# Analisi Tecnica e Funzionale Completa: Piattaforma Test

## 1. Introduzione

Questo documento descrive l'analisi funzionale e tecnica per una web application focalizzata sulla creazione e somministrazione di test che includono sia domande a scelta multipla che sfide di scrittura del codice. L'obiettivo è permettere agli utenti di accedere ai test tramite un link pubblico, sottomettere le risposte e visualizzare i risultati tramite un backoffice.

## 2. Requisiti Funzionali

### 2.1 Funzionalità di Svolgimento Test (Lato Utente)

- **Accesso al Test Tramite Link:** Un utente (non autenticato) può accedere a un test specifico tramite un URL unico. Al caricamento del link, viene creata una "sessione di test" anonima per l'utente.
- **Visualizzazione Domande:** Le domande vengono presentate all'utente una alla volta. L'interfaccia fornisce controlli di navigazione (es. 'Precedente', 'Successivo') per muoversi tra le domande prima della sottomissione finale.
- **Web IDE Integrato (per domande di codice):** Per le domande che richiedono la scrittura di codice, è disponibile un'area di testo (Web IDE) con evidenziazione della sintassi.
- **Sottomissione delle Risposte:** L'utente può sottomettere le proprie risposte per tutte le domande del test in un'unica azione finale. La sottomissione del codice implica l'invio della soluzione al backend per una valutazione completa e asincrona rispetto a tutti i test case. La piattaforma conferma l'avvenuta sottomissione.

### 2.2 Funzionalità di Backoffice (Lato Amministratore/Revisore)

- **Visualizzazione Sessioni di Test:** Accesso a una lista di tutte le sessioni di test completate.
- **Dettaglio Sessione di Test:** Visualizzazione dei dettagli di una sessione, incluse le risposte date e i risultati della valutazione (punteggio, test case superati/falliti visualizzati tramite grafici e barre di progresso).
- **Gestione Quiz e Domande:** Un'area dedicata per creare, modificare ed eliminare quiz e le relative domande (a scelta multipla o di codice).

### 2.3 Gestione dei Test a Tempo

- **Creazione Test a Tempo:** L'amministratore può impostare una durata (in minuti) per un intero quiz.

- **Svolgimento Test a Tempo:** All'avvio del test, un timer visibile viene mostrato all'utente. Alla scadenza del tempo, la sessione deve essere sottomessa automaticamente.

## 3. Architettura a Microservizi Proposta

L'architettura è basata su microservizi per garantire scalabilità, manutenibilità e sviluppo parallelo.

### 3.1 Elenco e Ruolo dei Microservizi

1. **Servizio Gestione Quiz:** Responsabile della CRUD di quiz, domande e test case.
2. **Servizio Sessioni di Test:** Gestisce il ciclo di vita delle sessioni, inclusa la registrazione delle risposte e la gestione dei timer.
3. **Servizio Esecuzione Codice (Code Runner):** Esegue il codice sottomesso in ambienti containerizzati isolati (Docker) per la valutazione.
4. **Servizio Valutazione:** Valuta le risposte sottomesse, calcola i punteggi e orchestra l'esecuzione del codice tramite il Code Runner.
5. **Servizio Backoffice & Reportistica:** Aggrega dati dagli altri servizi per fornirli all'interfaccia di backoffice.

## 4. Architettura di Comunicazione

### 4.1 Comunicazione Sincrona vs. Asincrona

- **Sincrona (API REST):** Usata per richieste che necessitano una risposta immediata (es. caricamento dati dal frontend). Semplice, ma accoppia i servizi.
- **Asincrona (Code di Messaggi/Eventi):** Usata per operazioni lunghe o che possono essere disaccoppiate (es. valutazione di un test). Il chiamante invia un messaggio e non attende la fine del processo, aumentando la resilienza del sistema.

### 4.2 Analisi e Raccomandazioni per Flusso

| Interazione | Tipo Consigliato | Motivazione |
|---|---|---|
| **Frontend → Tutti i Servizi** | **Sincrona** | L'utente si aspetta una risposta immediata dall'interfaccia. |
| **Servizio Sessioni di Test → Servizio Gestione Quiz** | **Sincrona** | Per avviare un test, la sua struttura è necessaria immediatamente. |
| **Servizio Sessioni di Test →** | **Asincrona** | **CASO CHIAVE:** La |

| | | |
|---|---|---|
| **Servizio Valutazione** | | valutazione completa può essere un processo lungo. L'utente non deve attendere il punteggio finale. È sufficiente ricevere una conferma immediata ("Test sottomesso con successo"). |
| **Servizio Valutazione → Servizio Esecuzione Codice** | **Sincrona** | Passo interno al processo di valutazione che richiede un risultato per procedere. |
| **Servizio Backoffice & Reportistica → Servizi vari** | **Sincrona** | L'amministratore richiede dati e si aspetta di visualizzarli subito. |

### 4.3 Flusso di Sottomissione Asincrono

1. **Frontend** invia le risposte al **Servizio Sessioni di Test**.
2. Il **Servizio Sessioni** salva le risposte, marca la sessione come "Completata" e pubblica un messaggio ValutazioneRichiesta su una coda (es. RabbitMQ).
3. Il **Servizio Sessioni** risponde **immediatamente** al Frontend con 202 Accepted.
4. Indipendentemente, il **Servizio Valutazione** preleva il messaggio ed esegue la valutazione.

## 5. Analisi Tecnica: Gestione dei Test a Tempo

L'introduzione di un timer impone requisiti di coerenza tra frontend e backend per garantire correttezza e prevenire manipolazioni.

### 5.1 Ruoli di Frontend e Backend

- **Frontend (Client-side):** È responsabile della **visualizzazione del countdown**. Gestisce l'UI per mostrare il tempo rimanente all'utente. Questo offre la migliore esperienza utente, senza richiedere una connessione costante al server.
- **Backend (Server-side):** È l'**autorità (source of truth)** sulla validità della sessione. Il server deve gestire la logica di business per determinare se una sottomissione è avvenuta in tempo.

### 5.2 Impatto sulla Comunicazione: REST è Sufficiente

Per questa funzionalità, non è necessario utilizzare WebSocket. Un'architettura basata su API REST è robusta e adeguata.
Il flusso di interazione sarebbe:

1. **Avvio del Test:**

    - Il frontend chiama un endpoint: POST /api/sessions/start
    - Il **Servizio Sessioni di Test** riceve la richiesta, registra il timestamp di inizio (startTime) nel suo database e calcola la scadenza esatta (endTime = startTime + duration).
    - Il servizio risponde al frontend con l'oggetto della sessione, che include la durata totale e il timestamp di scadenza ufficiale (endTime).
2. **Durante il Test:**
    - Il frontend avvia un suo timer interno basandosi sulla durata ricevuta. L'utente risponde alle domande e naviga. Non sono necessarie ulteriori comunicazioni con il backend.
3. **Scadenza del Tempo o Sottomissione Manuale:**
    - Allo scadere del timer (o se l'utente clicca "Sottometti"), il frontend invia tutte le risposte a: POST /api/sessions/{id}/submit.
    - Il **Servizio Sessioni di Test** riceve la sottomissione. Prima di accettarla, **verifica il timestamp corrente del server rispetto all' endTime** salvato per quella sessione.
        - Se la sottomissione è in tempo, viene accettata e inoltrata al processo di valutazione asincrono.
        - Se la sottomissione è in ritardo, viene rifiutata con un errore 403 Forbidden o 400 Bad Request.

**5.3 Perché non un WebSocket?**

Un WebSocket mantiene una connessione persistente tra client e server, ideale per comunicazioni *server-push* in tempo reale. Sarebbe utile in scenari come:

- Un amministratore che mette in pausa il test per tutti i candidati in tempo reale.
- Una chat tra candidato e supervisore.
- Aggiornamenti continui inviati dal server.

Per un semplice timer, dove l'unica logica critica è la validazione dell'istante di sottomissione, un WebSocket rappresenta una complessità architetturale ingiustificata. L'approccio REST garantisce la stessa affidabilità con una soluzione più semplice e standard.

## 6. Requisiti Non Funzionali

- **Sicurezza:** Esecuzione del codice in sandbox isolati con limiti stringenti su CPU, RAM e tempo.
- **Scalabilità:** Il Servizio Esecuzione Codice deve essere progettato per la scalabilità orizzontale.
- **Performance:** Risposte rapide dalle API sincrone (< 500ms) e caricamento rapido

del backoffice.

## 7. Conclusioni

Questo documento definisce un'architettura robusta, resiliente e scalabile. La chiara distinzione tra comunicazione sincrona e asincrona, unita a una gestione sicura dei test a tempo tramite API REST, fornisce una solida base per lo sviluppo della piattaforma.