Università degli Studi di Salerno
Corso di Ingegneria del Software

SwaGGed
System Design Document
Versione 2.0

Data: 29/12/2024

Coordinatore del progetto

Nome	Matricola

Partecipanti:

Nome	Matricola
Choaib Goumri	0512118390
Mattia Gallucci	0512116893

Scritto da:	Choaib Goumri
	Mattia Gallucci

Revision History

Data	Versione	Descrizione	Autore
21/11/2024	0.1	Inzio System Design Document, aggiunta di introduzione e Architettura sistema corrente.	Choaib Goumri, Gallucci Mattia
22/11/2024	1.0	Architettura sistema proposto e servizi dei sottosistemi.	Choaib Goumri, Gallucci Mattia
04/12/2024	1.1	Aggiustamenti al component diagram	Choaib Goumri, Gallucci Mattia
29/12/2024	2.0	Finalizzazione documento	Choaib Goumri, Gallucci Mattia

Sommario

1. INTRODUZIONE

1.1. Scopo del sistema

Lo scopo del sistema è soddisfare le esigenze degli appassionati di musica, creando un social network interamente dedicato alla scoperta, condivisione e ascolto di musica. In un'epoca in cui la musica riveste un ruolo centrale nella vita quotidiana e le piattaforme digitali stanno evolvendo rapidamente, SwaGGed offre un'esperienza unica che combina le funzionalità social con l'esplorazione musicale. La piattaforma consente agli utenti non solo di visualizzare le ultime novità musicali del momento, ma anche di connettersi con altri appassionati, scoprire nuovi artisti e tendenze, e partecipare attivamente a una community musicale.

1.2. Design Goals

Durante la fase di System Design abbiamo deciso di dare maggiore priorità ai seguenti obiettivi di progettazione

- Robustezza

L'obiettivo è garantire che il sistema sia in grado di gestire con robustezza sia gli input non validi forniti dall'utente sia le situazioni limite a livello applicativo. In caso di input errati, il sistema deve rispondere con messaggi di errore appropriati e informativi, mentre in presenza di condizioni non favorevoli o non convenzionali deve essere capace di identificare e adottare soluzioni alternative adeguate. Questo approccio amplifica il concetto di *reliability* richiesto dal cliente per gli input dell'utente, estendendolo a una gestione resiliente dell'intero sistema.

- **Usabilità**

L'obiettivo è progettare un'interfaccia utente adeguata per i diversi tipi di attori previsti dal sistema, garantendo un utilizzo delle funzionalità semplice e intuitivo. Questo approccio mira a facilitare l'interazione con il sistema e a ridurre al minimo il tempo medio necessario agli utenti per acquisire familiarità con l'applicazione. Tale obiettivo conferma e rafforza il requisito di *usability* evidenziato dal cliente nel *problem statement*.

- **Manutenibilità**

L'obiettivo è favorire l'evoluzione dell'architettura software in eventuali future iterazioni di sviluppo, semplificando al contempo le modifiche da parte degli sviluppatori sulle sue componenti. Per raggiungere questo scopo, il software sarà progettato come un insieme di sottosistemi altamente coesi e debolmente accoppiati, organizzati in strati che forniscono funzionalità allo stesso livello di astrazione. La coesione all'interno dei sottosistemi sarà ottimizzata includendo esclusivamente classi che condividono scopi e responsabilità comuni. Questo approccio amplia il requisito non funzionale di *supportability*, in particolare per quanto riguarda la stratificazione dell'applicazione su tre livelli di responsabilità.

1.3. Riferimenti

- *Libro: Object-oriented-Software-Engineering-3rd-Edition*
- *Documento: Requirement Analysis Document*

1.4. Panoramica

Il presente documento di *System Design* (SDD) fornisce i dettagli tecnici relativi al design del sistema SwaGGed. Informazioni aggiuntive sulle funzionalità e caratteristiche del sistema sono disponibili nel documento di Analisi dei Requisiti (RAD), mentre una panoramica generale è contenuta nel *Problem Statement*.

In questo documento sono presentati:

- Un'introduzione generale all'architettura e agli obiettivi di design del sistema;
- Una suddivisione del sistema in sottosistemi, con il relativo mapping Hardware/Software, che associa ciascun sottosistema a un componente hardware specifico;
- La descrizione dei meccanismi di controllo dell'accesso e dei problemi di sicurezza correlati;
- Il controllo generale del software e dei *boundary*, con particolare attenzione agli stati iniziali e alla gestione complessiva del sistema.

2. ARCHITETTURA DEL SISTEMA CORRENTE

Data l'assenza di un sistema esistente, il nostro sistema si ispira al social network chiamato **Reddit.**

Il sistema **SwaGGed** e il social network **Reddit** condividono alcune caratteristiche fondamentali, come la creazione di una community attiva e la centralità degli interessi condivisi. Tuttavia, presentano differenze significative sia in termini di struttura che di valore offerto agli utenti.

Reddit: il modello della community basata sui subreddit

- Reddit è un social network generico strutturato in **subreddit tematici**, ognuno dedicato a un argomento specifico. Gli utenti possono iscriversi a subreddit di interesse (ad esempio, r/musica o r/hiphopheads) e partecipare a discussioni sotto forma di thread.
- La scoperta di contenuti è **guidata dagli utenti** attraverso upvote e downvote, che determinano la visibilità dei post.
- L'esperienza è **multipiattaforma**, ma la musica è solo una delle tante categorie trattate. Reddit non è ottimizzato per l'ascolto o la condivisione musicale diretta.

SwaGGed: un social per appassionati di musica

A differenza di Reddit, **SwaGGed** si distingue per il suo approccio verticale e integrato, progettato specificamente per gli appassionati di musica.

1. **Focus musicale totale**:

   Mentre Reddit include la musica come un tema tra tanti, SwaGGed è interamente dedicato alla musica.

   - Accesso alle ultime novità e tendenze musicali in tempo reale.

2. **Esplorazione musicale avanzata**:

   A differenza di Reddit, SwaGGed mette in primo piano la **scoperta musicale integrata**. Gli utenti possono:

   - Scoprire nuovi artisti tramite le community
   - Partecipare attivamente alla promozione di artisti indipendenti, creando un **rapporto diretto tra artisti e fan**.

3. **Community più coesa e tematica**:

   Reddit ospita un'ampia gamma di utenti con interessi diversificati, spesso frammentati. SwaGGed, invece, costruisce una **community fortemente coesa**, legata da una passione comune: la musica. Questo approccio favorisce:

   - Un senso di appartenenza più marcato.
   - Maggiore coinvolgimento e partecipazione attiva.

Valore aggiunto di SwaGGed rispetto a Reddit

- **Personalizzazione dell'esperienza**: grazie all'attenzione esclusiva alla musica, SwaGGed offre un'esperienza profondamente mirata e personalizzata che Reddit, come piattaforma generica, non è in grado di eguagliare.
- **Supporto alla scena musicale indipendente**: SwaGGed non si limita a ospitare contenuti, ma diventa una piattaforma attiva per promuovere artisti emergenti, attraverso meccanismi social e partecipativi.

3. ARCHITETTURA DEL SISTEMA PROPOSTO

3.1 Panoramica

L'architettura del sistema proposto è organizzata in tre strati distinti, ciascuno dei quali contribuisce in modo specifico all'erogazione delle funzionalità del sistema. Si tratta di un'**architettura chiusa** (Closed Architecture), strutturata in modo gerarchico, con i seguenti livelli:

1. **Presentation Layer (Strato di Presentazione):**

   Questo livello si trova al vertice dell'architettura e ha il compito principale di gestire la visualizzazione delle pagine richieste dai client web. Si occupa di fornire un'interfaccia utente intuitiva e accessibile, rappresentando i dati in modo chiaro e navigabile.

2. **Application Layer (Strato di Controllo o di Business):**

   Situato tra il Presentation Layer e lo Storage Layer, questo strato funge da mediatore. È' responsabile della gestione delle richieste provenienti dai client web e dell'implementazione delle logiche di controllo del sistema. Inoltre, fornisce i dati richiesti alle view, che si occupano della loro rappresentazione, garantendo che le operazioni siano eseguite in modo corretto e coerente con le regole di business.

3. **Storage Layer (Strato di Persistenza):**

   L'ultimo livello è incaricato della gestione dei dati persistenti del sistema. Si occupa di aggiornare le informazioni, garantirne la consistenza durante il recupero e assicurare che i dati siano sempre sincronizzati e accessibili in maniera affidabile.

3.2 Decomposizione in sottosistemi

Procediamo con la decomposizione del sistema in sottosistemi, raggruppando le funzionalità in base al loro contesto di appartenenza.

Presentation layer

Il **Presentation Layer** si occupa della visualizzazione dell'interfaccia utente con i seguenti moduli:

- **Autenticazione**, **Registrazione**, **Homepage**, **Utente**, **Community**, e **Commenti**, ciascuno responsabile della visualizzazione e interazione con specifiche funzionalità del sistema.

Application Layer

Nell' **Application Layer**, i sottosistemi gestiscono le operazioni principali del sistema e implementano le logiche necessarie per le funzionalità offerte dalla piattaforma. I principali sottosistemi sono i seguenti:

- **Gestione Post**

  Gestisce tutte le operazioni relative ai post degli utenti, come creazione, modifica eliminazione, apprezzamento, salvataggio e segnalazione.

- **Gestione Commenti**

  Si occupa della gestione dei commenti associati ai post, permettendo agli utenti di aggiungere, rimuovere, apprezzare e segnalare commenti

- **Gestione Community**

  Consente la gestione delle community, inclusa la creazione, cancellazione, modifica nome e segnalazione di una community.

- **Gestione Utenti**

  Gestisce le informazioni relative agli utenti, inclusi i profili, e i ruoli (utente registrato e moderatore).

  Si occupa di autenticare gli utenti durante il login, creare sessioni sicure e gestire il logout. Collabora con il database per verificare le credenziali e proteggere le sessioni.

  Gestisce la registrazione degli utenti, verificando la validità e unicità delle informazioni fornite, come e-mail e username. Si occupa anche della crittografia delle password per garantire la sicurezza.

Storage Layer

Il livello di persistenza del sistema si basa su due componenti principali: il **DBMS** e il **FileSystem**, utilizzati per garantire la conservazione e l'accesso ai dati in modo efficace.

- I sottosistemi dell'Application **Layer, Gestione Autenticazione**, **Gestione Registrazione**, **Gestione Post**, **Gestione Community**, **Gestione Commenti** e **Gestione Utenti** interagiscono sia con il DBMS che con il FileSystem. Questi sottosistemi gestiscono dati strutturati, archiviati nel database (ad esempio, profili utenti, informazioni sui post e dettagli delle community), e contenuti di grandi dimensioni o non strutturati, come immagini, file e altri media, che vengono salvati nel FileSystem.

Questo approccio permette una gestione ottimizzata delle risorse di archiviazione, con il DBMS che garantisce operazioni rapide e sicure sui dati strutturati, e il FileSystem che offre uno spazio dedicato per contenuti di grandi dimensioni. Entrambi i componenti lavorano in sinergia per assicurare la consistenza, l'integrità e l'efficienza nella gestione dei dati della piattaforma.

3.3 Hardware/software Mapping

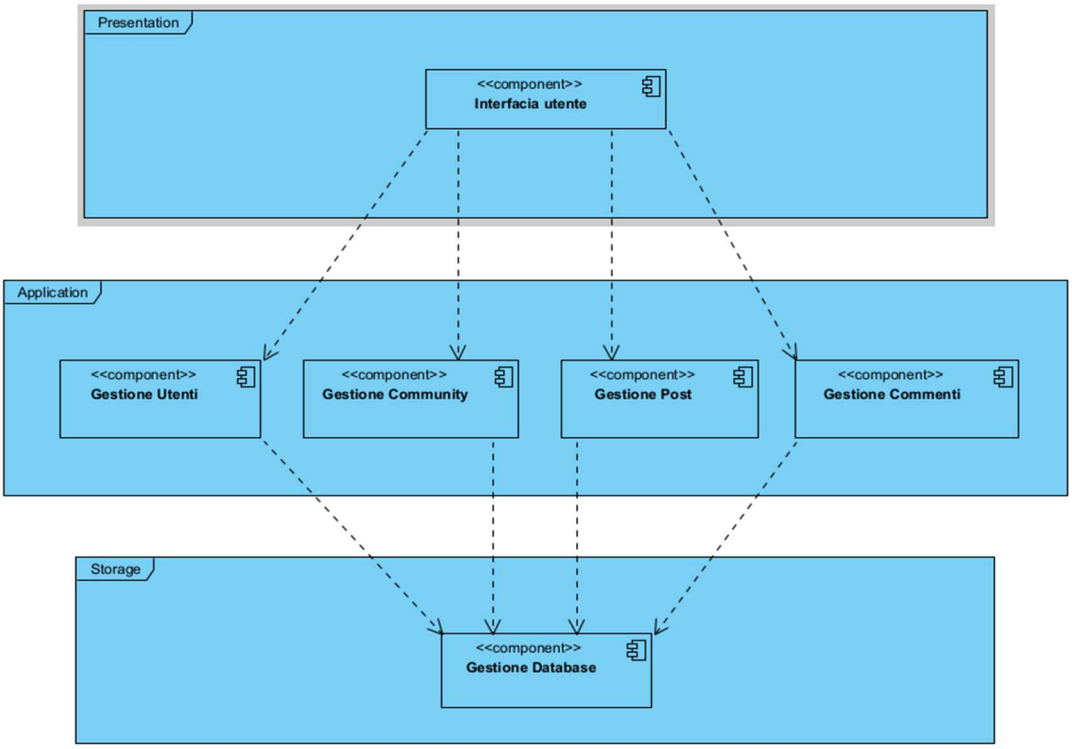
Il sistema adotta un'architettura **Client/Server**, in cui il **Web Server** è rappresentato da **Apache Tomcat 9**, ospitato su una singola macchina. La logica applicativa del sistema è implementata tramite **Java Servlet**, mentre l'interfaccia utente è sviluppata utilizzando **pagine JSP (Java Servlet Page)**.
Il **Client** è costituito dal **Web Browser** utilizzato dall'utente per interagire con il sistema.

La comunicazione tra i componenti avviene tramite:

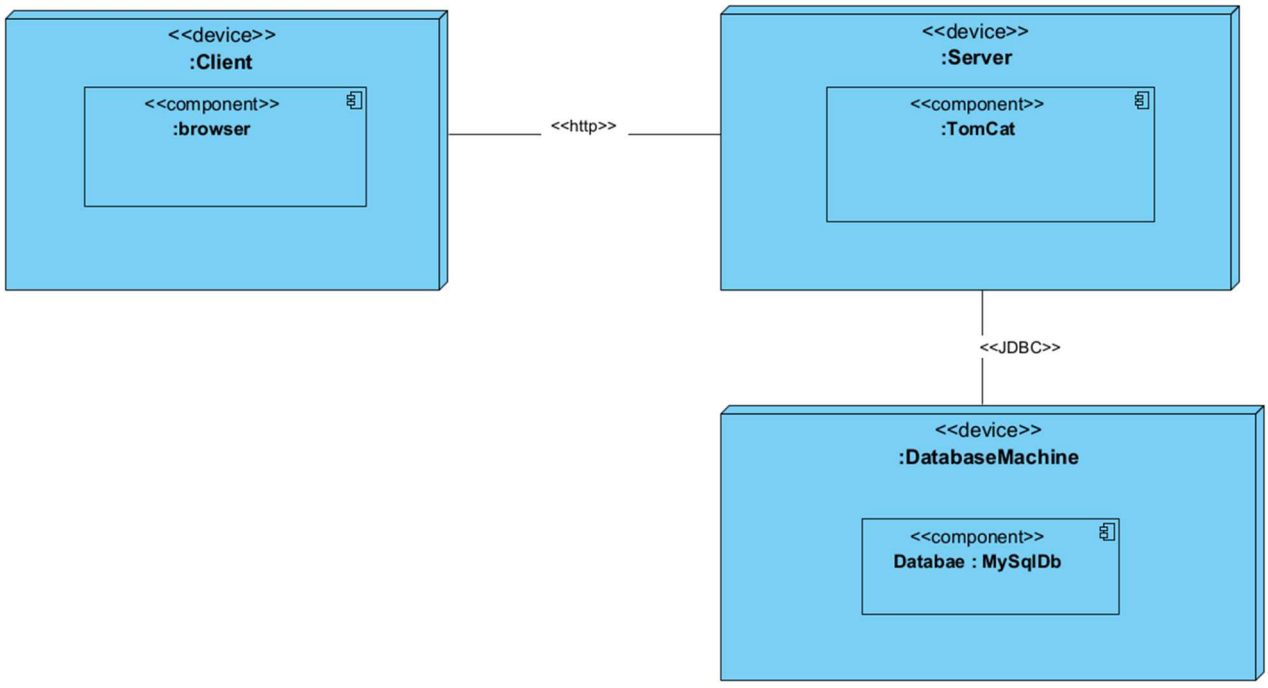
- **Richieste e risposte HTTP** tra il client e il server, per gestire le interazioni dell'utente.

- **Query JDBC** tra il server e il database, per garantire l'accesso e la gestione dei dati persistenti.

Component Diagram



Deployment Diagram



3.4 Gestione persistente dei dati

Procediamo con l'individuazione di tutti i dati/oggetti che devono essere resi persistenti all'interno del sistema:

- **Utente**
  - Tutti gli oggetti che rappresentano i ruoli associati (es. utente base e moderatore).

- **Post**
  - Per conservare tutti i contenuti pubblicati dagli utenti.

- **Commenti**
  - Per mantenere la traccia dei commenti associati ai post.

- **Community**
  - Per gestire la struttura e le informazioni relative alle community create dagli utenti.

- **Media Files**
  - Come immagini, video e file associati ai post.

Modalità di gestione della persistenza

Per ottimizzare la gestione delle risorse, il sistema adotta una modalità di **persistenza ibrida**:

1. **File System:**

   I dati di grandi dimensioni, come immagini, video associati ai post e alle community, verranno archiviati direttamente nel filesystem del server. Questa soluzione consente di gestire in modo efficiente file di grandi dimensioni senza sovraccaricare il database.

2. **Sessione per dati temporanei:**

   Per oggetti temporanei, che non devono essere condivisi tra dispositivi o sessioni diverse, utilizzeremo la sessione. Questo approccio riduce gli accessi al database e consente di risparmiare spazio di archiviazione.

3. **Database relazionale:**

   Tutti gli altri dati, come utenti, post, commenti, like, community e segnalazioni, saranno gestiti tramite un **database relazionale**. Questa scelta è motivata dall'eterogeneità dei dati e dalla necessità di garantire accessi concorrenti sicuri e coerenti.

   L'utilizzo di un database relazionale garantisce il recupero rapido ed efficiente dei dati necessari per le funzionalità del social network, mantenendo al contempo una gestione strutturata e scalabile delle informazioni persistenti.

3.5 Controllo degli accessi e sicurezza

Attore	Gestione Utente	Gestione Post	Gestione Community	Gestione Commenti
Utente non registrato	registrazione() ricercaUtente()	ricercaPost()	ricercaCommunity()	/
Utente registrato	login() logout() ricercaUtente() segnalaUtente() modificaCredenziali() seguiUtente() smettiDiSeguireUtente()	aggiuntaPost() eliminaPost() ricercaPost() salvaPost() segnalaPost() apprezzaPost() togliApprezzamentoPost ()	aggiuntaCommunity() eliminazioneCommunity() ricercaCommunity() modificaNomeCommunity segnalaCommunity()	aggiuntaCommento() eliminazioneCommento() apprezzaCommento() segnalaCommento()
Moderatore	login() logout() ricercaUtente() segnalaUtente() modificaCredenziali() seguiUtente() smettiDiSeguireUtente()	aggiuntaPost() eliminaPost() ricercaPost() salvaPost() segnalaPost() apprezzaPost() togliApprezzamentoPost ()	aggiuntaCommunity() eliminazioneCommunity() ricercaCommunity() modificaNomeCommunity segnalaCommunity()	aggiuntaCommento() eliminazioneCommento() apprezzaCommento() segnalaCommento()

4. SERVIZI DEI SOTTOSISTEMI

Sottosistema		Gestione Post
Descrizione		Questo sottosistema si occupa della gestione dei post
Servizio	Operazioni	Descrizione
Servizio Post	aggiunta post	Permette di aggiungere un post ad una community
	eliminazione post	Permette di rimuovere un post
Servizio Ricerca	ricerca post	Permette di ricercare un post
Servizio Segnalazione	segnala post	Permette di segnalare un post
Servizio Salvataggio Post	salva post	Permette di salvare un post
Servizio Apprezzamento	apprezza Post	Permette di apprezzare un post
	togli apprezzamento Post	Permette di togliere l'apprezzamento da un post

Sottosistema		Gestione Utente
Descrizione		Questo sottosistema si occupa della gestione degli utenti
Servizio	Operazioni	Descrizione
Servizio Ricerca Contenuto	ricerca utente	Permette di ricercare un utente
Servizio Segnalazione	segnala utente	Permette di segnalare un utente
Servizio Autenticazione	login	Permette a un utente di autenticarsi
	logout	Permette a un utente di disconnettersi
Servizio Registrazione	registrazione	Permette a un utente non registrato di registrarsi
Servizio Inibizione	inibisci utente	Permette di inibire un utente
Servizio Following	segui utente	Permette di seguire un utente
	smetti di seguire	Permette di smettere di seguire un utente
Servizio Modifica Credenziali	modifica credenziali	Permette di modificare le credenziali di un utente
Servizio Ruolo	Imposta ruolo	Permette di impostare il ruolo di un utente

Sottosistema		Gestione Commenti
Descrizione		Questo sottosistema si occupa della gestione dei commenti
Servizio	Operazioni	Descrizione
Servizio Commento	aggiunta commento	Permette di aggiungere un commento ad un post
	eliminazione commento	Permette di rimuovere un commento
Servizio Segnalazione	segnala commento	Permette di segnalare un commento

Sottosistema		Gestione Community
Descrizione		Questo sottosistema si occupa della gestione delle community
Servizio	Operazioni	Descrizione
Servizio Community	aggiunta community	Permette di aggiungere una community
	eliminazione community	Permette di rimuovere una community
Servizio Ricerca	ricerca community	Permette di ricercare una community
Servizio Modifica	modifica nome community	Permette di modificare il nome di una community

4.1 Controllo Software Globale

Sul lato client, il sistema utilizzerà un **control flow basato su eventi**, essenziale per la gestione dinamica dell'interfaccia grafica, dato che il sistema è progettato come un'applicazione web. L'interazione dell'utente con l'interfaccia sarà guidata da eventi, garantendo un'esperienza fluida e interattiva.

Sul lato server, il sistema adotterà un **control flow basato su thread**, che consentirà di gestire simultaneamente più richieste inviate dai client. Questa scelta è fondamentale per soddisfare il requisito implicito di permettere a più utenti di utilizzare la piattaforma contemporaneamente senza interferenze o rallentamenti, garantendo così un servizio efficiente e scalabile.

4.2 Boundary Condition

Le **boundary conditions** del sistema sono definite come segue:

- **Installazione del sistema:**

  L'installazione sarà eseguita da un operatore incaricato, il quale avrà anche il compito di inizializzare il database inserendo gli account predefiniti (ad esempio, per moderatori o amministratori di sistema). Il sistema sarà distribuito utilizzando un **server container Apache Tomcat**, che verrà installato su una macchina remota fornita da un'azienda di web hosting.

- **Avvio del sistema:**

  Il sistema sarà avviato da un operatore, che provvederà all'attivazione del **DBMS MySQL**. La connessione tra il sistema e il database sarà stabilita tramite driver **JDBC**, garantendo la corretta comunicazione tra le componenti.

- **Spegnimento del sistema:**

  Lo spegnimento del sistema sarà eseguito da un operatore, che dovrà seguire una procedura specifica. Prima di spegnere, sarà necessario:

  1. Completare eventuali transazioni in corso.
  2. Salvare tutti i dati critici.
  3. Chiudere correttamente le connessioni con il database, per garantire l'integrità e la consistenza delle informazioni.

- **Gestione dei guasti del sistema:**

  In caso di un **fallimento del sistema**, il comportamento previsto è il seguente:

1. **Interruzione delle attività in corso:** Tutte le operazioni attive al momento del fallimento saranno sospese.

2. **Backup e ripristino:** Un meccanismo di backup automatico garantirà la conservazione dei dati critici. Durante il riavvio, il sistema tenterà di ripristinare lo stato precedente utilizzando i dati di backup.

3. **Notifica dell'errore:** Il sistema genererà un report sull'errore verificatosi, che sarà inviato all'amministratore per analisi e risoluzione.

4. **Procedure di ripristino:** L'operatore dovrà verificare il funzionamento del database, del container Apache Tomcat e delle connessioni JDBC. Una volta individuata e risolta la causa del problema, il sistema potrà essere riavviato.

5. **Verifica dei dati:** Dopo il ripristino, si procederà a una verifica per assicurarsi che non vi siano inconsistenze o perdite di dati.