Report di Audit Ingegneristico

SWE_PROGETTO_VIDEOGIOCHI.pdf

SWE-Audit-Agent

23 febbraio 2026

Indice

1 Contesto del Documento

PANORAMICA STRUTTURATA DEL PROGETTO SOFTWARE

OBIETTIVO DEL PROGETTO

L'applicazione è una piattaforma per la gestione e la vendita di videogiochi che consente agli utenti di esplorare un catalogo, effettuare ricerche filtrate, aggiungere giochi al carrello e completare acquisti. I giochi acquistati vengono inseriti in una libreria personale dove l'utente può installarli, disinstallarli, avviarli e tracciare statistiche di gioco. Il sistema supporta abbonamenti mensili (Gold e Silver) con sconti esclusivi e accesso a giochi gratuiti, oltre a un ruolo amministratore per la gestione del catalogo e degli achievement.

CASI D'USO PRINCIPALI

- UC-001 - Acquistare un videogioco: Consente all'utente autenticato di selezionare giochi dal catalogo, aggiungerli al carrello e completare l'acquisto, aggiornando la libreria e il fondo. - UC-002 - Gestire la libreria personale: Permette all'utente di visualizzare, installare, disinstallare, avviare giochi e consultare statistiche e achievement associati. - UC-003 - Gestire il catalogo come amministratore: Consente all'admin di aggiungere, modificare, eliminare videogiochi e achievement e gestire lo stato gratuito dei giochi. - UC-004 - Gestire gli abbonamenti: Permette all'utente di scegliere e attivare un abbonamento (Gold o Silver) che offre sconti e accesso a giochi gratuiti.

REQUISITI FUNZIONALI

- Registrazione e autenticazione degli utenti con username, password e email. - Ricerca di videogiochi per genere, piattaforma e data di uscita. - Aggiunta e rimozione di videogiochi dal carrello. - Esecuzione di acquisti con verifica del saldo disponibile. - Gestione della libreria personale con installazione e disinstallazione di giochi. - Avvio di videogiochi e tracciamento del tempo di gioco. - Sblocco automatico di achievement in base al tempo di gioco. - Attivazione di abbonamenti Gold e Silver con applicazione di sconti (20% Gold, 10% Silver). - Ricarica del fondo dell'utente. - Operazioni CRUD su videogiochi e achievement da parte dell'admin. - Impostazione di videogiochi come gratuiti per gli abbonati.

REQUISITI NON FUNZIONALI

- Architettura modulare con separazione tra Domain Model, Business Logic e ORM. - Utilizzo di PostgreSQL come database relazionale. - Implementazione di pattern di design (Strategy per i prezzi, Observer per le notifiche, Singleton per la connessione al database). - Interfaccia utente a riga di comando (CLI). - Persistenza dei dati tramite JDBC e DAO. - Sicurezza mediante verifica delle credenziali e autorizzazione degli utenti.

ARCHITETTURA

L'applicazione segue un'architettura a strati con separazione delle responsabilità: il livello CLI comunica con la Business Logic (controller), che utilizza le entità del Domain Model e si affida all'ORM (DAO) per la persistenza. L'ORM traduce le operazioni sui dati in query SQL tramite JDBC verso PostgreSQL. Pattern utilizzati: Strategy per il calcolo dinamico dei prezzi in base agli abbonamenti, Observer per notificare la libreria degli acquisti completati, Singleton per la connessione al database. I controller principali sono UtenteController, CarrelloController, LibreriaController, CatalogoController, StatisticheController, AchievementController e AdminController.

STRATEGIA DI TESTING

I test sono implementati con JUnit 5 e verificano sia i flussi operativi standard che i casi di fallimento. Il database viene inizializzato tramite script SQL (reset.sql, ProgettoSWE.sql, InserimentoProgettoSWE.sql) prima di ogni test per garantire uno stato coerente. I test coprono: registrazione e autenticazione utenti, aggiunta e rimozione dal carrello, esecuzione acquisti con verifica del saldo, installazione e disinstallazione giochi, ricerca nel catalogo, attivazione abbonamenti, aggiornamento statistiche e sblocco

achievement. Sono testati anche i casi di errore come credenziali errate, fondi insufficienti, videogiochi duplicati e operazioni non autorizzate.

2 Sintesi Esecutiva

Panoramica Rapida

Problemi totali: **24** — HIGH: **4** MEDIUM: **15** LOW: **5**

Confidence media: **96%**

SINTESI ESECUTIVA AUDIT SWE_PROGETTO_VIDEOGIOCHI

INQUADRAMENTO DOCUMENTO

Il documento descrive l'architettura, i requisiti e il piano di testing per un sistema di vendita e gestione di videogiochi con supporto ad abbonamenti. Il progetto implementa un'architettura a tre livelli (Presentation, Business Logic, Data Access) con pattern di design quali DAO, Strategy e Observer. Sono stati identificati 24 problemi complessivi, di cui 4 critici (HIGH severity).

PATTERN DI PROBLEMI RISCONTRATI

I problemi riscontrati seguono tre pattern principali. Primo, violazioni dell'architettura dichiarata: il controller CatalogoController istanzia direttamente DAO invece di usare l'iniezione di dipendenze, compromettendo testabilità e coerenza. Secondo, specifiche incomplete dei requisiti: i casi d'uso non descrivono adeguatamente scenari critici come la gestione transazionale degli acquisti, il comportamento dei giochi gratuiti negli abbonamenti, e i flussi alternativi per errori di autorizzazione. Terzo, gap di copertura nei test: molti flussi alternativi critici (carrello vuoto, abbonamento già attivo, validazione input) non sono testati, e la tracciabilità tra UC e test è incoerente.

AREE CRITICHE (SEVERITY HIGH)

Quattro problemi richiedono attenzione immediata. REQ-001 riguarda l'assenza di transazionalità in eseguiAcquisto: il fondo viene scalato prima del trasferimento in libreria, rischiando inconsistenza del database in caso di errore. REQ-002 evidenzia che UC-002 non descrive flussi alternativi per errori critici di autorizzazione durante l'avvio di un videogioco. TST-001 e TST-002 segnalano che flussi alternativi critici (validazione input in UC-003 e abbonamento già attivo in UC-004) non hanno test corrispondenti.

VALUTAZIONE COMPLESSIVA

La qualita del documento è mediocre. L'architettura è ben strutturata nei principi ma violata nell'implementazione. I requisiti sono descritti in modo generico e privo di dettagli su scenari critici e comportamenti transazionali. Il piano di testing copre i flussi standard ma presenta lacune significative su casi di errore e scenari edge, con tracciabilità incoerente tra UC e test. La documentazione non affronta aspetti non funzionali come robustezza e gestione di errori di persistenza.

AZIONI PRIORITARIE

1. Refactoring architetturale: eliminare le istanziazioni dirette di DAO in CatalogoController e applicare coerentemente l'iniezione di dipendenze in tutti i controller, con verifica tramite test di integrazione.

2. Completamento specifiche requisiti: dettagliare UC-001, UC-002 e UC-004 con flussi alternativi espliciti per errori, definire il comportamento transazionale di eseguiAcquisto con rollback, e specificare la gestione dei giochi gratuiti negli abbonamenti.

3. Estensione della copertura di test: aggiungere test per flussi alternativi critici (validazione input, abbonamento attivo, carrello vuoto, libreria vuota), creare una matrice di tracciabilità UC-Test, e implementare test di integrazione per scenari transazionali.

3 Punti di Forza

PUNTI DI FORZA DEL DOCUMENTO

- Architettura modulare ben strutturata: il progetto è organizzato in tre pacchetti distinti (Domain Model, Business Logic, ORM) con chiara separazione delle responsabilità e relazioni ben definite tra i livelli, garantendo manutenibilità e scalabilità.

- Documentazione completa dei requisiti: sono presenti use case diagram dettagliati per utenti e admin, con template esaustivi che descrivono precondizioni, step, svolgimenti alternativi e postcondizioni per i casi d'uso principali.

- Copertura di testing approfondita: il documento evidenzia test su più livelli (Business Logic, Domain Model, ORM) con riferimenti specifici ai test case (Test 28, 29, 30, ecc.), dimostrando un approccio sistematico alla verifica.

- Design pattern implementati correttamente: l'uso del pattern Strategy per il calcolo dei prezzi in base all'abbonamento mostra consapevolezza nell'applicazione di pattern di design appropriati al contesto.

- Diagrammi UML completi: sono presenti use case diagram, class diagram con suddivisione nei pacchetti e ER diagram che rappresentano efficacemente la struttura del sistema.

- Mockup dell'interfaccia utente: sono forniti prototipi visivi delle schermate principali (login, catalogo, carrello, libreria, admin) che facilitano la comprensione delle funzionalità e dell'esperienza utente.

4 Scorecard per Area

Area	HIGH	MEDIUM	LOW	Punteggio	Voto
Architettura	0	2	2	74/100	**C**
Requisiti	2	7	1	0/100	**F**
Testing	2	6	2	0/100	**F**
Media Complessiva				**24/100**	**F**

5 Copertura Feature Attese

Su 7 feature attese: 6 presenti, 1 parziali, 0 assenti. Copertura media: **88%**.

Feature	Stato	Copertura	Evidenza
Unit testing framework implementation	Presente	80% (4/5)	La sezione 4 descrive test JUnit 5 con assertNotNull, assertEquals, assertThrows, assertDoesNotThrow, quindi verifica basata su asserzioni; sono presenti classi di test dedicate (UtenteControllerTest, CatalogoControllerTest, CarrelloControllerTest, LibreriaControllerTest, VideogiocoDAOTest, CarrelloDAOTest, LibreriaDAOTest) e viene spiegato un approccio sistematico ai test di singoli componenti e ai casi di fallimento. Sono coperti test di business logic e test del pacchetto ORM (integrazione con DB). Non c'è evidenza di uso di mock per dipendenze: i test usano un vero database inizializzato via script SQL.

Feature	Stato	Copertura	Evidenza
Use of UML Diagrams for system modeling	Presente	100% (8/8)	La sezione 2.1 include due Use Case Diagram (Utente e Admin) con attori, casi d'uso e relazioni «include»/«extend»; la sezione 2.4 presenta un diagramma di classe UML con pacchetti Domain Model, Business Logic, ORM e relazioni; la sezione 2.3 mostra numerosi mockup di interfaccia (menu iniziale, login, schermata principale, catalogo, carrello, libreria, menu admin) che chiariscono flussi di navigazione e requisiti funzionali. Le descrizioni indicano uso di notazione UML standard e relazioni tra attori e use case.
Identification and definition of system actors	Presente	100% (8/8)	Gli attori Utente e Admin sono identificati esplicitamente nei diagrammi dei casi d'uso e nei template UC-001..UC-004; le responsabilità di ciascun ruolo (esplorare catalogo, gestire carrello, libreria, abbonamenti per l'utente; CRUD videogiochi e achievement, gestione gratuiti per l'admin) sono descritte nello statement del progetto, nei casi d'uso e nei diagrammi. I template dei casi d'uso elencano azioni specifiche, flussi e alternative, strutturando i requisiti funzionali per ciascun ruolo.
Definition and Documentation of Use Cases	Presente	100% (5/5)	La sezione 2.2 contiene quattro Use Case Template (UC-001..UC-004) che definiscono interazioni specifiche (es. acquistare un videogioco, gestire libreria, gestire catalogo come admin, gestire abbonamenti), con campo "Descrizione" che esplicita il goal utente, passi numerati con flussi principali e "Svolgimenti alternativi" per i casi eccezionali; per ogni UC sono indicate preconditions e postconditions. Le relazioni tra casi d'uso (include/extend) sono descritte nei diagrammi di Figura 1 e 2.

Feature	Stato	Copertura	Evidenza
User interface and interaction design principles	Parziale	57% (4/7)	La sezione 2.3 fornisce mockup per tutte le schermate chiave (menu iniziale, login, menu utente con voci Catalogo/Carrello/Libreria/Ricarica Fondo/Gestione Abbonamento/Logout, catalogo con filtri e pulsanti Agg. al Carrello, carrello con totale e pulsanti Acquista, libreria con Installa/Disinstalla/Avvia/Statistiche, menu admin), mostrando elementi di navigazione chiari e campi strutturati per input (username/password, filtri di ricerca). I casi d'uso descrivono la gestione di errori e condizioni (es. fondo insufficiente, libreria vuota), ma non c'è una documentazione esplicita di meccanismi di validazione input né un processo di prenotazione.
Separation of concerns in software architecture	Presente	80% (4/5)	La sezione 1.2–1.3 e il diagramma di classe (Figura 10) descrivono una suddivisione in pacchetti Domain Model, Business Logic (controller) e ORM (DAO), più il livello Interfaccia (CLI) e JDBC/DBMS; sono spiegate le relazioni tra CLI, Business Logic, ORM, JDBC e RDBMS e il ruolo di ciascun componente. Non è presente un package "Service" separato, ma la Business Logic svolge quel ruolo; le interazioni tra controller e domain model/DAO sono descritte dettagliatamente nelle sezioni 3.2 e 3.3.
Data Access Object (DAO) pattern	Presente	100% (7/7)	La sezione 3.3 descrive numerosi DAO (UtenteDAO, VideogiocoDAO, CarrelloDAO, LibreriaDAO, AbbonamentoDAO, StatisticheVideogiocoDAO, AchievementDAO, AdminDAO) ciascuno dedicato a un'entità; gli snippet mostrano SQL incapsulato nei DAO (INSERT, SELECT, UPDATE con ON CONFLICT) e metodi CRUD come save, findById, findByCriteria, update, delete. La Business Logic usa questi DAO per astrarre l'accesso al DB, e la sezione 4.3 documenta test specifici per il pacchetto ORM (VideogiocoDAOTest, CarrelloDAOTest, LibreriaDAOTest).

6 Tabella Riassuntiva

Categoria	HIGH	MEDIUM	LOW	Totale
Architettura	0	2	2	4
Requisiti	2	7	1	10
Testing	2	6	2	10
Totale	**4**	**15**	**5**	**24**

7 Dettaglio delle Problematiche

7.1 Architettura (4 problemi)

UC-001

ARCH-002 — MEDIUM [100%] — Pagina 17 In ‘CatalogoController.getVideogiochiPerUtente‘ viene istanziato direttamente un nuovo ‘AbbonamentoDAO‘ invece di usare l’istanza iniettata come attributo, violando la separazione dei livelli e rendendo difficile il test e la coerenza con il resto dell’architettura. Inoltre, il metodo ricrea nuovi oggetti ‘Videogioco‘ con prezzi personalizzati, ma questo comportamento non è descritto nei requisiti: non è chiaro se il prezzo "personalizzato" debba essere persistito o solo calcolato a runtime.

> *public List<Videogioco> getVideogiochiPerUtente(Utente utente) { List<Videogioco> videogiochi = videogiocoDAO.findByCriteria(null, null, null); if (utente != null && utente.isAbbonamentoAttivo()) { Abbonamento abbonamento = new AbbonamentoDAO().findByUtenteId(utente.getId()); PrezzoStrategy prezzoStrategy = switch (abbonamento != null ? abbonamento. getTipo() : "Nessuno") { case "Silver" -> new SilverAbbonatoPrezzoStrategy(); case "Gold" -> new GoldAbbonatoPrezzoStrategy(); default -> new NonAbbonatoPrezzoStrategy(); }; return videogiochi.stream() .map(v -> new Videogioco(v.getId(), v.getTitolo(), v.getGenere(), v.getPiattaforma(), prezzoStrategy.calculatePrice(v, utente), v.getDataUscita(), v.isGratuito())) .collect(Collectors.toList()); } return videogiochi; }*

Raccomandazione: Adeguare l’architettura e i requisiti: (1) usare l’attributo ‘abbonamentoDAO‘ già presente nel controller invece di ‘new AbbonamentoDAO()‘, mantenendo l’iniezione di dipendenze coerente; (2) documentare nei requisiti (ad esempio in UC-001 o in una sezione non funzionale) che i prezzi mostrati nel catalogo sono calcolati dinamicamente in base all’abbonamento e non modificano il prezzo base nel database; (3) aggiungere test che verifichino che il prezzo base in ‘videogiochi‘ resti invariato e che solo la vista per l’utente applichi la strategia di sconto.

Vedi anche: REQ-001, REQ-004, REQ-005, TST-004, TST-006, TST-008

Problemi Generali

ARCH-001 — MEDIUM [100%] — Pagina 13 L’architettura dichiara una chiara separazione tra Business Logic e ORM tramite DAO, ma in CatalogoController.getVideogiochiPerUtente viene istanziato direttamente un nuovo AbbonamentoDAO invece di usare l’istanza iniettata, violando il principio di inversione delle dipendenze e rendendo più difficile il testing e il mocking.

> *La Business Logic coordina le interazio- ni tra le entit‘a del Domain Model (es. Utente, Videogioco, Carrello, Libreria, Abbonamento, StatisticheVideogioco, Achievement, Admin) e il database tramite i Data Access Object (DAO).*

Raccomandazione: Modificare CatalogoController per utilizzare solo DAO passati via costruttore o setter. In particolare, sostituire la creazione diretta di new AbbonamentoDAO() nel metodo getVideogiochiPerUtente con l’uso di un campo abbonamentoDAO iniettato (come fatto

in altre classi). Aggiornare i test di CatalogoControllerTest per verificare che il controller utilizzi il DAO iniettato (eventualmente introducendo un mock o una finta implementazione).

ARCH-003 — LOW [100%] — Pagina 11 Il diagramma di classe elenca i principali componenti ma non mostra in modo esplicito alcune dipendenze importanti emerse nel codice, come l'uso del pattern Observer tra 'CarrelloController' e 'LibreriaController' o il pattern Strategy tra 'Carrello' e 'PrezzoStrategy'. Questo rende difficile per il lettore comprendere l'architettura dei pattern rispetto alla descrizione testuale nella sezione Business Logic.

> *2.4 Class Diagram In questa sezione viene riportato il diagramma di classe del progetto [IMAGE 1]: 1. **Image Type:** - Diagram (UML Class Diagram) 2. **Details:** - **Diagram Type:** UML Class Diagram - **Elements:** - **Business Logic:** - 'CatalogoController' - 'AdminController' - 'UtenteController' - 'CarrelloController' - 'AchievementController' - 'StatisticheController' - 'LibreriaController' - **Domain Model:** - 'Videogioco' - 'Achievement' - 'StatisticheVideogioco' - 'Utente' - 'Admin' - 'Libreria' - 'Carrello' - 'Abbonamento' - **ORM:** - 'CarrelloDAO' - 'UtenteDAO' - 'AdminDAO' - 'AbbonamentoDAO' - 'AchievementDAO' - 'LibreriaDAO' - 'VideogiocoDAO' - 'StatisticheVideogiocoDAO' - 'DatabaseConnection' (Singleton) - 'DBMS'*

Raccomandazione: Aggiornare il diagramma di classe per includere: (1) l'interfaccia 'PrezzoStrategy' e le sue implementazioni ('GoldAbbonatoPrezzoStrategy', 'SilverAbbonatoPrezzoStrategy', 'NonAbbonatoPrezzoStrategy'), con la relazione di composizione/associazione con 'Carrello'; (2) l'interfaccia 'Observer' e la relazione tra 'CarrelloController' (Subject) e 'LibreriaController' (Observer); (3) eventuali altre interfacce o pattern usati (es. Singleton per 'DatabaseConnection'). Questo renderà il diagramma coerente con la descrizione testuale e aiuterà a verificare la corretta applicazione dei pattern rispetto ai requisiti.

ARCH-004 — LOW [100%] — Pagina 12 Il pattern Strategy per il calcolo dei prezzi è ben descritto e testato a livello di Carrello e CarrelloController, ma non è verificato in combinazione con CatalogoController.getVideogiochiPerUtente, che applica anch'esso una PrezzoStrategy per personalizzare i prezzi nel catalogo. Manca un test di integrazione che assicuri coerenza tra le due parti dell'architettura (prezzi mostrati nel catalogo vs prezzi usati per il totale del carrello).

> *La classe Carrello modella il carrello di un utente, con attributi che permettono la selezione di videogiochi per l'acquisto (videogiochi, prezzoStrategy) e l'associazione all'utente (idUtente). In particolare, l'at- tributo videogiochi contiene la lista dei giochi selezionati, mentre prezzoStrategy implementa il pattern Strategy tramite l'interfaccia PrezzoStrategy e le sue implementazioni (GoldAbbonatoPrezzoStrategy, SilverAbbonatoPrezzoStrategy, NonAbbonatoPrezzoStrategy).*

Raccomandazione: Aggiungere un test di integrazione in CatalogoControllerTest, ad esempio @Test void testGetVideogiochiPerUtente_PrezziCoerentiConCarrello(), che: (1) crea o recupera un utente con abbonamento Gold o Silver, (2) ottiene la lista di videogiochi personalizzati tramite getVideogiochiPerUtente, (3) aggiunge uno di questi videogiochi al Carrello e calcola il totale, (4) verifica che il prezzo mostrato nel catalogo e quello usato nel carrello (via PrezzoStrategy) coincidano.

7.2 Requisiti (10 problemi)

UC-001

REQ-001 — HIGH [100%] — Pagina 14 La logica di 'eseguiAcquisto' non è transazionale: il fondo dell'utente viene aggiornato e salvato prima del trasferimento in libreria e della cancellazione del carrello. In caso di errore in 'trasferisciInLibreria', 'notificaAcquistoCompletato' o 'carrelloDAO.delete', il sistema può lasciare il database in uno stato incoerente (fondo scalato

ma giochi non in libreria o carrello non svuotato). Non esiste un requisito o un caso d'uso che descriva il comportamento atomico dell'operazione di acquisto o la gestione di rollback.

```
public boolean eseguiAcquisto(Carrello carrello) { Utente utente = utenteDAO.findById(carrello.getIdUtente());
if (utente == null) { throw new IllegalStateException("Utente non trovato."); } double totale
= carrello.getTotale(utente); Libreria libreria = libreriaDAO.findByUtenteId(utente.getId());
if (libreria != null) { for (Videogioco v : carrello.getVideogiochi()) { if (libreria.getVideogiochiAcquistati().contains(v
{ throw new IllegalArgumentException("Il gioco " + v.getTitolo() + " e ' gia' presente nella
tua libreria."); } } } if (utente.getFondo() >= totale) { utente.setFondo(utente.getFondo()
- totale); utenteDAO.update(utente); trasferisciInLibreria(utente, carrello.getVideogiochi());
notificaAcquistoCompletato(utente, carrello); carrelloDAO.delete(carrello.getId()); return true;
} else { throw new IllegalStateException("Fondo insufficiente per completare l' acquisto.");
} }
```

Raccomandazione: Introdurre un requisito funzionale esplicito che definisca l'acquisto come operazione atomica: o tutte le modifiche (fondo, libreria, carrello) vanno a buon fine, o nessuna viene applicata. A livello di implementazione, usare una transazione DB che includa l'update dell'utente, il salvataggio/aggiornamento della libreria e la cancellazione del carrello, con rollback in caso di eccezioni. Aggiornare UC-001 aggiungendo un flusso di errore che descriva cosa succede se l'operazione fallisce a metà (messaggio all'utente, nessun addebito, carrello invariato). Aggiungere test che simulino errori in 'libreriaDAO' o 'carrelloDAO' e verifichino che il fondo non venga scalato in modo permanente.

Vedi anche: ARCH-002, TST-004, TST-006, TST-008

REQ-004 — MEDIUM [100%] — Pagina 5 Nel Use Case Diagram Utente alcune relazioni include/extend sono semanticamente discutibili e non coerenti con i template dei casi d'uso. Ad esempio, "Search Games" che include "Add to Cart" implica che ogni ricerca comporti necessariamente l'aggiunta al carrello, mentre UC-001 prevede che l'utente possa visualizzare risultati senza aggiungere nulla. Analogamente, "View Catalog" include "View Cart", ma nel mockup il carrello è una funzionalità separata. Questo crea ambiguità sui flussi principali e opzionali.

Relationships: - "Registration" extends "Login" - "Search Games" includes "Add to Cart" - "View Catalog" includes "View Cart" - "Add to Cart" includes "Remove Game" - "Add to Cart" includes "Purchase Games" - "Install Game" includes "Manage Games" - "Manage Games" includes "Uninstall Game" - "Manage Games" includes "Launch Game" - "View Library" includes both "Achievements" and "Manage Games"

Raccomandazione: Rivedere il Use Case Diagram Utente per allinearlo ai template UC-001/UC-002 e ai mockup: (1) trasformare le relazioni include in relazioni più appropriate (ad esempio, rimuovere l'include tra "Search Games" e "Add to Cart" e modellare l'aggiunta al carrello come un passo opzionale nel caso d'uso di ricerca/visualizzazione catalogo); (2) separare chiaramente "View Catalog" e "View Cart" come casi d'uso distinti, eventualmente collegati da una relazione di navigazione ma non da «include»; (3) aggiornare la documentazione testuale per spiegare quali azioni sono obbligatorie e quali opzionali. Assicurarsi che ogni relazione include/extend sia giustificata da un passo esplicito nei template dei casi d'uso.

Vedi anche: ARCH-002, TST-004, TST-006, TST-008

REQ-005 — MEDIUM [100%] — Pagina 6 Il caso d'uso UC-001 non specifica chiaramente le condizioni sui contenuti del carrello prima dell'acquisto (es. presenza di giochi duplicati, giochi gratuiti, giochi già in libreria) e non descrive cosa accade in caso di errori applicativi o di persistenza durante l'aggiornamento di libreria e fondo. Inoltre, il flusso alternativo 6a parla di "accede ad un carrello vuoto" ma non è chiaro se si riferisce alla sola visualizzazione o al tentativo di acquisto, creando ambiguità rispetto al comportamento atteso.

Use Case 1: Acquistare un videogioco (UC-001) ... Preconditions L'utente deve essersi autenticato nel sistema e si deve trovare nel menu principale Steps 1. L'utente seleziona l'opzione per accedere al catalogo (dal menu in Mock-up 5) 2. L'utente sceglie tra l'opzione di visualizzazione o di ricerca 3. L'utente visualizza l'elenco completo di tutti i giochi acquistabili, oppure solo quelli filtrati attraverso la ricerca (Mock-up 6, Test 29) 4. L'utente seleziona il videogioco che vuole acquistare e lo aggiunge al carrello 5. L'utente pu'o visualizzare il carrello e costo totale (Mock-up 7) 6. L'utente esegue l'acquisto dei giochi nel carrello. Svolgimenti alternativi • 4a. Se l'utente sceglie un gioco che ha gi'a acquistato viene notificato con un messaggio e riprova (Test 30) • 6a. Se l'utente accede ad un carrello vuoto riceve un messaggio e torna al catalogo • 6b. Se l'utente ha un fondo insufficiente viene notificato e non pu'o completare l'acquisto (Test 28) Postconditions Il carrello viene svuotato, i giochi sono spostati nella libreria personale e salvati nel sistema e il fondo viene aggiornato.

Raccomandazione: Espandere UC-001 aggiungendo: (1) una precondizione sullo stato del carrello (ad esempio: "Il carrello può contenere solo videogiochi non già presenti in libreria" oppure specificare che il controllo è fatto al passo 6); (2) un flusso alternativo esplicito per il caso in cui, al passo 6, il sistema rilevi videogiochi già presenti in libreria (allineandosi al comportamento di 'eseguiAcquisto' che lancia un'eccezione); (3) chiarire il passo 6a, distinguendo tra semplice visualizzazione di un carrello vuoto e tentativo di acquisto, descrivendo i messaggi e la schermata di ritorno; (4) aggiungere un flusso di errore per problemi di salvataggio su database (ad esempio: il sistema mostra un messaggio di errore, non modifica il fondo, non svuota il carrello).

Vedi anche: ARCH-002, TST-004, TST-006, TST-008

UC-002

REQ-002 — HIGH [100%] — Pagina 16 La logica di avvio videogioco gestisce casi di errore importanti (gioco non acquistato, non installato, utente non autorizzato) tramite eccezioni, ma UC-002 non descrive nessun flusso alternativo per questi scenari. Non è definito come la CLI debba reagire a queste eccezioni (messaggi, ritorno al menu, eventuale logging), lasciando un gap tra requisiti e comportamento effettivo in casi critici di autorizzazione.

```
public void avviaVideogioco(Utente utente, Videogioco videogioco, int tempo) { if (utente.getId() == libreria.getIdUtente()) { if (libreria.getVideogiochiAcquistati().contains(videogioco))
{ if (libreria.isVideogiocoInstallato(videogioco)) { System.out.println("Avvio del videogioco: "
+ videogioco.getTitolo() + " per " + tempo + " secondi"); statisticheController.aggiornaTempoGioco(utente,
videogioco, tempo); } else { throw new IllegalStateException("Videogioco non installato"); }
} else { throw new IllegalArgumentException("Videogioco non acquistato"); } } else { throw
new SecurityException("Utente non autorizzato"); } }
```

Raccomandazione: Estendere UC-002 con flussi alternativi specifici per l'avvio del videogioco: (1) aggiungere un ramo "3b.x" per il caso "videogioco non acquistato" con passi come: il sistema mostra un messaggio di errore, non aggiorna le statistiche, ritorna alla schermata della libreria; (2) un ramo per "videogioco non installato" con messaggio e suggerimento di installazione; (3) un ramo per "utente non autorizzato" (ad esempio, se si tenta di usare una libreria non propria). Aggiornare la descrizione della CLI per specificare come queste eccezioni vengono mappate in messaggi all'utente e aggiungere test di integrazione che verifichino tali comportamenti a livello di controller/CLI.

Vedi anche: TST-003, TST-005, TST-007

REQ-006 — MEDIUM [100%] — Pagina 7 UC-002 mescola in un unico passo 4 sia l'aggiornamento dello stato di installazione sia l'aggiornamento del tempo di gioco, ma nei flussi alternativi 3b e 3c introduce azioni aggiuntive (avvio gioco, visualizzazione statistiche/achievement) che non sono coperte nei passi principali. Inoltre, non è specificato cosa accade se l'utente tenta di avviare

un gioco non installato o non acquistato, mentre la logica di 'LibreriaController.avviaVideogioco' lancia eccezioni in questi casi.

> *Use Case 2: Gestire la libreria personale (UC-002) ... Steps 1. L'utente seleziona l'opzione per accedere alla propria libreria di giochi (Mock-up 5) 2. Il sistema mostra i giochi acquistati dall'utente, con la loro descrizione e stato di installazione (Mock-up 8) 3. L'utente seleziona di installare o disinstallare un gioco 4. Il sistema aggiorna lo stato di installazione o il tempo di gioco associato al videogioco selezionato. Svolgimenti alternativi • 3a. Se la libreria 'e vuota l'utente riceve un messaggio per avvertirlo di non poter compiere azioni. • 3b. L'utente seleziona l'opzione di avviare un videogioco e giocarlo – 3b.1 L'utente seleziona quanto tempo vuole giocare al videogioco. – 3b.2 L'utente simula di giocare a un videogioco e viene aggiornato il tempo di gioco. • 3c. L'utente seleziona l'opzione di visualizzare statistiche e achievement relativi ad un videogioco (Test 32) – 3c.1 Il sistema mostra le statistiche associate al gioco. – 3c.2 se non ci sono achievement associati ad un gioco il sistema manda un messaggio. (Test 33) • 3d. Se l'utente seleziona di installare un gioco gi'a installato o viceversa il sistema glielo impedisce e manda un messaggio (Test 31) Postconditions Il sistema aggiorna lo stato di installazione o il tempo di gioco associato al videogioco selezionato e le modifiche sono salvate nel database*

Raccomandazione: Ristrutturare UC-002 separando chiaramente: (1) un sotto-flusso per installazione/disinstallazione (passi 3–4) con postcondizione specifica sul solo stato di installazione; (2) un sotto-flusso per avvio gioco (3b) con passi dettagliati: controllo che il gioco sia acquistato e installato, aggiornamento del tempo di gioco, eventuale sblocco achievement; (3) un sotto-flusso per visualizzazione statistiche/achievement (3c) con precondizioni (es. esistenza di statistiche). Aggiungere flussi alternativi espliciti per i casi gestiti da 'avviaVideogioco' (gioco non acquistato, non installato, utente non autorizzato), descrivendo i messaggi mostrati e la schermata di ritorno, così da allineare requisiti e implementazione.

Vedi anche: TST-003, TST-005, TST-007

UC-003

REQ-003 — MEDIUM [100%] — Pagina 3 Lo statement del progetto introduce "accesso a una selezione di giochi gratuiti" per gli abbonamenti, ma nei casi d'uso e nella logica di business non è specificato come questi giochi gratuiti vengono gestiti: non è chiaro se entrino automaticamente in libreria, se siano giocabili solo finché l'abbonamento è attivo, né come siano distinti dai giochi acquistati. L'ER diagram ha 'is_gratuito', ma i casi d'uso non descrivono scenari specifici per l'uso di giochi gratuiti.

> *Il sistema prevede inoltre due tipologie di abbonamento mensile, Gold e Silver, che offrono agli utenti sconti esclusivi e accesso a una selezione di giochi gratuiti. Per aver accesso a questi abbonamenti e poter comprare i videogiochi che ha inserito nel carrello, un utente ha un fondo, che pu'o ricaricare.*

Raccomandazione: Definire requisiti funzionali dedicati alla gestione dei giochi gratuiti: (1) descrivere come l'admin seleziona i giochi gratuiti (già parzialmente accennato in UC-003, ma da esplicitare); (2) aggiungere un caso d'uso utente (o estendere UC-002/UC-004) che specifichi come l'utente vede e utilizza i giochi gratuiti (visualizzazione nel catalogo, aggiunta in libreria, limitazioni temporali legate all'abbonamento); (3) chiarire cosa accade ai giochi gratuiti quando l'abbonamento scade (restano giocabili? vengono rimossi dalla libreria?); (4) aggiungere test che coprano almeno un flusso di utilizzo di un gioco gratuito con abbonamento attivo e dopo la scadenza.

Vedi anche: TST-001

REQ-007 — MEDIUM [94%] — Pagina 7 UC-003 è troppo generico rispetto alle molte operazioni amministrative effettivamente implementate (CRUD videogiochi, CRUD achievement, gestione gratuito). Non specifica precondizioni e postcondizioni per ciascuna azione (es. vincoli di unicità sui titoli, gestione di videogiochi referenziati in carrelli o librerie, impatto su abbonamenti e giochi gratuiti). Inoltre, non sono descritti i flussi alternativi per casi come tentativo di rimozione di un videogioco inesistente o con vincoli referenziali, mentre i DAO permettono delete diretti.

> *Use Case 3: Gestire il catalogo come amministratore (UC-003) ... Steps 1. L'admin ha accesso al menu da amministratore (Mock-up 9) 2. L'admin seleziona azione: aggiungi, modifica, elimina videogioco o achievement oppure imposta/rimuovi gratuito (Test 29 e Test 33). 3. Il sistema esegue l'operazione e mostra la conferma. Svolgimenti alternativi • 1a. Se le credenziali non sono valide viene mandato un messaggio di errore (Test 34) • 2a. Se i dati immensi dall'admin durante la creazione o la modifica non sono validi riceve messaggio di errore Postconditions Il catalogo viene aggiornato nel database e le modifiche ai videogiochi e agli achievement salvate*

Raccomandazione: Scomporre UC-003 in più casi d'uso specifici (es. "Aggiungere videogioco", "Modificare videogioco", "Eliminare videogioco", "Gestire stato gratuito", "Gestire achievement"). Per ciascuno: (1) definire precondizioni (admin autenticato, entità esistente/non esistente); (2) descrivere i passi dettagliati, inclusi i controlli di validità (es. ID valido, formato dati); (3) aggiungere flussi alternativi per entità inesistenti, violazioni di vincoli (videogioco presente in librerie/carrelli), errori di persistenza; (4) specificare postcondizioni precise (es. "il videogioco è marcato gratuito e visibile come tale nel catalogo"). Allineare i messaggi di errore con quelli effettivamente lanciati dai controller/DAO e coperti dai test.

Vedi anche: TST-001

REQ-010 — LOW [100%] — Pagina 5 Nel Use Case Diagram Admin, il caso d'uso "CRUD Videogiochi" estende "Aggiungi Achievement", mentre la gestione degli achievement è già modellata separatamente in "CRUD Achievement". Questo crea una relazione poco chiara: non è specificato in quali condizioni l'aggiunta di un achievement estenda le operazioni CRUD sui videogiochi, e UC-003 non descrive questo legame. Il diagramma rischia di confondere la responsabilità tra gestione videogiochi e gestione achievement.

> *Relationships: - CRUD Videogiochi «include» Aggiungi Videogioco - CRUD Videogiochi «include» Rimuovi Videogioco - CRUD Videogiochi «include» Modifica Videogioco - CRUD Videogioc hi «extend» Aggiungi Achievement - Gestione Abbonamenti «include» Rimuovi Videogioco Gratuito - Gestione Abbonamenti «include» Imposta Videogioco Gratuito - CRUD Achievement «include» Aggiungi Achievement - CRUD Achievement «include» Modifica Achievement - CRUD Achievement «include» Rimuovi Achievement*

Raccomandazione: Chiarire il rapporto tra gestione videogiochi e gestione achievement: (1) se l'aggiunta di achievement è una funzionalità separata, rimuovere l'«extend» da "CRUD Videogiochi" verso "Aggiungi Achievement" e mantenere solo "CRUD Achievement"; (2) se invece l'aggiunta di achievement è un passo opzionale durante la creazione/modifica di un videogioco, esplicitare questo passo in un caso d'uso dettagliato (es. "Configurare achievement per un videogioco") e descrivere le condizioni di estensione; (3) aggiornare UC-003 per riflettere chiaramente quando e come l'admin passa dalla gestione videogiochi alla gestione achievement.

Vedi anche: TST-001

UC-004

REQ-008 — MEDIUM [93%] — Pagina 7 UC-004 non specifica la durata dell'abbonamento, il rinnovo, la scadenza e cosa accade ai benefici (sconti e giochi gratuiti) al termine della validità.

Inoltre, la postcondizione "i benefici ... sono applicati" è vaga: non è descritto come i giochi gratuiti diventano accessibili (entrano in libreria? sono solo giocabili finché l'abbonamento è attivo?) né come gli sconti si riflettano sul calcolo dei prezzi, mentre l'implementazione usa 'Abbonamento' con 'dataInizio', 'dataFine', 'stato', 'tipo' e strategie di prezzo.

> *Use Case 4: Gestire gli abbonamenti (UC-004) ... Steps 1. L'utente seleziona seleziona l'opzione per gestire gli abbonamenti dal menu utente (Mock-up 5) 2. Il sistema mostra lo stato attuale dell'abbonamento (nessuno, Gold o Silver) e chiede che tipo di abbonamento attivare. 3. L'utente seleziona un'opzione. 4. Il sistema verifica che il fondo dell'utente sia sufficiente per il costo dell'abbonamento. 5. Il sistema addebita il costo e attiva l'abbonamento Svolgimenti alternativi • 3a. Se 'e gi'a presente un abbonamento attivo l'utente 'e notificato con un messaggio. • 4a. Se il fondo 'e insufficiente Il sistema mostra un errore e propone di ricaricare il fondo (Test 28). Postconditions Lo stato dell'abbonamento 'e aggiornato nel database, i benefici (sconti e giochi gratuiti) sono applicati.*

Raccomandazione: Arricchire UC-004 con: (1) una descrizione esplicita della durata (es. 30 giorni) e delle regole di scadenza (cosa succede a 'stato' e 'abbonamentoAttivo'); (2) flussi per rinnovo, scadenza e disattivazione manuale (se prevista); (3) una specifica chiara dei benefici: come vengono determinati i giochi gratuiti, se e come compaiono nel catalogo/libreria, e come gli sconti si applicano al carrello (collegando al pattern 'PrezzoStrategy'); (4) postcondizioni separate per attivazione riuscita, fallimento per fondi insufficienti e presenza di abbonamento già attivo. Aggiungere test che verifichino il comportamento alla scadenza (ad esempio, che la strategia di prezzo torni a 'NonAbbonatoPrezzoStrategy').

Vedi anche: TST-002

Problemi Generali

REQ-009 — MEDIUM [100%] — Pagina 25 La sezione test descrive una buona copertura di flussi standard e di alcuni casi di fallimento, ma non esistono requisiti non funzionali espliciti (es. robustezza, gestione errori, sicurezza) a cui questi test siano tracciati. Inoltre, mancano test per scenari critici come errori di connessione al database, fallimenti nelle operazioni DAO durante transazioni logiche (es. acquisto), o concorrenza, che sono importanti per un sistema di vendita.

> *L'obiettivo principale dei test 'e stato quello di validare il flusso operativo standard, come l'aggiunta di un videogioco al carrello, l'esecuzione di un acquisto, o l'installazione di un videogioco nella libreria di un utente. Tuttavia, particolare attenzione 'e stata posta sui casi di fallimento e sui comportamenti anomali, come tentativi di autenticazione con credenziali errate, installazione di videogiochi non acquistati, o disinstallazione di videogiochi non installati.*

Raccomandazione: Aggiungere una sezione di requisiti non funzionali (ad esempio "Affidabilità" e "Gestione errori") che specifichi: (1) come il sistema deve comportarsi in caso di errori di persistenza (messaggi, rollback, nessuna perdita di denaro); (2) requisiti minimi di robustezza (nessun crash della CLI, gestione controllata delle eccezioni); (3) eventuali requisiti di sicurezza (protezione credenziali, accesso admin). Successivamente, estendere la suite di test con casi che simulino errori nei DAO (ad esempio usando mock o forzando eccezioni) e verifichino che i controller gestiscano correttamente tali errori in linea con i nuovi requisiti.

7.3 Testing (10 problemi)

UC-003

TST-001 — HIGH [85%] — Pagina 7 Per UC-003 è previsto un flusso alternativo critico (2a: dati non validi in creazione/modifica) ma nei test di CatalogoControllerTest e AchievementControllerTest vengono verificati solo casi di duplicazione o admin non autenticato, non la validazione dei dati di input (titolo vuoto, prezzo negativo, data non valida, ecc.).

> *Use Case 3: Gestire il catalogo come amministratore (UC-003) ... • 2a. Se i dati immensi dall'admin durante la creazione o la modifica non sono validi riceve messaggio di errore*

Raccomandazione: Estendere CatalogoControllerTest e/o AchievementControllerTest con casi come @Test void testAggiungiVideogioco_Fallimento_DatiNonValidi() che provi ad aggiungere un videogioco con prezzo negativo o titolo vuoto e verifichi che venga lanciata l'eccezione attesa, e un test analogo per la modifica (es. testModificaVideogioco_Fallimento_DatiNonValidi()).
Vedi anche: REQ-003, REQ-007, REQ-010

UC-004

TST-002 — HIGH [92%] — Pagina 7 Per UC-004 esiste un flusso alternativo critico (3a: abbonamento già attivo) che non è coperto da alcun test esplicito. I test su UtenteController coprono solo il caso di fondi insufficienti e di attivazione corretta, ma non verificano il comportamento quando l'utente ha già un abbonamento attivo.

> *Use Case 4: Gestire gli abbonamenti (UC-004) ... • 3a. Se 'e gi'a presente un abbonamento attivo l'utente 'e notificato con un messaggio. • 4a. Se il fondo 'e insufficiente Il sistema mostra un errore e propone di ricaricare il fondo (Test 28).*

Raccomandazione: Aggiungere un test in UtenteControllerTest, ad esempio @Test void testAttivaAbbonamento_Fallimento_AbbonamentoGiaAttivo(), che: (1) crea o recupera un utente con abbonamento già attivo, (2) tenta di attivare un nuovo abbonamento, (3) verifica che venga lanciata l'eccezione prevista o restituito un errore coerente con il flusso 3a di UC-004 e che il fondo e lo stato dell'abbonamento non vengano modificati.
Vedi anche: REQ-008

UC-002

TST-003 — MEDIUM [89%] — Pagina 6 Il flusso 3b di UC-002 (avvio videogioco e aggiornamento tempo di gioco) non è coperto da un test end-to-end sulla Business Logic. Esistono test unitari su StatisticheController (aggiornamento e lettura statistiche) ma nessun test che verifichi che LibreriaController.avviaVideogioco invochi correttamente StatisticheController.aggiornaTempoGioco e che il tempo venga effettivamente aggiornato.

> *Use Case 2: Gestire la libreria personale (UC-002) ... • 3b. L'utente seleziona l'opzione di avviare un videogioco e giocarlo – 3b.1 L'utente seleziona quanto tempo vuole giocare al videogioco. – 3b.2 L'utente simula di giocare a un videogioco e viene aggiornato il tempo di gioco.*

Raccomandazione: Aggiungere un test in LibreriaControllerTest, ad esempio @Test void testAvviaVideogioco_AggiornaStatistiche(), che: (1) prepara un utente con un gioco acquistato e installato, (2) invoca avviaVideogioco con un tempo specifico (es. 600 secondi), (3) usa StatisticheController.getStatistiche per verificare che il tempo di gioco sia aumentato del valore atteso.
Vedi anche: REQ-002, REQ-006

TST-005 — MEDIUM [91%] — Pagina 6 La tracciabilità tra UC-002 e i test è parzialmente incoerente: il template riferisce "(Test 32)" e "(Test 33)", ma nel documento Test 32 è StatistichControllerTest e Test 33 è AchievementControllerTest, che non verificano il flusso completo di visualizzazione da LibreriaController. Inoltre, non esiste un test che verifichi il comportamento di LibreriaController quando non ci sono achievement associati (3c.2) a livello di Business Logic.

> *Use Case 2: Gestire la libreria personale (UC-002) ... • 3c. L'utente seleziona l'opzione di visualizzare statistiche e achievement relativi ad un videogioco (Test 32) – 3c.1 Il sistema mostra le statistiche associate al gioco. – 3c.2 se non ci sono achievement associati ad un gioco il sistema manda un messaggio. (Test 33)*

Raccomandazione: 1) Correggere la tracciabilità nel template UC-002 specificando chiaramente quali metodi e classi di test coprono 3c.1 e 3c.2. 2) Aggiungere in LibreriaControllerTest un test come @Test void testVisualizzaStatisticheEAchievement_NessunAchievement(), che: (a) prepara un videogioco senza achievement, (b) invoca il metodo di LibreriaController che recupera statistiche e achievement, (c) verifica che la lista di achievement sia vuota e che venga gestito correttamente il caso (messaggio o eccezione).

Vedi anche: REQ-002, REQ-006

TST-007 — MEDIUM [79%] — Pagina 6 Per UC-002 il flusso alternativo 3a (libreria vuota) non è coperto dai test. LibreriaControllerTest verifica installazione/disinstallazione, gioco non installato e notifica di acquisto, ma non il comportamento quando la libreria è vuota e l'utente tenta di eseguire azioni.

> *Use Case 2: Gestire la libreria personale (UC-002) ... • 3a. Se la libreria 'e vuota l'utente riceve un messaggio per avvertirlo di non poter compiere azioni.*

Raccomandazione: Aggiungere un test in LibreriaControllerTest, ad esempio @Test void testInstallaVideogioco_Fallimento_LibreriaVuota(), che: (1) inizializza una Libreria vuota per un utente, (2) tenta di installare o avviare un videogioco, (3) verifica che venga lanciata l'eccezione prevista o che venga restituito un messaggio coerente con il flusso 3a, senza modificare lo stato della libreria.

Vedi anche: REQ-002, REQ-006

UC-001

TST-004 — MEDIUM [83%] — Pagina 6 Il flusso alternativo 6a di UC-001 (accesso a carrello vuoto) non è coperto dai test. I test di CarrelloControllerTest verificano aggiunta, rimozione, acquisto con fondi insufficienti e gioco già in libreria, ma non il comportamento quando l'utente tenta di acquistare o visualizzare un carrello vuoto.

> *Use Case 1: Acquistare un videogioco (UC-001) ... • 6a. Se l'utente accede ad un carrello vuoto riceve un messaggio e torna al catalogo*

Raccomandazione: Aggiungere un test in CarrelloControllerTest, ad esempio @Test void testEseguiAcquisto_Fallimento_CarrelloVuoto(), che: (1) recupera il carrello di un utente senza videogiochi, (2) invoca eseguiAcquisto, (3) verifica che venga lanciata un'eccezione o restituito false secondo la specifica, e che nessuna modifica venga applicata al fondo o alla libreria.

Vedi anche: ARCH-002, REQ-001, REQ-004, REQ-005

TST-006 — MEDIUM [88%] — Pagina 6 Nel template UC-001 il flusso 6b (fondo insufficiente) è tracciato a "Test 28", ma il Test 28 effettivo nel documento è riferito a UtenteControllerTest (attivazione abbonamento) e non a CarrelloControllerTest. Esiste un test di acquisto con fondi insufficienti, ma l'eccezione attesa nel test non è coerente con l'implementazione del metodo eseguiAcquisto.

> *Use Case 1: Acquistare un videogioco (UC-001) ... • 6b. Se l'utente ha un fondo insufficiente viene notificato e non pu'o completare l'acquisto (Test 28)*

Raccomandazione: Allineare test e implementazione: (1) in CarrelloControllerTest, nel metodo testEseguiAcquisto_Fallimento_FondoInsufficiente(), aggiornare l'assertThrows per aspettarsi IllegalStateException (coerente con eseguiAcquisto) oppure (2) modificare eseguiAcquisto per lanciare IllegalArgumentException come indicato nel test, e aggiornare la descrizione del test per riflettere il tipo di eccezione corretto. Inoltre, aggiornare il riferimento "(Test 28)" nel template UC-001 per puntare al test corretto di CarrelloControllerTest o rinumerare i test in modo coerente.

Vedi anche: ARCH-002, REQ-001, REQ-004, REQ-005

TST-008 — MEDIUM [100%] — Pagina 25 La strategia di test si basa su uno stato di database predefinito, ma nel documento non è presente una tabella di tracciabilità che colleghi in modo sistematico ogni UC ai test corrispondenti. I riferimenti ai numeri di test nei template (es. Test 28, 29, 32, 33) non sono allineati con la numerazione effettiva dei test nelle sezioni 4.1–4.3, rendendo difficile verificare la copertura rispetto ai requisiti.

> *I test sono stati implementati utilizzando il framework JUnit 5, con l'inizializzazione del database tramite script SQL (reset.sql, ProgettoSWE.sql, InserimentoProgettoSWE.sql),che ha permesso di caricare dati predefiniti per simulare scenari realistici senza creare nuovi oggetti, in linea con i requisiti della piattaforma.*

Raccomandazione: Introdurre una matrice di tracciabilità requisiti/Use Case test case. Ad esempio, creare una tabella che per ogni UC (UC-001..UC-004) elenchi: (1) i metodi di Business Logic coinvolti, (2) i nomi dei metodi di test JUnit (es. testEseguiAcquisto_Successo, testEseguiAcquisto_Fallimento_FondoInsufficiente), (3) eventuali flussi alternativi coperti. Aggiornare i riferimenti "Test 28", "Test 29", ecc. nei template per puntare ai nomi reali dei metodi di test o a un ID univoco di test definito nella matrice.

Vedi anche: ARCH-002, REQ-001, REQ-004, REQ-005

Problemi Generali

TST-009 — LOW [100%] — Pagina 25 La strategia di test copre bene i flussi standard e molti casi di errore, ma non sono presenti test di concorrenza o di consistenza transazionale sul database (ad esempio, cosa succede se due operazioni di acquisto o ricarica fondo avvengono in parallelo). Dato l'uso di un DB reale, questi aspetti potrebbero essere rilevanti per la robustezza, anche se non richiesti esplicitamente.

> *L'obiettivo principale dei test 'e stato quello di validare il flusso operativo standard, come l'aggiunta di un videogioco al carrello, l'esecuzione di un acquisto, o l'installazione di un videogioco nella libreria di un utente. Tuttavia, particolare attenzione 'e stata posta sui casi di fallimento e sui comportamenti anomali, come tentativi di autenticazione con credenziali errate, installazione di videogiochi non acquistati, o disinstallazione di videogiochi non installati.*

Raccomandazione: Se il corso lo consente, aggiungere almeno un test che simuli due operazioni sequenziali potenzialmente conflittuali sullo stesso utente (es. due acquisti consecutivi che consumano quasi tutto il fondo) per verificare che il saldo non diventi negativo e che le operazioni siano applicate in modo consistente. In alternativa, documentare esplicitamente che la gestione della concorrenza non è coperta in questo progetto, per chiarire il perimetro dei test.

TST-010 — LOW [100%] — Pagina 34 Solo la classe Carrello del Domain Model è testata esplicitamente. Anche se molte classi di dominio sono semplici DTO, alcune (es. Libreria con gestione statoInstallazione, Abbonamento con stato e tipo) contengono logica non banale che potrebbe beneficiare di test unitari diretti, soprattutto per evitare regressioni su mapping e metodi helper.

Data la loro semplicit'a e il ruolo di contenitori di dati, si 'e scelto di limitare i test unitari a una singola classe del modello di dominio, la classe Carrello, che riveste un ruolo cen- trale nella gestione del calcolo del totale dei videogiochi acquistati.

Raccomandazione: Aggiungere almeno test mirati per le classi di dominio con logica: ad esempio, in LibreriaTest verificare i metodi addVideogioco, isVideogiocoInstallato e setVideogiocoInstallato in presenza di più giochi; in AbbonamentoTest verificare la corretta gestione di stato e tipo (Gold/Silver) e l'eventuale metodo isAttivo(). Questo riduce il rischio di errori silenziosi nella logica di supporto usata dai controller.

8 Raccomandazioni Prioritarie

Le seguenti azioni sono considerate prioritarie:

1. **REQ-001** (pag. 14): Introdurre un requisito funzionale esplicito che definisca l'acquisto come operazione atomica: o tutte le modifiche (fondo, libreria, carrello) vanno ...
2. **REQ-002** (pag. 16): Estendere UC-002 con flussi alternativi specifici per l'avvio del videogioco: (1) aggiungere un ramo "3b.
3. **TST-001** (pag. 7): Estendere CatalogoControllerTest e/o AchievementControllerTest con casi come @Test void testAggiungiVideogioco_Fallimento_DatiNonValidi() che provi ad...
4. **TST-002** (pag. 7): Aggiungere un test in UtenteControllerTest, ad esempio @Test void testAttivaAbbonamento_Fallimento_AbbonamentoGiaAttivo(), che: (1) crea o recupera un...

9 Matrice di Tracciabilità

Su 24 casi d'uso tracciati: 24 completamente coperti, 0 senza design, 0 senza test.

ID	Caso d'Uso	Design	Test	Gap
UC-001	Acquistare un videogioco	✓	✓	—
UC-002	Gestire la libreria personale	✓	✓	—
UC-003	Gestire il catalogo come amministra...	✓	✓	—
UC-004	Gestire gli abbonamenti	✓	✓	—
UC-Login-Utente	Login utente	✓	✓	—
UC-Registrazione-Utente	Registrazione utente	✓	✓	—
UC-Visualizza-Catalogo	Visualizza catalogo videogiochi	✓	✓	—
UC-Ricerca-Giochi	Ricerca giochi nel catalogo	✓	✓	—
UC-Aggiungi-Carrello	Aggiungi videogioco al carrello	✓	✓	—
UC-Visualizza-Carrello	Visualizza carrello	✓	✓	—
UC-Rimuovi-Dal-Carrello	Rimuovi videogioco dal carrello	✓	✓	—
UC-Acquista-Giochi	Acquista giochi dal carrello	✓	✓	—
UC-Visualizza-Libreria	Visualizza libreria utente	✓	✓	—
UC-Installa-Videogioco	Installa videogioco dalla libreria	✓	✓	—
UC-Disinstalla-Videogioco	Disinstalla videogioco dalla librer...	✓	✓	—
UC-Avvia-Videogioco	Avvia videogioco e gioca	✓	✓	—
UC-Visualizza-Statistiche-Gioco	Visualizza statistiche di gioco	✓	✓	—
UC-Visualizza-Achievement	Visualizza achievement di un videog...	✓	✓	—
UC-Ricarica-Fondo	Ricarica fondo utente	✓	✓	—
UC-Sottoscrivi-Abbonamento	Sottoscrivi/attiva abbonamento (Gol...	✓	✓	—

ID	Caso d'Uso	Design	Test	Gap
UC-Gestione-Abbonamenti-Admin	Gestione abbonamenti lato admin (im...	✓	✓	—
UC-CRUD-Videogiochi-Admin	CRUD Videogiochi (Admin)	✓	✓	—
UC-CRUD-Achievement-Admin	CRUD Achievement (Admin)	✓	✓	—
UC-Login-Admin	Login amministratore	✓	✓	—

10 Coerenza Terminologica

Rilevate **10** incoerenze terminologiche (1 maggiori, 9 minori).

Gruppo	Varianti trovate	Severità	Suggerimento
Use Case Diagram / diagramma degli use case	«Use Case Diagram» (indice, sez. 2.1); «diagramma degli use case» (testo descrittivo); «Diagramma ER: Rappresentazione della struttura delle tabelle create dal DBMS» usa invece «Diagramma ER» come etichetta italiana.	MINOR	Uniformare a «Use Case Diagram» oppure tradurre ovunque in «Diagramma dei casi d'uso», scegliendo una sola forma e mantenendola in indice, testo e didascalie.
Mockup / Mock-up / Mock-Ups	Titolo sezione: «Mock-Ups»; testo: «mockup»; elenco strumenti: «mock-up»; figure: «Mockup raffigurante...»; strumento: «draw.io: impiegato per la creazione dei mockup.»	MINOR	Usare coerentemente una sola forma: o «Mockup» (senza trattino) oppure «Mock-up», sia nei titoli di sezione, sia nelle figure, sia nel testo.
CLI / Interfaccia (CLI) / interfaccia a riga di comando	«Command Line Interface (CLI)»; «Interfaccia (CLI): L'interfaccia a riga di comando (CLI) rappresenta...»; «l'interfaccia utente è stata implementata come una Command Line Interface (CLI)».	MINOR	Uniformare il termine per l'interfaccia a riga di comando: usare sempre «CLI (Command Line Interface)» alla prima occorrenza e poi solo «CLI» oppure solo «interfaccia a riga di comando (CLI)».
DBMS / RDBMS / database PostgreSQL	«DBMS» (figura 11: «tabelle create dal DBMS»); «RDBMS (PostgreSQL)»; «database relazionale PostgreSQL»; «database PostgreSQL».	MINOR	Scegliere una sola forma per indicare il database relazionale: ad esempio «DBMS PostgreSQL» oppure «RDBMS PostgreSQL», e usarla in modo coerente in tutto il documento.

Gruppo	Varianti trovate	Severità	Suggerimento
ORM / DAO / livello di accesso ai dati	«ORM (Object-Relational Mapping): utilizzato per mappare gli oggetti...»; «pacchetto ORM»; «classi DAO (Data Access Object) gestiscono le operazioni CRUD»; «Test del Pacchetto ORM» che in realtà testa classi DAO JDBC.	MINOR	Uniformare il termine per il livello di accesso ai dati: usare sempre «ORM (Object-Relational Mapping)» oppure «DAO layer», chiarendo che nel progetto l'ORM è implementato tramite DAO JDBC.
fondo / saldo	Descrizione: «un utente ha un fondo, che può ricaricare»; altrove: «ricarica del saldo»; «ricarica del fondo»; attributo DB: «fondo».	MINOR	Usare un solo termine per indicare il saldo economico dell'utente: preferibilmente «fondo» (già usato in DB e codice) e sostituire eventuali altre varianti.
Tipo di eccezione per fondo insufficiente	Descrizione UtenteController: «Se il fondo è insufficiente viene notificato...»; snippet 3: lancia IllegalStateException; testAttivaAbbonamento_Fallimento_FondiInsufficienti: si aspetta IllegalStateException; descrizione CarrelloControllerTest: «Si aspetta che venga lanciata un'IllegalArgumentException»; snippet 30 per eseguiAcquisto_Fallimento_FondoInsufficiente: assertThrows(IllegalArgumentException.class...).	MAJOR	Uniformare il tipo di eccezione lanciata in caso di fondo insufficiente, sia nel codice sia nei test, per evitare ambiguità: ad esempio usare sempre IllegalStateException e aggiornare i test di conseguenza.
Admin / amministratore / admin	Use case: «Admin» come attore; descrizione UC-003: «Consente all'amministratore di...»; passi: «L'admin ha accesso al menu da amministratore»; testo introduttivo: «ruolo amministratore (Admin)»; codice: classe Admin, AdminController.	MINOR	Uniformare il termine per indicare l'amministratore: usare sempre «Admin» (coerente con classi e use case) oppure sempre «amministratore», ma non alternarli senza criterio.
Utente / user	Diagramma use case: attore «Utente (User)»; resto del documento: sempre «Utente»; indice figure: «Use Case Diagram - Utente».	MINOR	Uniformare il termine per indicare l'utente del sistema: usare sempre «Utente» (con iniziale maiuscola quando è un attore/classe) e non alternare con «user» o altre forme.

Gruppo	Varianti trovate	Severità	Suggerimento
Business Logic Test / Domain Model Test / Test del Pacchetto ORM	Sezione 4.1: «Business Logic Test»; sezione 4.2: «Domain Model Test»; sezione 4.3: «Test del Pacchetto ORM»; testo: «I test in UtenteControllerTest verificano...» (inglese/italiano misto).	MINOR	Uniformare la denominazione dei test: usare sempre la forma italiana o sempre quella inglese, ad esempio «Test della Business Logic» e «Test del Domain Model», oppure «Business Logic Tests» e «Domain Model Tests».