distribuita a **5 livelli** che garantisce separazione delle responsabilità, scalabilità e manutenibilità:

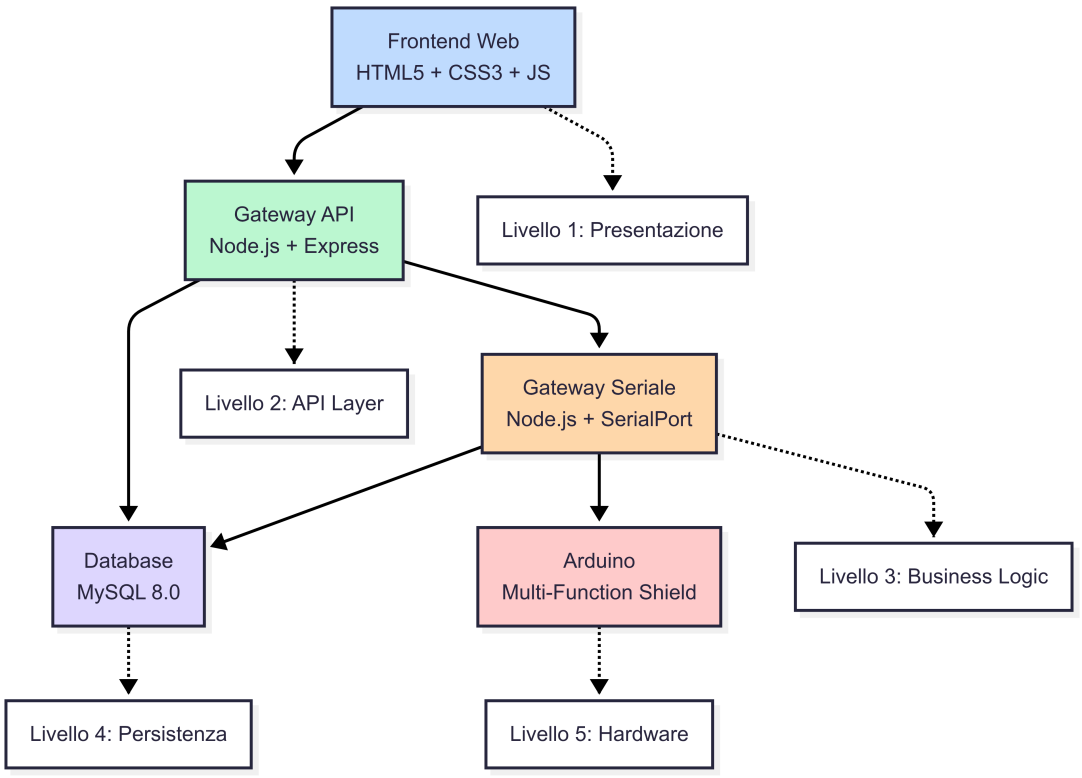


Figura 1: Architettura a 5 Livelli del Sistema

2.2 Principi Architetturali

Separation of Concerns Ogni componente ha responsabilità specifiche e ben definite

Loose Coupling I moduli comunicano attraverso interfacce standardizzate

High Cohesion Funzionalità correlate sono raggruppate nello stesso modulo

Scalabilità Architettura modulare che permette estensioni future

Fault Tolerance Gestione robusta di errori e disconnessioni

3 Componenti del Sistema

3.1 1. Frontend Web (Livello Presentazione)

Tecnologie: HTML5, CSS3 (Tailwind), JavaScript ES6+

Responsabilità:

- Interfaccia utente per gestione lavorazioni
- Form validati per inserimento parametri
- Visualizzazione real-time stato sistema
- Tabelle dinamiche per storico e monitoraggio
- Notifiche toast e feedback UX

Funzionalità Implementate:

✓ Form inserimento lavorazioni con validazione

✓ Status indicators per Arduino e Database

✓ Auto-refresh ogni 5-30 secondi

✓ Interfaccia responsive e moderna

3.2 2. Gateway API (Livello API)

Tecnologie: Node.js, Express.js, CORS

Responsabilità:

- Esposizione API REST per frontend
- Validazione input e gestione errori
- Interfacciamento con database MySQL
- Gestione stati lavorazione
- Monitoring stato Arduino via file condiviso

Endpoints Implementati:

Metodo	Endpoint	Descrizione	Payload
POST	/api/lavorazioni	Crea nuova lavorazione	identificativo, nome, durata
GET	/api/lavorazioni	Lista tutte le lavorazioni	-
POST	/api/lavorazioni/:id/invia	Mette in coda per Arduino	-
GET	/api/log	Storico lavorazioni	-
GET	/api/status	Stato sistema completo	-
GET	/api/progress	Progress lavorazione attiva	-

Tabella 1: API Endpoints del Gateway

3.3 3. Gateway Seriale (Livello Business Logic)

Tecnologie: Node.js, SerialPort, Auto-discovery

Responsabilità:

- Comunicazione seriale con Arduino
- Polling database ogni secondo
- Gestione stati lavorazione
- Logging eventi nel database
- Auto-discovery e riconnessione Arduino

Algoritmo di Auto-Discovery Arduino:

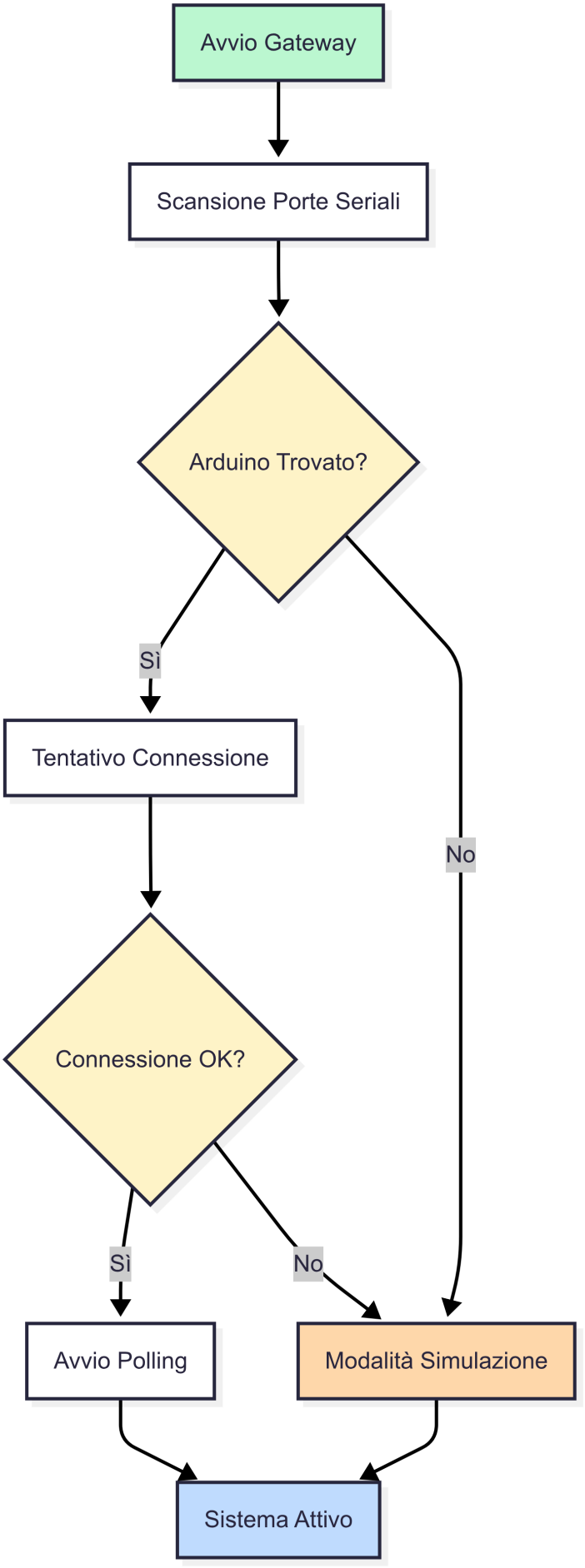


Figura 2: Algoritmo Autodiscovery Arduino

3.4 4. Database MySQL (Livello Persistenza)

Tecnologie: MySQL 8.0, Prepared Statements

Schema Database:

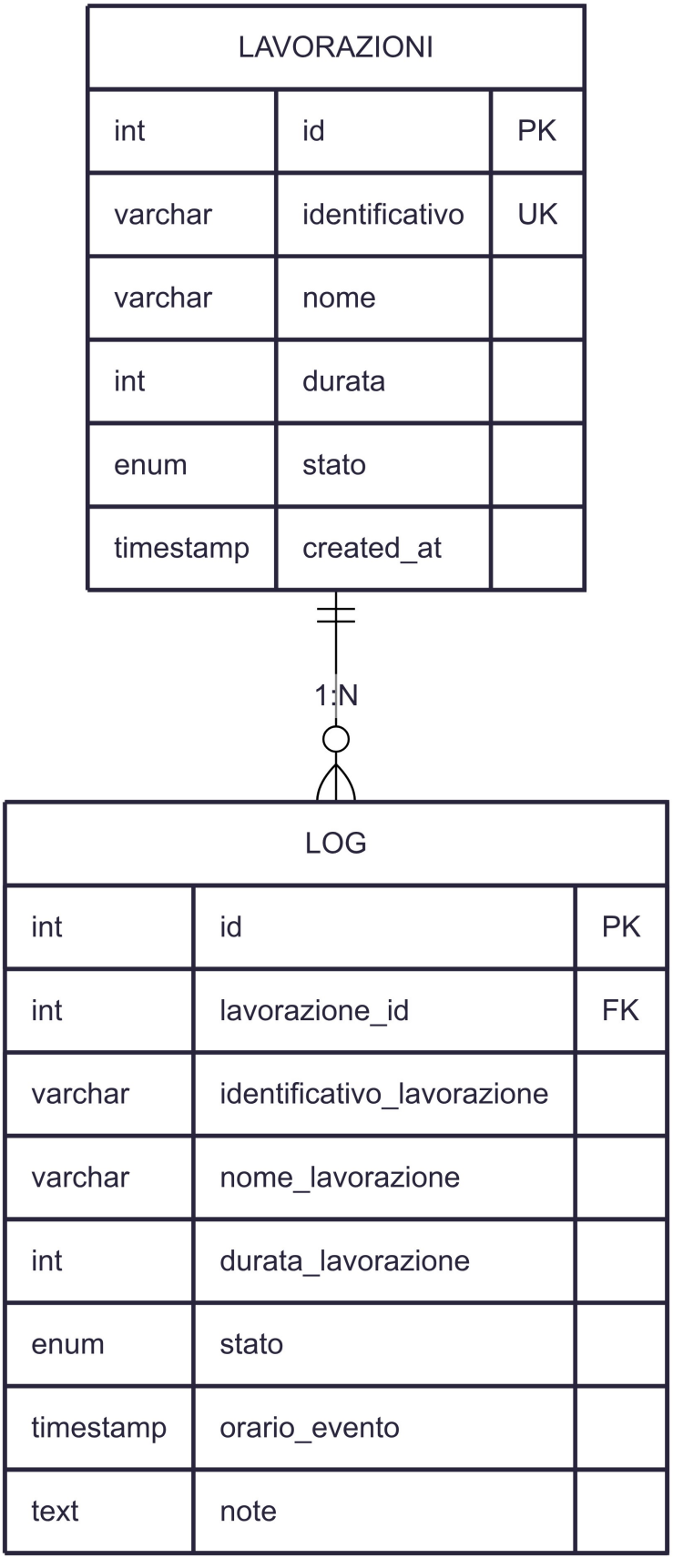


Figura 3: Schema Tabelle Database

Stati Lavorazione:

- **CONFIGURATA**: Lavorazione creata da frontend
- **IN_CODA**: In attesa di invio ad Arduino
- **INVIATA**: Inviata ad Arduino, in attesa accettazione
- **ACCETTATA**: Accettata dall'operatore (Pulsante 1)
- **AVVIATA**: Countdown iniziato (Pulsante 2)
- **COMPLETATA**: Lavorazione terminata con successo
- **RIFIUTATA**: Rifiutata dall'operatore (Pulsante 3)
- **CANCELLATA**: Cancellata durante esecuzione

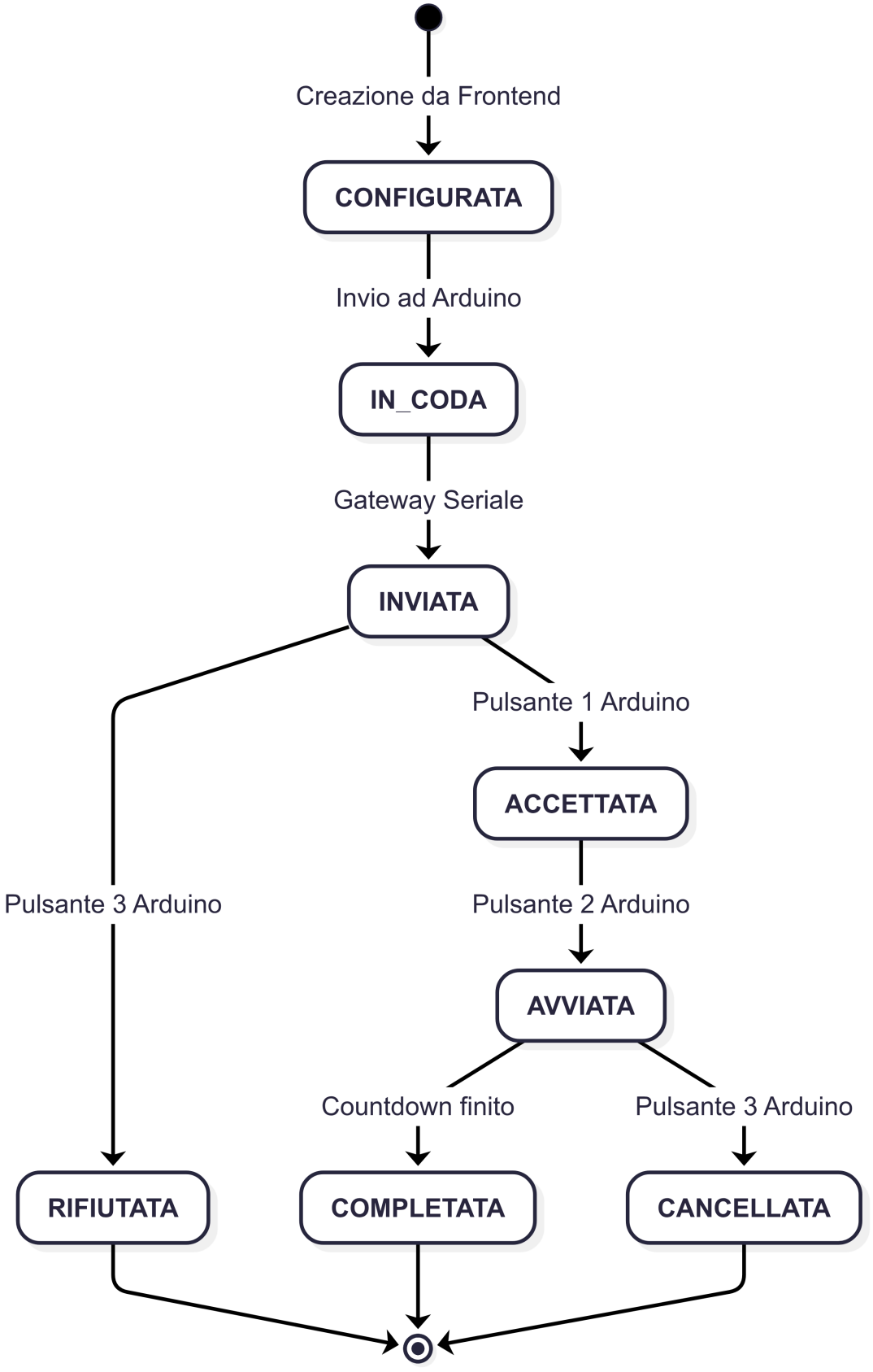


Figura 4: Diagramma Stati Lavorazione

3.5 5. Arduino Multi-Function Shield (Livello Hardware)

Tecnologie: C++, TimerOne, MultiFuncShield, ArduinoJson

Hardware Utilizzato:

- Display 7-segmenti a 4 cifre
- 4 LED programmabili (Rosso, Verde, Blu, Giallo)
- 3 Pulsanti (S1, S2, S3)
- Buzzer integrato

Mapping Hardware secondo Specifiche Esame:

Componente	Funzione	Stato Associato
LED 1	Lavorazione in coda	LAVORAZIONE_IN_CODA
LED 2	Lavorazione in corso	COUNTDOWN_ATTIVO
LED 3	Rifiuto/Cancellazione	LAVORAZIONE_RIFIUTATA
LED 4	Completamento	LAVORAZIONE_COMPLETATA
Pulsante 1	Accetta lavorazione	-
Pulsante 2	Avvia lavorazione	-
Pulsante 3	Rifiuta/Cancella	-
Display	Countdown/Nome/Stato	-
Buzzer	Segnale completamento	-

Tabella 2: Mapping Hardware Arduino

4 Flussi Operativi

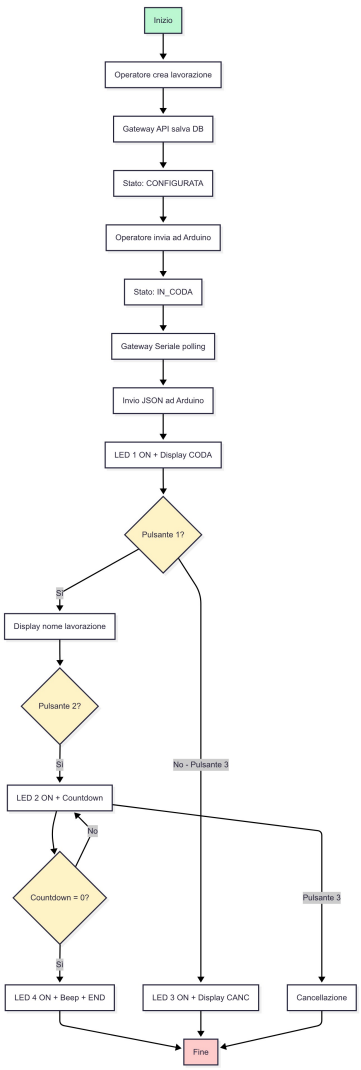


Figura 5: Flusso Completo Lavorazione

4.2 Diagramma di Sequenza - Interazioni Componenti

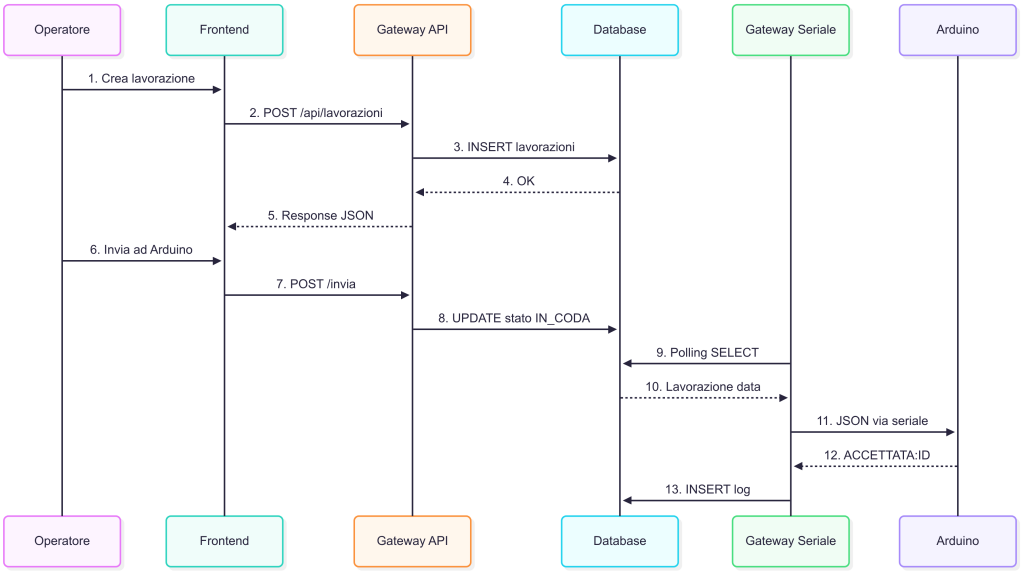


Figura 6: Diagramma di Sequenza - Interazioni tra Componenti

5 Logica Distribuita

5.1 Distribuzione delle Responsabilità

Il sistema implementa una **logica distribuita** che garantisce:

Frontend Web Si occupa esclusivamente della presentazione e dell'interazione utente, delegando tutta la business logic ai gateway

Gateway API Gestisce la logica di validazione, persistenza e stato delle lavorazioni, esponendo interfacce REST standardizzate

Gateway Seriale Implementa la logica di comunicazione hardware, polling database e sincronizzazione stati tra sistema e Arduino

Database Centralizza la persistenza e garantisce consistenza dei dati attraverso vincoli di integrità

Arduino Gestisce la logica operativa locale, interfaccia utente hardware e feedback real-time

5.2 Strategie di Comunicazione

Componenti	Protocollo	Formato	Frequenza
Frontend Gateway API	HTTP/REST	JSON	On-demand + 30s
Gateway API Database	TCP/SQL	SQL Queries	On-demand
Gateway Seriale Database	TCP/SQL	SQL Queries	1 secondo
Gateway Seriale Arduino	RS232/USB	JSON + Newline	Real-time
Gateway API Gateway Seriale	File System	JSON Status	5 secondi

Tabella 3: Strategie di Comunicazione tra Componenti

5.3 Gestione dello Stato Distribuito

Il sistema mantiene la coerenza dello stato attraverso:

- **Database come Single Source of Truth**: Tutte le transizioni di stato vengono persistite
- **Event Sourcing**: Ogni cambio di stato genera un evento nel log
- **Polling Synchronization**: Il Gateway Seriale sincronizza gli stati ogni secondo
- **Status File**: Condivisione stato Arduino tra gateway tramite file JSON
- **Idempotenza**: Operazioni ripetibili senza effetti collaterali

6 Interazioni con il Database

6.1 Modello Dati Relazionale

Il database implementa un modello relazionale ottimizzato per:

Integrità Referenziale Vincoli di foreign key garantiscono coerenza

Performance Indici ottimizzati su campi di ricerca frequente

Auditability Tracciamento completo di ogni operazione

Scalabilità Schema estendibile per future funzionalità

6.2 Pattern di Accesso ai Dati

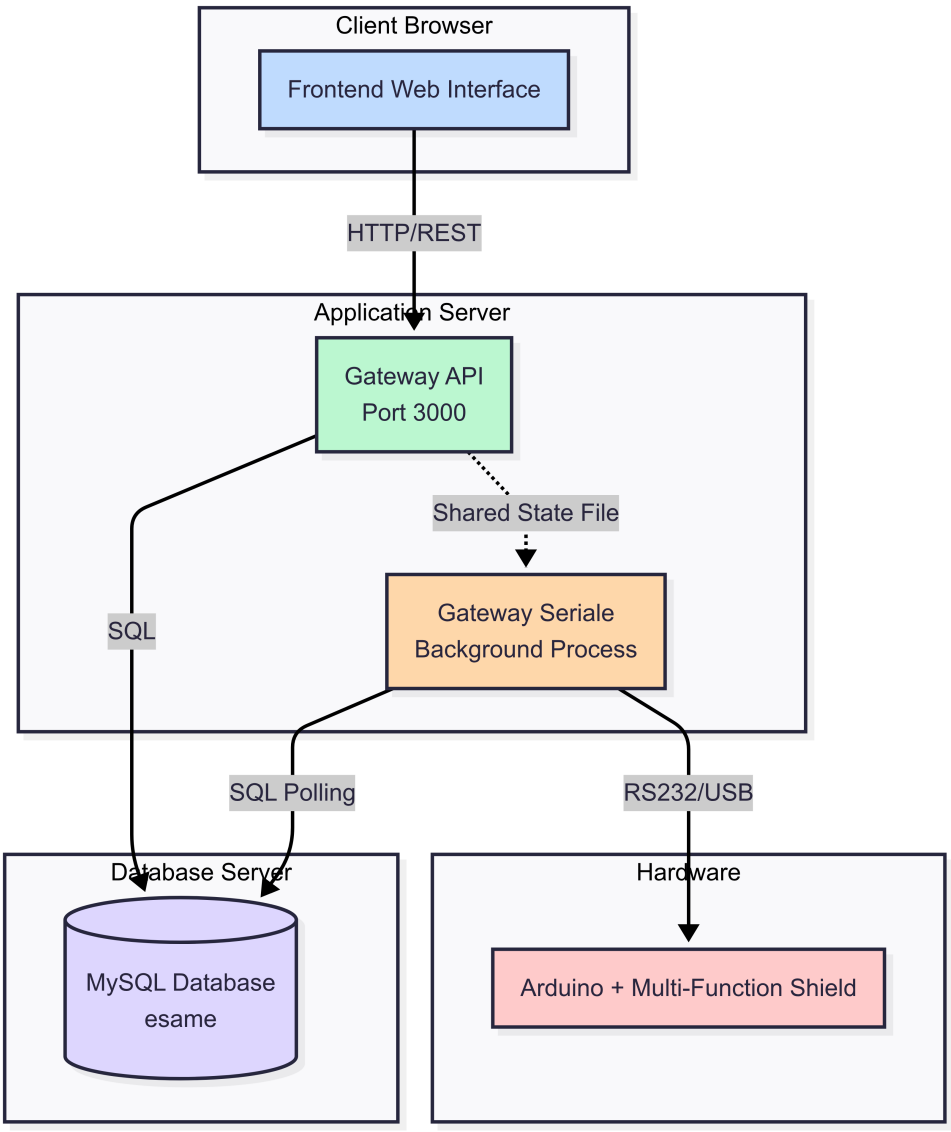


Figura 7: Pattern di Accesso ai Dati e Deployment

Query Principali Implementate:

1. **Inserimento Lavorazione** (Gateway API)

```
INSERT INTO lavorazioni (identificativo, nome, durata, stato)
VALUES (?, ?, ?, 'CONFIGURATA')
```

2. **Polling Lavorazioni in Coda** (Gateway Seriale)

```
SELECT * FROM lavorazioni
WHERE stato = 'IN_CODA'
ORDER BY created_at ASC LIMIT 1
```

3. **Logging Eventi** (Gateway Seriale)

```
INSERT INTO log (lavorazione_id, identificativo_lavorazione,
                 nome_lavorazione, durata_lavorazione, stato, note)
VALUES (?, ?, ?, ?, ?, ?)
```

4. **Status Sistema** (Gateway API)

```sql
SELECT stato, COUNT(*) as count
FROM lavorazioni GROUP BY stato
```

6.3 Gestione Transazioni e Consistenza

- **ACID Compliance**: Tutte le operazioni rispettano proprietà ACID
- **Isolation Level**: READ COMMITTED per bilanciare performance e consistenza
- **Connection Pooling**: Gestione efficiente delle connessioni database
- **Prepared Statements**: Prevenzione SQL injection e ottimizzazione performance

7 Specifiche Tecniche Implementative

7.1 Tecnologie e Dipendenze

Componente	Tecnologia	Versione
Frontend	HTML5 + CSS3	Standard
	Tailwind CSS	CDN Latest
	JavaScript ES6+	Vanilla
Gateway API	Node.js	$\geq$ 16.0.0
	Express.js	^4.18.2
	MySQL2	^3.6.0
	CORS	^2.8.5
Gateway Seriale	Node.js	$\geq$ 16.0.0
	SerialPort	^12.0.0
	Readline Parser	^12.0.0
Database	MySQL	$\geq$ 8.0
Arduino	Arduino IDE	$\geq$ 1.8.0
	TimerOne	Latest
	MultiFuncShield	Latest
	ArduinoJson	Latest

Tabella 4: Stack Tecnologico del Sistema

7.2 Configurazioni di Sicurezza

Input Validation Validazione rigorosa di tutti gli input utente

SQL Injection Prevention Uso esclusivo di prepared statements

CORS Policy Configurazione CORS appropriata per ambiente

Error Handling Gestione errori senza esposizione di informazioni sensibili

Connection Security Connessioni database con timeout e retry logic

8 Gestione Errori e Fault Tolerance

8.1 Strategie di Resilienza

Arduino Disconnection Auto-discovery e riconnessione automatica ogni 5 secondi

Database Timeout Retry automatico con backoff esponenziale

API Failures Graceful degradation con messaggi utente appropriati

Serial Communication Timeout e validazione formato messaggi

State Synchronization Recupero stato da database in caso di restart

8.2 Logging e Monitoring

- **Application Logs**: Log strutturati per debugging e monitoring
- **Database Audit**: Tracciamento completo di tutte le operazioni
- **Status Files**: Condivisione stato real-time tra componenti
- **Error Tracking**: Logging centralizzato degli errori
- **Performance Metrics**: Monitoring tempo di risposta e throughput

9 Workflow di Deploy e Manutenzione

9.1 Procedura di Installazione

1. **Prerequisiti**

```
- Node.js >= 16.0.0
- MySQL >= 8.0
- Arduino IDE >= 1.8.0
- Multi-Function Shield
```

2. **Setup Database**

```
npm run setup-db
```

3. **Installazione Dipendenze**

```
npm install
```

4. **Programmazione Arduino**

```
- Caricare arduino/lavorazioni-controller.ino
- Installare librerie richieste
```

5. **Avvio Sistema**

```
# Terminal 1
npm start

# Terminal 2
npm run gateway-serial
```

9.2 Monitoraggio Sistema

Il sistema implementa un monitoraggio completo a pi`u livelli:

- **Frontend Dashboard**: Status real-time di tutti i componenti
- **API Health Checks**: Endpoint dedicati per verifiche stato
- **Database Performance**: Monitoring query e connessioni
- **Arduino Communication**: Tracking connessione e messaggi
- **Error Logging**: Sistema centralizzato di gestione errori

10 Estensioni Future

10.1 Roadmap Evolutiva

Multi-Arduino Supporto per pi`u schede Arduino in parallelo

Web Dashboard Dashboard real-time con grafici e analytics

Mobile App Applicazione mobile per monitoring remoto

Cloud Integration Integrazione con servizi cloud per backup e analytics

IoT Expansion Integrazione con sensori ambientali e altri dispositivi IoT

10.2 Punti di Estensione Architetturale

- **Plugin Architecture**: Sistema di plugin per nuove funzionalit`a
- **Event Bus**: Message broker per comunicazione asincrona
- **Microservices**: Decomposizione in microservizi specializzati
- **Load Balancing**: Distribuzione carico per alta disponibilit`a
- **Containerization**: Docker per deployment semplificato

11 Conclusioni

Il **Sistema Gestione Lavorazioni Temporizzate** implementa un'architettura distribuita moderna e scalabile che soddisfa completamente i requisiti della prova d'esame.

11.1 Obiettivi Raggiunti

- ✓ **Architettura a 5 livelli** con separazione chiara delle responsabilita'
- ✓ **Comunicazione robusta** tra tutti i componenti del sistema
- ✓ **Gestione stati completa** con tracciamento degli eventi
- ✓ **Interfaccia utente moderna** e user-friendly
- ✓ **Auto-discovery Arduino** per facilit`a di setup
- ✓ **Fault tolerance** e gestione errori avanzata
- ✓ **Real-time monitoring** dello stato sistema
- ✓ **Documentazione completa** e professionale