soundrate

Object Design Document

1. Introduzione

Dopo la stesura del documento di analisi dei requisiti e di progettazione del sistema ci si appresta a definire gli aspetti implementativi. Il documento di progettazione degli oggetti pone come obiettivo la produzione di un modello coerente con le funzionalità e le altre informazioni individuate nei documenti precedenti.

Il documento prevede una panoramica del sistema, le interfacce delle classi che lo compongono, le operazioni supportate, i tipi dei dati, i parametri delle procedure, le firme dei sottosistemi individuati, linee guida da seguire e trade-off.

1.1. Object design trade-offs

Seguono i trade-off individuati nel sistema.

1.1.1. Prestazioni Vs. Costi

Il sistema fa affidamento a tecnologie open source non richiedenti licenze a pagamento per l'utilizzo di queste, tuttavia ampiamente diffuse e notoriamente diffuse, come il sistema di gestione di basi di dati relazionali *MySQL* e l'application server *Apache TomEE*. Il committente si impegna nel pagare i servizi offerti da *Amazon Web Service* (*AWS*) per l'hosting dell'application server e della base di dati, il che risulta decisamente più economico del costruire un'infrastruttura privata da dover manutenere a lungo termine, e offre diversi altri vantaggi dal punto di vista delle prestazioni, come scalabilità automatica, replica e distribuzione trasparente di contenuto tramite nodi geografici.

1.1.2. Interfaccia Vs. Usabilità

La piattaforma **soundrate** nasce per semplificare e rendere attraenti le procedure di espressione di opinioni musicali, e di conseguenza vanterà di un'interfaccia utente intuitiva e concisa con lo scopo di semplificare l'utilizzo del sistema ai suoi utenti finali.

1.1.3. Sicurezza Vs. Efficienza

A causa del tempo limitato per lo sviluppo del sistema, la sicurezza coprirà esclusivamente gli aspetti strettamente necessari, come la comunicazione sicura tra client e server, l'accesso ristretto alle funzionalità amministrative e la necessità di autenticazione per usufruire delle funzionalità chiave della piattaforma. Sarà possibile in futuro estendere il sistema implementando autenticazione a due fattori o altre tecnologie per la protezione degli account utenti; tuttavia, come già descritto nella documentazione precedente, le credenziali degli utenti della piattaforma verranno memorizzate in maniera sicura, con rischio virtualmente inesistente che agenti malintenzionati possano ottenerne accesso.

1.2. Linee guida per la documentazione delle interfacce

Per assicurare un'alta leggibilità e di conseguenza una buona manutenibilità del codice, gli sviluppatori dovranno sottostare a delle linee guida per la stesura del codice.

1.2.1. Code style

Gli sviluppatori dovranno attenersi quanto più possibile a delle linee guida ben precise dettate dalle convenzioni di stile del codice *Google Java*. In particolare, il codice dovrà essere indentato utilizzando tab e non spazi, ed essere formattato seguendo uno stile conciso e comprensibile.

1.2.2. Naming convention

È buona pratica che i nomi da utilizzare siano:

- descrittivi;
- pronunciabili;
- di uso comune;
- non abbreviati (se non per variabili temporanee);
- contenenti esclusivamente caratteri consentiti (*A-Z*, *a-z*, *0-9*).

1.2.2.1. Classi

I nomi delle classi devono rispettare lo stile *upper camel case* (o *Pascal case*), dove la prima parola di ogni lettera è in maiuscolo. I nomi delle classi non dovrebbero contenere acronimi o abbreviazioni, a meno che l'abbreviazione non sia molto più diffusa della forma normale, come ad esempio URL o HTML).

Esempi:

- `class User {}`
- `class NameGenerator {}`

1.2.2.2. Metodi e funzioni

I nomi dei metodi e delle funzioni devono rispettare lo stile lower camel case (o Dromedary case), dove la prima lettera della prima parola è in minuscolo e la prima lettera di ogni parola successiva è in maiuscolo. I nomi di metodi e devono essere composti da verbi o da più parole che cominciano per un verbo.

Esempi:

- `run();`
- `runFast();`
- `getName();`

1.2.2.3. Variabili

Anche i nomi delle variabili locali, istanze di variabili, e variabili di classe devono rispettare lo stile *lower camel case*. I nomi delle variabili non dovrebbero iniziare per underscore (_) o per il simbolo del dollaro ($), anche se entrambi sono permessi.

I nomi di variabile devono cercare di essere corti ma significativi. La scelta del nome di una variabile dovrebbe essere mnemonica. Nomi di variabili composte da un singolo carattere andrebbero evitati se non per variabili temporanee come contatori o contenitori di singoli caratteri.

Esempi:

- `int i;`
- `char c;`
- `float averageRating;`

1.2.2.4. Costanti

I nomi delle costanti dovrebbero essere composte da parole in maiuscolo separate da underscore. I nomi delle costanti possono anche contenere cifre se appropriato, ma non come primo carattere.

Esempi:

- `static final int MAX_LENGTH = 10;`
- `const SUCCESS = 1`

2. Design pattern

Per assicurare lo sviluppo di un sistema coeso e facilmente manutenibile, si utilizzano diversi design pattern riutilizzabili per diverse componenti all'interno dell'applicazione.

2.1. Singleton

Il design pattern del *singleton* permette di restringere il numero di istanze di una classe ad una singola istanza.

2.1.1. Funzionalità

Il pattern del singleton permette la riduzione della memoria utilizzata per le classi che non hanno bisogno di mantenere uno stato o che possono usufruire di uno stato condiviso tra i suoi utilizzatori, in quanto esiste una singola istanza per ogni classe che adotta l'utilizzo del pattern.

Il design pattern del singleton viene ampiamente utilizzato per l'accesso centralizzato a funzionalità e componenti in un'unica entità condivisa dai suoi utilizzatori.

2.1.2. Utilizzo

Il sistema utilizzerà dei singleton per classi d'utilità e altre classi che non hanno bisogno di mantenere uno stato o con stato condiviso che possono essere accedute concorrentemente, come le classi che si occuperanno di mantenere in memoria i risultati delle richieste recenti sul catalogo.

2.2. Data access object (DAO)

Il design pattern del *data access object* (*DAO*) è un pattern utilizzato per fornire un'interfaccia ad una o più basi di dati o altri tipi di meccanismi di persistenza.

2.2.1. Funzionalità

Il pattern del DAO fornisce l'accesso a delle operazioni specifiche sui dati persistenti senza esporre dettagli sulla base di dati sottostante attraverso la mappatura di chiamate interne all'applicazione al layer di persistenza.

Un data access object permette di separare l'insieme di dati di cui un'applicazione richiede l'utilizzo da come gli stessi vengono trattati ad un livello più basso, permettendo all'utilizzatore di ignorare la tipologia di base di dati sottostante e le strutture dati che utilizzate.

2.2.2. Utilizzo

Il sistema adotterà il pattern DAO per le classi che si occuperanno di realizzare i servizi che implementano le diverse funzionalità individuate (funzionalità riguardanti gli account utenti, le recensioni, il catalogo, ecc.) e che dovranno di conseguenza lavorare con dati persistenti.

2.3. Observer

Il design pattern dell'*observer* è un pattern in cui un oggetto detto *subject* notifica un insieme di *osservatori* riguardo i suoi cambi di stato o esecuzioni di azioni.

2.3.1. Funzionalità

Il pattern observer viene generalmente usato in sistemi di gestione di eventi in cui il subject viene detto "sorgente del flusso di eventi", mentre gli observer vengono denominati "pozzi di eventi". Questo pattern risulta particolarmente utile per soddisfare dipendenze tramite accoppiamento debole.

2.3.2. Utilizzo

Il pattern observer verrà utilizzato dal sistema per aggiornare le sessioni degli utenti autenticati alla piattaforma se lo stato del loro account utente viene aggiornato, per mantenere le sessioni consistenti con lo stato dell'account utente.

Verrà inoltre utilizzato per aggiornare il contenuto delle cache implementate dall'applicazione in maniera trasparente, senza richiedere alcuna azione esplicita dall'utilizzatore.

Il pattern observer potrà inoltre essere utilizzato per risolvere problemi di logica trasversale, come il logging delle operazioni svolte dal sistema.

3. Package

Trattandosi di un'applicazione web basata sulla piattaforma *Java Enterprise Edition*, questa consisterà principalmente di due porzioni: una contenente classi e risorse Java, e una contenente risorse web come pagine web (in questo caso pagine e frammenti *JSP*) e ulteriore contenuto come fogli di stile CSS e file JavaScript per modellare l'aspetto dell'interfaccia e per permettere agli utenti di interagire con gli elementi che la compongono.

La prima delle due porzioni, in cui risiede la logica applicativa, consisterà di due package principali, suddivisi a loro volta in subpackage:

- il package `application` conterrà i subpackage che si occuperanno di implementare la logica applicativa, ovvero:
    - il package `entities` contiene le classi che vengono mappate alle entità della base di dati relazionale utilizzata dalla piattaforma (utente, recensione, voto, segnalazione).
    - il package `model` contiene le classi che si occupano di implementare le funzionalità offerte dal sistema (funzionalità riguardanti gli utenti, funzionalità riguardanti le recensioni, funzionalità riguardanti il catalogo della piattaforma) e un subpackage `exceptions` contenente le eccezioni lanciabili dalle classi che implementano i servizi.
    - altri subpackage di grado secondario che implementeranno la cache del sistema, classi di utilità, e componenti proprie della piattaforma *Java EE* come interceptor, eventi e producer.
- il package `endpoints` conterrà le classi che implementeranno gli endpoint HTTP che si occupano di soddisfare le richieste HTTP provenienti dai client, e consisterà dei seguenti subpackage:
    - il package `services` conterrà le classi che si occuperanno di soddisfare le richieste degli utenti che richiedono l'interazione con i servizi offerti dal sistema (ad esempio, la registrazione di un nuovo utente, l'aggiornamento di una recensione, l'autenticazione al proprio account utente, la restituzione delle recensioni segnalate).
    - il package `dispatchers` conterrà le classi che inoltreranno le richieste dei client alle pagine web della piattaforma, restituendole. Questo package sarà a sua volta separato nei package `fragments` e `pages`, dove il primo inoltrerà le richieste ai frammenti di pagine (ad esempio l'header), mentre il secondo a pagine web intere che potrebbero a loro volta fare uso dei frammenti.

Le classi nel package `endpoints` si occuperanno quindi di rispondere direttamente alle richieste dei client, interagendo con le classi dei subpackage di primo grado (`entities`, `model`) presente nel package `application`; il package `application` non dovrà avviare interazioni con il package `endpoints`, ignorandone l'esistenza.

3.1. Package `application.entities`

File	Descrizione
`User.java`	Classe rappresentante un utente registrato al sistema.
`Review.java`	Classe rappresentante una recensione pubblicata da un utente registrato per un progetto musicale.
`Vote.java`	Classe rappresentante un voto assegnato da un utente registrato a una recensione
`Report.java`	Classe rappresentante una segnalazione effettuata da un utente registrato a una recensione.
`BacklogEntry.java`	Classe rappresentante una entry di un progetto musicale nella lista d'ascolto di un utente registrato.

3.2. Package `application.model`

File	Descrizione
`UsersAgent.java`	Classe che implementa i servizi volti a soddisfare le funzionalità riguardanti gli account utenti.
`ReviewsAgent.java`	Classe che implementa i servizi volti a soddisfare le funzionalità riguardanti le recensioni.
`CatalogAgent.java`	Classe che implementa i servizi volti a soddisfare le funzionalità riguardanti il catalogo della piattaforma.

3.3. Package `endpoints.services`

File	Descrizione
`AuthenticationService.java`	Classe implementante gli endpoint che si interfacciano alle funzionalità riguardanti l'autenticazione degli utenti (login, logout, signup).
`CatalogService.java`	Classe implementante gli endpoint che si interfacciano alle funzionalità riguardanti il catalogo della piattaforma (aggiornamento lista d'ascolto, ecc.).
`ReviewsService.java`	Classe implementante gli endpoint che si interfacciano alle funzionalità riguardanti le recensioni (pubblicazione, aggiornamento, eliminazione di recensioni, ecc.).
`UsersService.java`	Classe implementante gli endpoint che si interfacciano alle funzionalità riguardanti gli account utenti (aggiornamento indirizzo e-mail, recupero password, ecc).

3.4. Package `endpoints.dispatchers.fragments`

File	Descrizione
`HeaderFragmentServlet.java`	Servlet che restituisce il frammento che compone l'header delle pagine web della piattaforma.
`SignInFragmentServlet.java`	Servlet che restituisce il frammento che compone il modal contenente i form per il log in e per il sign up degli utenti.
`UserSettingsFragmentServlet.java`	Servlet che restituisce il frammento che compone il modal contenente i form per l'aggiornamento dell'indirizzo e-mail e della password degli utenti.

3.5. Package `endpoints.dispatchers.pages`

File	Descrizione
`IndexPageServlet.java`	Servlet che restituisce la homepage della piattaforma.
`AlbumPageServlet.java`	Servlet che restituisce la pagina web per un determinato album.
`ArtistPageServlet.java`	Servlet che restituisce la pagina web per un determinato artista.
`SearchPageServlet.java`	Servlet che restituisce la pagina web contenente i risultati di una ricerca.
`ReviewPageServlet.java`	Servlet che restituisce la pagina web per una determinata recensione.
`UsersPageServlet.java`	Servlet che restituisce la pagina web per un determinato utente.
`BacklogPageServlet.java`	Servlet che restituisce la pagina web contenente la lista d'ascolto di un determinato utente.
`RecoverPageServlet.java`	Servlet che restituisce la pagina web per il recupero delle proprie credenziali.
`ResetPageServlet.java`	Servlet che restituisce la pagina web per reimpostare la password associata al proprio account utente.
`ReportsPageServlet.java`	Servlet che restituisce la pagina web per moderatori per permettere l'eliminazione di recensioni segnalate e delle segnalazioni di queste.