Software Engineering Report
Validation Analysis

Library Management Application

Automated Validation System

December 2, 2025

Contents

1 Executive Summary

1.1 Scope of the Validation

This report presents an automated validation analysis of the document *"Library Management Application"*, an elaborato di Ingegneria del Software describing the design and implementation of a library management system.

The analysis follows these phases:

1. Extraction of structure and sections from the PDF.
2. Classification and extraction of requirements, use cases, architecture, and tests.
3. Consolidation of extracted content into a coherent model.
4. Traceability analysis between requirements, use cases, architecture components, and tests.
5. Feature-based validation against universal software engineering best practices.
6. Generation of this LaTeX-based validation report.

1.2 Validation Status

Table 1: Overall Validation Metrics

Metric	**Value**
Total Requirements Extracted	18
Total Use Cases Extracted	13 (13 explicit, 0 implicit)
Total Architecture Components	16
Total Tests Extracted	12 logical test suites (with multiple test methods each)
Requirements Coverage	100% (18/18 linked to at least one use case)
Use Case Coverage by Architecture	100% (13/13 mapped to components)
Use Case Coverage by Tests	100% (13/13 have at least one associated test suite)
Feature-Based Coverage (vs. generic SE KB)	82.0% (approx. 41/50 features covered)
Overall Status	**PASSED (with minor improvement opportunities)**

1.3 Key Findings

- **Strengths**
  - Rich and precise use case documentation (13 explicit use cases with pre-conditions, post-conditions, normal and alternative flows).
  - Clear architectural organization using MVC, Service Layer, DAO, Singleton, and a relational database schema.
  - Good structural test coverage: DAO and Service layers are tested with JUnit 5, Mockito, and Maven.
  - Database schema is fully specified with SQL DDL, enabling unambiguous persistence design.
- **Areas for Improvement**

- No explicit non-functional requirements (performance, security, usability, reliability) are stated as requirements.
- Requirements are mostly implicit in use cases; a dedicated requirements section with identifiers is missing.
- Test section focuses on structural tests; there is no explicit mapping to coverage metrics (e.g., code coverage, boundary conditions).
- Error handling and robustness are well covered in use case alternative flows but not summarized as cross-cutting requirements.

2 Content Extraction Results

2.1 Phase 1: Table of Contents and Sections

2.1.1 Extracted Table of Contents

From the provided PDF content, the following logical sections and page ranges were identified (page numbers approximate, based on the document excerpts):

Table 2: Extracted Table of Contents

Section	Start Page	End Page
1 Introduzione	4	4
1.1 Obiettivi del progetto	4	4
1.2 Tecnologie e architettura utilizzate	4	4
2 Progettazione	5	20
2.1 Attori coinvolti e relativi casi d'uso	5	5
2.2 Use Case Template	6	12
2.2.1 Use Case condivisi	6	7
2.2.2 Utente	8	10
2.2.3 Amministratore	10	12
2.3 Diagramma delle classi	13	16
2.4 Aspetti rilevanti del progetto	17	18
2.4.1 Model View Controller	17	17
2.4.2 Singleton	17	17
2.4.3 DAO	18	18
2.5 Page Navigation Diagram	19	19
2.6 Entity Relationship Diagram	20	20
3 Implementazione delle classi	21	37
3.1 DomainModel	21	23
3.2 Controller	24	28
3.3 Orm	29	31
3.4 Service	31	32
3.5 View	32	35
3.6 HelloApplication	36	36
3.7 Struttura del database	36	37
4 Testing	38	44

Continued on next page

Table 2 – continued

Section	Start Page	End Page
4.1–4.13 Test suites and Maven results	39	44

2.1.2 Section Classification

Each section was classified into one or more categories: Requirements, Use Cases, Architecture, Test, Mockups.

Table 3: Section Classification

Section	Labels
1.1 Obiettivi del progetto	Requirements
1.2 Tecnologie e architettura utilizzate	Architecture
2.1 Attori coinvolti e relativi casi d'uso	Use Cases
2.2 Use Case Template (all subsections)	Use Cases, Requirements
2.3 Diagramma delle classi	Architecture
2.4 Aspetti rilevanti del progetto (MVC, Singleton, DAO)	Architecture
2.5 Page Navigation Diagram	Architecture, Mockups
2.6 Entity Relationship Diagram	Architecture
3.1 DomainModel	Architecture
3.2 Controller	Architecture
3.3 Orm	Architecture
3.4 Service	Architecture
3.5 View	Architecture, Mockups
3.6 HelloApplication	Architecture
3.7 Struttura del database	Architecture
4 Testing (4.1–4.13)	Test

2.2 Requirements Extraction

The document does not contain a dedicated requirements chapter; requirements are mostly implicit in the project objectives and in the use case descriptions (pre-conditions, post-conditions, and flows). From these, 18 atomic requirements were extracted.

2.2.1 Requirements Quality Distribution

Table 4: Requirements Quality Assessment

Quality Level	Count	Percentage
Well-defined	12	66.7%
Needs Detail	6	33.3%
Vague/Unquantified	0	0.0%
Total	**18**	**100%**

2.2.2 Requirements by Type

Table 5: Requirements Classification

Type	Count	Percentage
Functional	16	88.9%
Non-Functional	0	0.0%
Constraint	2	11.1%
Goal/Background	0	0.0%
Total	**18**	**100%**

2.2.3 Detailed Requirements List

Table 6: Extracted Requirements

ID	Type	Description	Quality
REQ-1	Functional	The system must allow users to borrow and return books, digital media, and periodicals.	Needs Detail
REQ-2	Functional	The system must allow administrators to add, modify, and remove any type of element from the library catalog.	Well-defined
REQ-3	Constraint	The system must persist data using a remote PostgreSQL database accessed via JDBC.	Well-defined
REQ-4	Functional	The system must provide a login mechanism where users authenticate using email and password.	Well-defined
REQ-5	Functional	The system must provide a signup mechanism that creates a personal account for a new user.	Needs Detail
REQ-6	Functional	The system must validate email format, password strength, mandatory name and surname, and optional phone number during signup.	Well-defined
REQ-7	Functional	The system must allow authenticated users to view the catalog of elements in a tabular form.	Well-defined

Continued on next page

Table 6 – continued from previous page

ID	Type	Description	Quality
REQ-8	Functional	The system must allow users to search the catalog using filters to obtain a list of elements with desired characteristics.	Needs Detail
REQ-9	Functional	The system must allow users to view detailed information about a selected element.	Well-defined
REQ-10	Functional	The system must allow a user to request an element on loan, subject to availability and existing loans.	Well-defined
REQ-11	Functional	The system must allow a user to return one or more borrowed elements, making them available again for loan.	Well-defined
REQ-12	Functional	The system must allow a user to view the list of elements currently borrowed by that user.	Well-defined
REQ-13	Functional	The system must allow an administrator to add a new element to the database, including validation of required fields and ISBN uniqueness for books.	Needs Detail
REQ-14	Functional	The system must allow an administrator to remove an element from the database, preventing removal if the element is currently on loan.	Well-defined
REQ-15	Functional	The system must allow an administrator to modify an existing element, including validation of fields and ISBN uniqueness for books.	Needs Detail
REQ-16	Functional	The system must allow an administrator to add one or more genres to the database, avoiding duplicates by name.	Well-defined
REQ-17	Functional	The system must allow associating and removing genres from elements.	Needs Detail
REQ-18	Constraint	The system must maintain a single shared instance of the database connection and of the main controller using the Singleton pattern.	Needs Detail

Notes on Requirements Quality

- Several requirements (e.g., REQ-1, REQ-5, REQ-8, REQ-13, REQ-15, REQ-17, REQ-18) could be made more testable by adding explicit constraints (e.g., maximum number of concurrent loans, password policy metrics, maximum catalog size, performance expectations).
- No explicit non-functional requirements (performance, security, usability, reliability) are stated, although some quality aspects are implicitly addressed via validation rules in use cases.

2.3 Use Case Extraction

The report provides a detailed use case template section with 13 explicit use cases.

2.3.1 Use Case Summary

Table 7: Extracted Use Cases

ID	Name	Type	Actors
UC-1	Login	Explicit	Utente, Amministratore, Sistema
UC-2	Visualizza dettagli elemento	Explicit	Utente, Amministratore, Sistema
UC-3	Logout	Explicit	Utente, Amministratore, Sistema
UC-4	Signup	Explicit	Utente, Sistema
UC-5	Ricerca nel catalogo	Explicit	Utente, Amministratore, Sistema
UC-6	Visualizza catalogo	Explicit	Utente, Amministratore, Sistema
UC-7	Richiesta di un elemento in prestito	Explicit	Utente, Sistema
UC-8	Restituzione di un elemento preso in prestito	Explicit	Utente, Sistema
UC-9	Visualizza elementi presi in prestito	Explicit	Utente, Sistema
UC-10	Aggiungere elemento al database	Explicit	Amministratore, Sistema
UC-11	Rimuovere elemento	Explicit	Amministratore, Sistema
UC-12	Modifica un elemento nel database	Explicit	Amministratore, Sistema
UC-13	Aggiunta di un genere	Explicit	Amministratore, Sistema

2.3.2 Detailed Use Case Descriptions (Selected)

UC-1: Login

- **Actors:** Utente, Amministratore, Sistema
- **Pre-conditions:** Disporre di un account; essere connessi al database.
- **Post-conditions:** L'utente esegue l'accesso correttamente oppure viene segnalato errore al login.
- **Main Flow:**
  1. Utente inserisce e-mail e password.
  2. Utente preme il tasto Login.
  3. Login avviene con successo.
- **Alternative Flows:**
  - 3A. Se identificativo non nel database, compare scritta "E-mail non presente".
  - 3B. Se identificativo esiste ma password sbagliata, compare scritta "Password errata".
  - 3C. Se non c'è connessione al database, compare scritta "Impossibile connettersi con il database".

UC-4: Signup

- **Actors:** Utente, Sistema
- **Pre-conditions:** Disporre di una e-mail; essere connessi al database.
- **Post-conditions:** L'utente crea un account correttamente oppure viene segnalato errore al signup.
- **Main Flow:**
  1. Utente inserisce e-mail, password, nome e cognome.
  2. Utente preme il tasto Signup.
  3. Signup avviene con successo.
- **Alternative Flows:** 3A–3G covering invalid email, weak password, missing name/surname, invalid phone, email already in use, and database connection failure.

UC-7: Richiesta di un elemento in prestito

- **Actors:** Utente, Sistema
- **Pre-conditions:** Disporre di un account; essere connessi al database; aver effettuato il login.
- **Post-conditions:** L'elemento viene dato in prestito all'utente oppure viene segnalato un errore.
- **Main Flow:**
  1. Utente visualizza un elemento.
  2. Utente preme il tasto per il prestito.
  3. Utente riceve l'elemento in prestito.
- **Alternative Flows:**
  - 3A. Se l'elemento è già stato preso in prestito dall'utente, compare la scritta "Elemento già preso in prestito".
  - 3B. Se l'elemento non è disponibile, compare la scritta "Elemento già preso in prestito" (testo probabilmente intende "Elemento non disponibile").
  - 3C. Se non c'è connessione al database, compare la scritta "Impossibile connettersi al database.".

UC-10: Aggiungere elemento al database

- **Actors:** Amministratore, Sistema
- **Pre-conditions:** Avere effettuato il login; essere connessi al database.
- **Post-conditions:** L'elemento viene aggiunto nel database correttamente oppure viene segnalato un errore.
- **Main Flow:**
  1. Amministratore clicca sul bottone "Aggiungi elemento".
  2. Amministratore compila il form con le informazioni necessarie.
  3. Amministratore clicca su "Save".
  4. L'elemento viene aggiunto al database.
- **Alternative Flows:** 3A (Cancel), 4A (informazioni insufficienti o non valide), 4B (ISBN già presente), 4C (impossibile connettersi al database).

2.4 Architecture Extraction

2.4.1 Architectural Pattern

The architecture is explicitly described as:

- **MVC (Model-View-Controller)** for separation of concerns between data, UI, and control logic.
- **Service Layer** between controllers and data access.
- **DAO (Data Access Object)** for persistence abstraction.
- **Singleton** for connection management and main controller.

Overall pattern: **Layered MVC Architecture with Service and DAO layers**.

2.4.2 Architecture Components

Table 8: Architecture Components Analysis

Component	Responsibility	Design Notes
DomainModel (Element, Book, DigitalMedia, PeriodicPublication, Genre, User)	Represents domain entities of the library (items, genres, users) with attributes and relationships.	Clear domain modeling; inheritance used for different element types; User includes isAdmin flag.
Controller package (various ViewControllers)	Handles GUI events, orchestrates user interactions, and delegates to services.	Follows MVC; controllers extend BaseViewController for shared behavior.
BaseViewController	Provides common controller methods and navigation support via lastView.	Promotes reuse; centralizes navigation logic.
AddItemViewController	Manages UI for adding items (books, periodicals, digital media) with validation and type-specific fields.	Good separation of UI and validation; extends ElementCheckViewController.
BorrowedItemsViewController	Displays items borrowed by a user, interacting with MainService and LibraryUserService.	Clear responsibility; interacts only with services, not DAOs.
ElementCheckViewController	Validates text input fields according to predefined filters before acceptance.	Encapsulates input validation; improves consistency and maintainability.
HomeViewController	Manages the main screen, showing catalog in a table and providing filters for search.	Central UI for browsing; aligns with UC-5 and UC-6.

Continued on next page

Table 8 – continued from previous page

Component	Responsibility	Design Notes
LoginViewController	Manages login screen and authentication flow.	Uses BaseViewController for error messages; aligns with UC-1.
MenuBarViewControlle	Manages menu bar, options depending on user type, navigation, genre management, and logout.	Implements role-based UI; central navigation hub.
SignupViewController	Manages signup screen, validates inputs, and logs user in on success.	Aligns with UC-4; encapsulates validation logic.
Orm / ConnectionManager (Singleton)	Manages a single PostgreSQL database connection; provides configuration and validation methods.	Correct use of Singleton; centralizes DB access; potential single point of failure.
Orm / ElementDAO and specialized DAOs (BookDAO, DigitalMediaDAO, PeriodicPublicationDAO, BorrowsDAO, GenreDAO, UserDAO)	Encapsulate CRUD operations and queries for elements, books, digital media, periodicals, borrows, genres, and users.	Clear DAO responsibilities; use ConnectionManager; some naming inconsistencies (Manager vs DAO) in text.
Service / MainService (Singleton)	Core service that manages data access, tracks user state, and provides services to the rest of the application.	Combines state management and data access; could be decomposed further.
Service / UserService	Manages user authentication, registration, logout, and element search/filtering.	Implements UC-1, UC-4, UC-5, UC-6; central user-related business logic.
Service / LibraryUserService	Extends UserService to handle user operations such as loan requests and returns.	Implements UC-7, UC-8, UC-9; good specialization.
Service / LibraryAdminService	Extends UserService to handle administrative operations on elements and genres.	Implements UC-10, UC-11, UC-12, UC-13; clear admin responsibility.
View (JavaFX FXML views)	GUI screens: Signup-view, AddItem-view, ElementDetails-view, Login-view, Home-view, Borrowed-view.	Concrete MVC views; screenshots provided; no behavior.

Continued on next page

Table 8 – continued from previous page

Component	Responsibility	Design Notes
HelloApplication	Main JavaFX Application class; configures window, loads initial view, initializes singletons, and starts program.	Entry point; centralizes startup concerns.
Database (PostgreSQL schema)	Stores users, elements, books, digital media, periodic publications, borrows, genres, and element-genre associations.	Well-defined relational schema; foreign keys enforce integrity.

2.4.3 Architecture Analysis

Pattern: Layered MVC with Service and DAO layers.

Summary: The architecture exhibits a clear separation of concerns:

- Presentation: JavaFX views and controllers.
- Business logic: Service layer (UserService, LibraryUserService, LibraryAdminService, MainService).
- Persistence: DAO layer and PostgreSQL database.
- Infrastructure: ConnectionManager Singleton.

Strengths:

- Clear mapping between use cases and services/controllers.
- Proper use of design patterns (MVC, DAO, Singleton, Service Layer).
- Database schema is explicit and normalized for the domain.

Areas for Improvement:

- MainService has broad responsibilities (state, data access, coordination); could be decomposed.
- Error handling and transaction management strategies are not described at the architectural level.
- No explicit description of how concurrency or multiple simultaneous users are handled.

2.5 Test Extraction

2.5.1 Test Type Distribution

The Testing section lists 12 named test classes (suites). Each suite contains multiple test methods; the document summarizes representative tests for some DAOs.

Table 9: Test Suites by Type

Test Type	Suites	
Unit / Structural	12	BookDAOTest, BorrowsDAOTest, ConnectionManagerTest, DigitalMed
Integration	0 (explicit)	
System / End-to-End	0 (explicit)	
Performance	0	

2.5.2 Detailed Test List (Logical Level)

Table 10: Extracted Test Suites and Representative Methods

ID	Type	Artifact	Coverage Hint
TEST-1	Unit	BookDAOTest.addBookTest	Insertion of a book into the database (UC-10, UC-12 support)
TEST-2	Unit	BookDAOTest.updateBookTest	Update of an existing book (UC-12 support)
TEST-3	Unit	BookDAOTest.getBookByIsbnTest	Retrieval of a book by ISBN (UC-5, UC-6 support)
TEST-4	Unit	BorrowsDAOTest.addBorrowTest	Addition of a loan (UC-7 main flow)
TEST-5	Unit	BorrowsDAOTest.removeBorrowTest	Removal of a loan (UC-8 main flow)
TEST-6	Unit	BorrowsDAOTest.getBorrowedElementsForUserTest	Retrieval of borrowed elements for a user (UC-9 main flow)
TEST-7	Unit	ConnectionManagerTest.getConnection / isConnectionValid / closeConnection	Connection lifecycle and Singleton behavior (infrastructure constraint)
TEST-8	Unit	DigitalMediaDAOTest.addDigitalMediaTest	Insertion of digital media (UC-10, UC-12 support)
TEST-9	Unit	DigitalMediaDAOTest.updateDigitalMediaTest	Update of digital media (UC-12 support)
TEST-10	Unit	DigitalMediaDAOTest.getDigitalMediaTest	Retrieval of digital media (UC-5, UC-6 support)
TEST-11	Unit	ElementDAOTest.removeElement / getElement	Removal and retrieval of generic elements (UC-10, UC-11, UC-12 support)

Continued on next page

Table 10 – continued from previous page

ID	Type	Artifact	Coverage Hint
TEST-12	Unit	GenreDAOTest (associateGenreWithElementTest, addGenreTest, getAllGenresTest, getGenresForElementTest, removeGenreFromElementTest)	Genre management and associations (UC-10, UC-12, UC-13 support)
TEST-13	Unit	PeriodicPublicationDAOTest (add/update)	Management of periodic publications (UC-10, UC-12 support)
TEST-14	Unit	UserDAOTest	User CRUD and uniqueness checks (UC-1, UC-4 support)
TEST-15	Unit	MainServiceTest	Core service behavior and state management (multiple UCs support)
TEST-16	Unit	UserServiceTest	Implementation of UC-1, UC-4, UC-5, UC-6 including alternative flows
TEST-17	Unit	LibraryUserServiceTest	Implementation of UC-7, UC-8, UC-9 including alternative flows
TEST-18	Unit	LibraryAdminServiceTest	Implementation of UC-10, UC-11, UC-12, UC-13 including alternative flows

Observations

- Tests are primarily unit/structural, focusing on DAOs and services.
- The document explicitly states that tests verify correct interaction with the database and its usage.
- There is no explicit mention of integration or system-level tests, nor of performance or load tests.

3 Consolidated Architecture Model

3.1 Overall Pattern and Layers

Pattern: Layered MVC Architecture with Service and DAO layers, backed by a relational database and using Singleton for shared resources.

3.2 Layers and Components

Table 11: Consolidated Architecture Layers and Components

Layer	**Component**	**Responsibility**	**Communicates With**
Presentation	JavaFX Views (Signup-view, AddItem-view, ElementDetails-view, Login-view, Home-view, Borrowed-view)	Render UI screens and capture user input.	Corresponding View-Controllers.
Presentation	BaseViewController	Common controller behavior and navigation.	Other controllers, Main-Service.
Presentation	LoginViewController	Login screen logic.	UserService, BaseView-Controller.
Presentation	SignupViewController	Signup screen logic and validation.	UserService, Ele-mentCheckViewCon-troller, BaseViewCon-troller.
Presentation	HomeViewController	Home screen, catalog display, filters.	LibraryUserService, LibraryAdminService, MainService.
Presentation	BorrowedItemsViewController	Borrowed items list for a user.	LibraryUserService, MainService.
Presentation	AddItemViewController	Add/modify item UI.	LibraryAdminService, ElementCheckView-Controller.
Presentation	MenuBarViewController	Menu bar, navigation, logout, genre manage-ment.	MainService, UserSer-vice, LibraryAdminSer-vice.
Business / Service	MainService (Sin-gleton)	Core service, manages access to data and user state, provides services to controllers.	DAOs, UserService, LibraryUserService, LibraryAdminService.
Business / Service	UserService	Authentication, reg-istration, logout, search/filter, view catalog.	UserDAO, Element-DAO, GenreDAO, MainService.
Business / Service	LibraryUserService	User operations: re-quest loan, return, view borrowed items.	BorrowsDAO, Element-DAO, MainService.
Business / Service	LibraryAdminService	Admin operations: add, remove, mod-ify elements; manage genres.	ElementDAO, Book-DAO, DigitalMedi-aDAO, PeriodicPublica-tionDAO, GenreDAO, MainService.

Continued on next page

Table 11 – continued from previous page

Layer	Component	Responsibility	Communicates With
Persistence / DAO	ConnectionManager (Singleton)	Single PostgreSQL connection; configuration and validation.	All DAOs.
Persistence / DAO	UserDAO	CRUD and uniqueness checks for users.	ConnectionManager, users table.
Persistence / DAO	ElementDAO	CRUD for generic elements and complex associated data.	ConnectionManager, elements table, genres association.
Persistence / DAO	BookDAO	Book-specific operations (ISBN, author, etc.).	ConnectionManager, books table, elements table.
Persistence / DAO	DigitalMediaDAO	Digital media operations.	ConnectionManager, digitalmedias table, elements table.
Persistence / DAO	PeriodicPublicationDAO	Periodic publication operations.	ConnectionManager, periodicpublication table, elements table.
Persistence / DAO	BorrowsDAO	Loan operations and queries.	ConnectionManager, borrows table.
Persistence / DAO	GenreDAO	Genre CRUD and element-genre associations.	ConnectionManager, genres and elementgenres tables.
Infrastructure	PostgreSQL Database	Persistent storage for all domain data.	All DAOs.
Infrastructure	HelloApplication	Application entry point; initializes singletons and loads initial view.	JavaFX runtime, MainService, ConnectionManager.

3.3 Architecture Analysis Summary

Separation of Concerns: Good. Presentation, business logic, and persistence are clearly separated.

Coupling and Cohesion:

- Controllers are coupled to services, not DAOs, which is appropriate.
- Services are cohesive around user vs admin responsibilities, though MainService is somewhat broad.
- DAOs are cohesive per entity type.

Missing or Implicit Aspects:

- No explicit description of transaction boundaries or rollback strategies (though tests mention rollback).

- No explicit concurrency model or session management description.
- Error handling is described in use cases but not summarized as a cross-cutting architectural concern.

4 Traceability Matrix

4.1 Requirements to Use Cases Mapping

Table 12: Requirements to Use Cases Traceability

Req ID	Use Cases	Status	Rationale
REQ-1	UC-7, UC-8, UC-9	Covered	Borrowing and returning elements, and viewing borrowed items, are explicitly modeled by UC-7, UC-8, and UC-9.
REQ-2	UC-10, UC-11, UC-12	Covered	Admin operations to add, remove, and modify elements are covered by UC-10, UC-11, and UC-12.
REQ-3	All UCs (pre-condition)	Covered	Many use cases require being connected to the database as a pre-condition, implying PostgreSQL persistence.
REQ-4	UC-1	Covered	UC-1 describes login with email and password.
REQ-5	UC-4	Covered	UC-4 describes account creation (signup).
REQ-6	UC-4	Covered	UC-4 alternative flows specify validation rules for email, password, name, surname, and phone.
REQ-7	UC-6	Covered	UC-6 describes viewing the catalog in a table.
REQ-8	UC-5	Covered	UC-5 describes searching the catalog using filters.
REQ-9	UC-2	Covered	UC-2 describes viewing detailed information of a selected element.
REQ-10	UC-7	Covered	UC-7 describes requesting an element on loan with availability checks.
REQ-11	UC-8	Covered	UC-8 describes returning borrowed elements.
REQ-12	UC-9	Covered	UC-9 describes viewing currently borrowed elements.
REQ-13	UC-10	Covered	UC-10 describes adding an element with validation and ISBN uniqueness.
REQ-14	UC-11	Covered	UC-11 describes removing an element, preventing removal if on loan.

Continued on next page

Table 12 – continued from previous page

Req ID	Use Cases	Status	Rationale
REQ-15	UC-12	Covered	UC-12 describes modifying an element with validation and ISBN uniqueness.
REQ-16	UC-13	Covered	UC-13 describes adding genres and avoiding duplicates.
REQ-17	UC-10, UC-12, UC-13	Covered	Genre association is implied in element add/modify and explicit in UC-13.
REQ-18	All UCs (infrastructure)	Covered	Singleton ConnectionManager and MainService are used by all flows requiring DB access and state.

4.2 Use Cases to Architecture Mapping

Table 13: Use Cases to Architecture Traceability

UC ID	UC Name	Components	Status	Rationale
UC-1	Login	LoginViewController, UserService, UserDAO, ConnectionManager, users table	Covered	LoginViewController captures credentials, UserService authenticates via UserDAO and DB.
UC-2	Visualizza dettagli elemento	ElementDetailsViewController, LibraryUserService / LibraryAdminService, ElementDAO, ConnectionManager, elements table	Covered	Controller requests element details from services, which query ElementDAO.
UC-3	Logout	MenuBarViewController, UserService, MainService	Covered	MenuBarViewController triggers logout via UserService and updates state in MainService.
UC-4	Signup	SignupViewController, UserService, UserDAO, ConnectionManager, users table	Covered	SignupViewController validates input and calls UserService to create user via UserDAO.

Continued on next page

Table 13 – continued from previous page

UC ID	UC Name	Components	Status	Rationale
UC-5	Ricerca nel catalogo	HomeViewController, UserService, ElementDAO, GenreDAO, ConnectionManager	Covered	HomeViewController sends filter parameters to UserService, which queries DAOs.
UC-6	Visualizza catalogo	HomeViewController, UserService, ElementDAO, ConnectionManager	Covered	HomeViewController loads catalog via UserService and ElementDAO.
UC-7	Richiesta di un elemento in prestito	ElementDetailsViewController / HomeViewController, LibraryUserService, BorrowsDAO, ElementDAO, ConnectionManager	Covered	Controllers call LibraryUserService, which uses BorrowsDAO and ElementDAO to register loan and update availability.
UC-8	Restituzione di un elemento preso in prestito	BorrowedItemsViewController, LibraryUserService, BorrowsDAO, ElementDAO, ConnectionManager	Covered	BorrowedItemsViewController triggers return via LibraryUserService and DAOs.
UC-9	Visualizza elementi presi in prestito	BorrowedItemsViewController, LibraryUserService, BorrowsDAO, ConnectionManager	Covered	Controller requests list of loans from LibraryUserService and BorrowsDAO.
UC-10	Aggiungere elemento al database	AddItemViewController, LibraryAdminService, ElementDAO, BookDAO, DigitalMediaDAO, PeriodicPublicationDAO, GenreDAO, ConnectionManager	Covered	AddItemViewController collects data; LibraryAdminService validates and persists via DAOs.
UC-11	Rimuovere elemento	ElementDetailsViewController / HomeViewController, LibraryAdminService, ElementDAO, BorrowsDAO, ConnectionManager	Covered	Admin triggers removal; LibraryAdminService checks loans via BorrowsDAO and removes via ElementDAO.

Continued on next page

Table 13 – continued from previous page

UC ID	UC Name	Components	Status	Rationale
UC-12	Modifica un elemento nel database	AddItemViewController, LibraryAdminService, ElementDAO, BookDAO, DigitalMediaDAO, PeriodicPublicationDAO, GenreDAO, ConnectionManager	Covered	Similar to UC-10 but updating existing records.
UC-13	Aggiunta di un genere	MenuBarViewController / dedicated view, LibraryAdminService, GenreDAO, ConnectionManager	Covered	Admin uses UI to add genres; LibraryAdminService calls GenreDAO.

4.3 Use Cases to Tests Mapping

Table 14: Use Cases to Tests Traceability

UC ID	UC Name	Main Flow	Alt Flows	Status & Details
UC-1	Login	Tested	Tested	**Fully Covered**. UserServiceTest verifies correct implementation of UC-1, including alternative flows for invalid email, wrong password, and DB connection errors.
UC-2	Visualizza dettagli elemento	Tested	Tested (DB errors)	**Fully Covered**. ElementDAOTest and related service tests ensure retrieval of element details and error handling when DB is unavailable.
UC-3	Logout	Tested	N/A	**Fully Covered**. UserServiceTest and MainServiceTest verify logout behavior and state reset.
UC-4	Signup	Tested	Tested	**Fully Covered**. UserServiceTest covers signup, including all validation alternative flows described in UC-4.

Continued on next page

Table 14 – continued from previous page

UC ID	UC Name	Main Flow	Alt Flows	Status & Details
UC-5	Ricerca nel catalogo	Tested	Partially	**Fully Covered (functional)**. UserServiceTest and DAO tests cover search and retrieval; some UI-level variations (e.g., no results) are likely but not explicitly described in tests.
UC-6	Visualizza catalogo	Tested	Tested (DB errors)	**Fully Covered**. UserServiceTest and ElementDAOTest ensure catalog retrieval and DB error handling.
UC-7	Richiesta di un elemento in prestito	Tested	Tested	**Fully Covered**. LibraryUserServiceTest and BorrowsDAOTest verify loan creation and alternative flows (already borrowed, not available, DB errors).
UC-8	Restituzione di un elemento preso in prestito	Tested	Tested	**Fully Covered**. LibraryUserServiceTest and BorrowsDAOTest verify returns and alternative flows (element not borrowed, DB errors).
UC-9	Visualizza elementi presi in prestito	Tested	Tested	**Fully Covered**. LibraryUserServiceTest and BorrowsDAOTest verify retrieval of borrowed items and handling of empty lists and DB errors.
UC-10	Aggiungere elemento al database	Tested	Tested	**Fully Covered**. LibraryAdminServiceTest plus DAO tests (ElementDAO, BookDAO, DigitalMediaDAO, PeriodicPublicationDAO, GenreDAO) cover main and alternative flows (cancel, invalid data, duplicate ISBN, DB errors).
UC-11	Rimuovere elemento	Tested	Tested	**Fully Covered**. LibraryAdminServiceTest and ElementDAOTest verify removal and alternative flows (element on loan, DB errors).

Continued on next page

Table 14 – continued from previous page

UC ID	UC Name	Main Flow	Alt Flows	Status & Details
UC-12	Modifica un elemento nel database	Tested	Tested	**Fully Covered**. LibraryAdminServiceTest and DAOs verify update operations and alternative flows (cancel, invalid data, duplicate ISBN, DB errors).
UC-13	Aggiunta di un genere	Tested	Tested	**Fully Covered**. LibraryAdminServiceTest and GenreDAOTest verify genre addition and alternative flows (duplicate genre, DB errors).

4.4 Complete Traceability Overview

Table 15: Traceability Coverage Summary

Artifact Type	Total	Covered	Uncovered	Coverage %
Requirements	18	18	0	100%
Use Cases	13	13	0	100%
Components	16	16	0	100%
Test Suites	12	12	0	100%

4.5 Orphan Artifacts

No orphan requirements, use cases, or test suites were identified based on the information provided. All functional requirements are traceable to at least one use case, which in turn are mapped to architecture components and to at least one test suite.

5 Feature-Based Validation

5.1 Overview

The report was evaluated against a generic knowledge base of 50 universal software engineering features (problem definition, architecture, security, performance, maintainability, testing, project management, documentation, etc.).

Based on the content, approximately 41 of these features are covered to some extent, yielding an estimated coverage of 82%.

Table 16: Feature-Based Validation Summary

Metric	Value
Total Knowledge Base Features	50
Features Covered in Report (approx.)	41
Features Not Covered (approx.)	9
Coverage Percentage	**82.0%**

5.2 Selected Covered Features

Table 17: Examples of Covered Universal Features

ID	Feature	Category	Evidence from Report
F-1	Clear functional scope	Problem Definition	"Questo progetto implementa un sistema di gestione di una biblioteca che permette: agli utenti di prendere in prestito e restituire libri, media digitali e periodici; agli amministratori di aggiungere, modificare e rimuovere ognuno di questi elementi dal catalogo della biblioteca."
F-2	Technology stack explicitly documented	Architecture	Section 1.2 lists Java 23, IntelliJ, PostgreSQL, JDBC, JavaFX, Scene Builder, StarUML, JUnit 5, Mockito, Maven.
F-3	Use of layered architecture	Architecture	"Tra le architetture utilizzate abbiamo invece: MVC, Service Layer, DAO."
F-4	Domain modeling with UML	Architecture	Section 2.3 describes class diagrams for DomainModel, Controller, ORM, Service.
F-5	Database schema specified	Architecture	Section 3.7 provides full SQL DDL for users, elements, books, digitalmedias, periodicpublication, borrows, genres, elementgenres.
F-6	Detailed use case descriptions	Documentation	Section 2.2 provides use case templates with pre-conditions, post-conditions, normal and alternative flows.
F-7	Input validation rules specified	Maintainability / Quality	UC-4 (Signup) details validation for email, password, name, surname, phone.
F-8	Error handling scenarios documented	Reliability	Many use cases include alternative flows for database connection failures and invalid operations.
F-9	Unit testing with mocking	Testing	Section 4 describes JUnit 5 and Mockito-based tests for DAOs and services.
F-10	Build automation	Project Management	Section 1.2 mentions Maven for build automation and test execution.

5.3 Selected Uncovered or Weakly Covered Features

Table 18: Examples of Uncovered or Weakly Covered Features

ID	Feature	Category	Description
UF-1	Explicit performance requirements	Performance	No quantitative performance targets (e.g., response time, throughput) are specified.
UF-2	Security requirements and threat analysis	Security	No explicit security requirements (e.g., password hashing, encryption, access control policies) or threat analysis are documented.
UF-3	Scalability and concurrency considerations	Performance	No discussion of concurrent users, locking, or scalability strategies.
UF-4	Deployment architecture	Architecture	No description of deployment topology (e.g., single machine, client-server, cloud).
UF-5	Monitoring and logging strategy	Maintainability	No mention of logging, monitoring, or observability.
UF-6	Configuration management	Project Management	No description of configuration files, environment separation, or secrets management.
UF-7	Code quality metrics	Maintainability	No mention of static analysis, code coverage, or style guidelines.
UF-8	Usability evaluation	Documentation	UI screenshots are provided, but no usability goals or evaluation results are documented.
UF-9	Backup and recovery strategy	Reliability	No description of backup, restore, or disaster recovery procedures.

5.4 Checklist Compliance Example: Clear Functional Scope

Feature: Clear functional scope.

Description: The system's functional scope is clearly defined with main actors and capabilities.

Table 19: Checklist Compliance for "Clear functional scope"

ID	Checklist Item	Status	Explanation
1.1	Main actors are identified	True	Section 2.1 explicitly identifies "Amministratore" and "Utente" as actors, plus the system itself.
1.2	Core capabilities are described	True	Section 1.1 lists borrowing/returning items for users and catalog management for administrators.

Continued on next page

Table 19 – continued from previous page

ID	Checklist Item	Status	Explanation
1.3	Out-of-scope items are mentioned or implied	Partial	The scope is clear for core functions, but there is no explicit "Out of Scope" subsection; non-functional aspects are not scoped.
1.4	Functional scope is consistent across sections	True	Use cases, architecture, and tests consistently reflect the same set of functionalities.

6 Detailed Analysis

6.1 Requirements Quality Assessment

6.1.1 Strengths

- Functional requirements are well captured through detailed use cases with explicit pre-conditions, post-conditions, and alternative flows.
- Validation rules (e.g., for signup) are precisely specified, enabling straightforward test design.
- Error conditions (e.g., DB connection failures, invalid operations) are systematically described in alternative flows.

6.1.2 Gaps and Recommendations

- **Lack of explicit non-functional requirements:** Add a dedicated section specifying performance, security, usability, and reliability requirements with measurable criteria.
- **Implicit requirements:** Many requirements are only implicit in use cases. Introduce a requirements list with identifiers (FR-1, NFR-1, etc.) to improve traceability and maintainability.
- **Constraints:** Constraints such as use of PostgreSQL, JDBC, and Singleton patterns are described but not framed as explicit requirements; consider formalizing them.

6.2 Use Case Completeness Analysis

- All major user and admin operations are covered by explicit use cases.
- Alternative flows cover common error scenarios (invalid input, duplicates, DB connection issues).
- Some minor inconsistencies in wording (e.g., UC-7 alternative flow 3B message text) could be clarified but do not affect functional understanding.

6.3 Architecture Quality Evaluation

Table 20: Architecture Quality Metrics (Qualitative)

Principle	Status	Notes
Separation of Concerns	Good	MVC + Service + DAO layers are clearly separated.
Loose Coupling	Good	Controllers depend on services, not DAOs; DAOs depend on Connectio
High Cohesion	Good	DAOs and services are cohesive around entity or role responsibili
Single Responsibility	Moderate	MainService has broad responsibilities; could be decomposed.
Extensibility	Good	New element types or genres can be added via DomainModel and I

6.4 Test Coverage Analysis

- **Strengths**
  - DAOs are thoroughly tested for CRUD operations and key queries.
  - Service tests explicitly claim to cover use cases and alternative flows.
  - Use of Mockito allows isolation from real database in some tests.
  - Maven is used to automate test execution and reporting.
- **Areas for Improvement**
  - No explicit code coverage metrics (e.g., line or branch coverage) are reported.
  - No integration or end-to-end tests are described; all tests are structural/unit-level.
  - No performance or stress tests are mentioned.

7 Recommendations

7.1 Priority 1: Critical Items

1. **Introduce explicit non-functional requirements**
   - Add a section specifying performance (e.g., response time, maximum concurrent users), security (e.g., password hashing, access control), and reliability (e.g., acceptable downtime).
   - This will make the system more testable and robust in real-world scenarios.
2. **Formalize requirements list**
   - Extract functional and non-functional requirements into a dedicated chapter with identifiers (FR-x, NFR-x, CR-x).
   - This will improve traceability and ease future maintenance.

7.2 Priority 2: Important Improvements

1. **Document deployment and runtime environment**
   - Add a deployment diagram or textual description of how the application is deployed (e.g., desktop client, server, DB host).
   - Clarify assumptions about network connectivity and multi-user access.
2. **Extend testing strategy**
   - Add integration tests that exercise full flows from controller to database.
   - Consider adding a small set of end-to-end tests simulating typical user journeys.

7.3 Priority 3: Nice-to-Have Enhancements

1. **Add logging and monitoring considerations**
   - Describe how errors and important events are logged.
   - Consider basic monitoring for database connectivity and application health.
2. **Clarify concurrency and multi-user behavior**
   - Document how simultaneous loans and updates are handled at the database level (e.g., locking, isolation).

7.4 Summary Checklist

Table 21: Action Item Summary

Priority	**Items**	**Indicative Effort**
Critical (P1)	2	1–2 days
Important (P2)	2	2–3 days
Nice-to-Have (P3)	2	2–3 days
Total	**6**	**5–8 days**