Software Engineering Audit Report

Relazione_Ingegneria_Software-1.pdf

CAPRA

February 25, 2026

Contents

1 Document Context

Project Objective

The project implements a Library Management System for the University of Florence, developed in Java with JavaFX GUI. The application digitalizes the book reservation, pickup, and catalog management processes, allowing users to remotely browse the library catalog, reserve books for in-person pickup, view active loans with expiration dates, and extend loan durations.

Main Use Cases

- **UC-1** – **Registrazione**: Register a new user by inserting personal data (CF, email, password, etc.) and system verification before redirecting to login.
- **UC-2** – **Effettua Prestito**: Request a book loan by searching the catalog, selecting a book, and system verification of availability and loan limits.
- **UC-3** – **Aggiunge Libro**: Librarian adds a new book to the catalog by inserting book information with system duplicate checking.
- **UC-4** – **Conclude Prestito**: Librarian terminates an active loan after user returns the physical book, making it available again.
- **UC-5** – **Modifica Account**: User modifies personal account information or deletes the account entirely.
- **UC-6** – **Prolunga Prestito**: User extends an active loan duration within system limits.
- **UC-7** – **Cancella Prestito**: User cancels an active loan request.
- **UC-8** – **Effettua Commento**: User leaves comments on book editions (reviews) or specific volumes (physical condition) after or during a loan.
- **UC-9** – **Visualizza Catalogo**: User searches and views the complete book catalog filtered by title, author, genre, or publisher.
- **UC-10** – **Modifica Libro**: Librarian modifies existing book information in the catalog.

Functional Requirements

- User registration with validation of personal data (CF, email, password, phone, birth date, address).
- User login and authentication with email and password verification.
- Book catalog display with search functionality by title, author, genre, and publisher.
- Book reservation with availability verification and loan limit enforcement (maximum 3 concurrent loans per user).
- Loan extension (renewal) with maximum renewal count enforcement.
- Loan cancellation and termination by both users and librarians.
- Comment system for book editions (reviews) and specific volumes (physical condition).
- Librarian functions: add, modify, and delete books from catalog.
- User profile management: view and modify personal information, delete account.
- Real-time catalog and loan status updates across concurrent user sessions.
- Persistent data storage in PostgreSQL database with CRUD operations.

Non-Functional Requirements

- **Security**: Password validation with regex patterns (minimum 5 characters, at least 1 digit and 1 special character); prepared statements to prevent SQL injection; session management for concurrent users.
- **Reliability**: Comprehensive error handling for user errors (invalid input, unauthorized operations) and system errors (database connection failures, data loading errors).
- **Maintainability**: Modular architecture with separated concerns (View, Controller, Business Logic, ORM, Model); design patterns (Singleton, MVC, DAO); organized package structure.
- **Usability**: Intuitive JavaFX GUI with clear navigation between 11 pages; informative alert messages guiding users on errors and solutions.
- **Concurrency**: Session class managing multiple simultaneous user sessions with independent state tracking.
- **Data Integrity**: Field validation using regex patterns; foreign key constraints; transaction management in DAO operations.
- **Testability**: Automated unit tests with JUnit for database operations; UI tests with TestFX for user interactions.

Architecture

The application follows a layered **MVC (Model-View-Controller)** architecture with Page Controller Pattern:

- **View Layer**: 11 JavaFX pages (FXML) for user interface including login, registration, catalog, profile, comments, and book management.
- **Controller Layer**: Page-specific controllers (`LoginController`, `HomePageController`, `ProfiloController`, etc.) handling user requests and navigation.
- **Business Logic Layer**: Service classes (`UserService`, `LoanService`, `BookService`, etc.) implementing validation and complex operations; manages DAO dependencies.
- **Model Layer**: Domain entities (`Utente`, `Prestito`, `Opera`, `Edizione`, `Volume`, `Commento`, `Catalogo`, `Session`).
- **ORM Layer**: `DatabaseConnection` (Singleton pattern) and DAO classes (`UserDAO`, `LoanDAO`, `BookDAO`, `VolumeDAO`, `OperaDAO`, `EditionDAO`, `InfoCommDAO`) for database access via JDBC.
- **Database**: PostgreSQL relational database with 8 tables (Utente, Opera, Edizione, Volume, Prestito, Commento, Commento_edizione, Commento_volume).
- **Design Patterns**: Singleton (database connection), MVC (separation of concerns), DAO (data persistence), Page Controller (page-specific logic).

Testing Strategy

- **Unit Testing (JUnit)**: Tests for database query correctness including book reservation/cancellation, ISBN retrieval, and availability verification with assertions on expected outcomes.
- **UI Testing (TestFX)**: Automated tests simulating user interactions including login with invalid credentials, book reservation flow, loan extension, and registration field validation.
- **Field Validation**: Regex-based validation for email, fiscal code (CF), password, phone number, and birth date with error alerts guiding users.
- **Error Handling Tests**: Verification of user error messages (invalid login, unavailable books, loan limit exceeded) and system error handling (database connection failures, data loading errors).
- **Concurrency Testing**: Session management validation ensuring multiple simultaneous users maintain independent state without data conflicts.
- **Test Coverage**: Tests cover main use cases, error scenarios, and boundary conditions across both business logic and user interface layers.

2 Executive Summary

Quick Overview

Total issues: **15** — HIGH: **3** MEDIUM: **11** LOW: **1**

Average confidence: **99%**

Executive Summary: Software Engineering Document Audit

This document is a Software Engineering specification for a university library management system, comprising requirements, design, architecture, and test planning. It describes a multi-user application with user registration, book loans, catalog management, and commenting features. The audit identified 15 issues across architecture, requirements, and testing, with 3 classified as HIGH severity.

The primary patterns of issues reflect a significant gap between high-level requirements and detailed specification. Business rules are often stated in natural language prose but lack formal capture in use cases, preconditions, postconditions, or test cases. Critical constraints—such as the maximum of 3 active loans per user, uniqueness of comments per user-book pair, and authentication requirements for certain views—are mentioned inconsistently or incompletely across different sections. Additionally, the document specifies only 4 use cases while the use-case diagram and narrative describe many more operations (login, account modification, loan extension, commenting, book management), leaving several important functions without detailed textual specification.

The three HIGH-severity issues demand immediate attention. First, the requirement that users can have at most 3 active loans is stated in the Domain Model but is neither explicitly detailed in UC-2 nor clearly enforced in the DAO/Service logic, making this critical business rule difficult to verify. Second, error and alternative flows for loan-related use cases are handled in code but lack automated test coverage, leaving uncertainty about correct behavior under failure conditions. Third, there is no explicit traceability between the four defined use cases and the implemented tests, making it difficult to confirm that each critical scenario is covered end-to-end.

The overall quality of the document is **moderate**. The system design is architecturally sound and the Domain Model is well-structured; however, the requirements specification is incomplete and inconsistent. The gap between narrative descriptions and formal specifications, combined with limited test coverage for business rules and error paths, creates risk that the implementation may not fully satisfy the intended requirements. The architectural inconsistency regarding Session (described as multi-instantiable but implemented as a static singleton) further undermines confidence in design-code alignment.

1. **Formalize and complete all business rules:** Explicitly document the 3-loan limit, comment uniqueness constraints, and authentication requirements in use-case preconditions, postconditions, and alternative flows. Add missing use cases for login, account management, loan extension, and commenting with clear pre/post conditions and error handling.
2. **Establish traceability and close test gaps:** Create a traceability matrix linking each use case to corresponding JUnit and TestFX tests. Add automated tests for all error and alternative flows (unavailable books, exceeded loan limits, concurrent reservations, database failures) and for the complete book-addition workflow.
3. **Resolve architectural inconsistencies:** Clarify the Session design (either make it truly multi-instantiable or document it as a singleton and adjust tests accordingly). Reconcile the Page Navigation diagram with authentication requirements to explicitly mark which views require login.

3 Strengths

- **Comprehensive architectural design with clear separation of concerns.** The document demonstrates a well-structured MVC architecture with distinct packages (View, Controller, Business Logic, Model, ORM) that isolate responsibilities and promote maintainability. The use of design patterns (Singleton, DAO, Page Controller Pattern) is properly documented and justified for system extensibility.
- **Thorough requirements documentation with detailed use cases.** The project includes a complete use-case diagram with four detailed use-case templates (UC-1 through UC-4) covering registration, loan requests, book management, and loan conclusion, with both basic and alternative courses clearly specified.

- **Complete visual design documentation with mockups and navigation diagrams.** Seven detailed UI mockups (MCK.1 through MCK.7) illustrate all major user interactions, complemented by a comprehensive Page Navigation Diagram showing all possible workflow transitions between system screens.
- **Robust testing strategy combining multiple testing approaches.** The document describes both unit testing (JUnit) and UI testing (TestFX) with specific test cases for critical operations like book reservations, loan extensions, and field validation, demonstrating practical test implementation.
- **Comprehensive security and error handling mechanisms.** The project implements field validation using regex patterns, concurrent access management through session handling, and detailed error handling with user-friendly alerts for login failures, invalid fields, and database connection issues.
- **Well-designed database schema with proper normalization.** The Entity-Relationship model and relational schema are clearly documented with appropriate primary and foreign keys, supporting complex operations like managing comments on both editions and volumes.

4 Expected Feature Coverage

Of 7 expected features: 6 present, 1 partial, 0 absent. Average coverage: **87%**.

Feature	**Status**	**Coverage**	**Evidence**
Unit testing framework implementation	Present	80% (4/5)	The document shows multiple JUnit tests using assertions (assertTrue, assertFalse, assertNotNull, assertEquals) for methods like prenotaLibro, annullaPrestito, ottieniIsbn, operaDisponibile, and describes them as unit tests on specific DAO methods. It also documents a systematic approach in section 3.7 with a summary table, and clearly defines dedicated test methods and classes for specific functionalities, plus TestFX UI tests, but there is no mention of using mocks or stubs for isolated testing; integration-style DB tests are present but not explicitly labeled as such.

Feature	Status	Coverage	Evidence
Use of UML Diagrams for system modeling	Present	100% (8/8)	Section 2.1 presents a use case diagram with actors Utente and Biblioteca and use cases like login, registrazione, effettua prestito, with include relationships. Multiple UML class diagrams are provided for Domain Model, ORM, Business Logic, Controller, View, and a package diagram, and a Page Navigation diagram visualizes navigation flows. Numerous mockups (MCK.1–MCK.7) illustrate UI screens, and the diagrams use standard UML notation and are explicitly used to clarify functional requirements and actor–use case relationships.
Identification and definition of system actors	Present	100% (8/8)	Section 2.1 explicitly identifies two actors, "Utente" and "Biblioteca", and describes in detail their responsibilities such as modifying account, requesting/extending/cancelling prestiti, leaving commenti, and managing the catalogo. It defines interactions between these roles and the system (e.g., searching catalog, concluding prestiti), provides specific user actions, and structures functional requirements per role, which are further detailed in use case templates UC-1 to UC-4.
Definition and Documentation of Use Cases	Partial	60% (3/5)	Use case templates UC-1 to UC-4 define specific user interactions (e.g., registrazione, effettua prestito, aggiunge libro, conclude prestito) and clearly state the Description (user goal) and Main Actor. Each includes a Basic Course and Alternative Course steps, but there is no explicit section for pre-conditions or post-conditions, and relationships between use cases are only visible in the overall use case diagram (e.g., «summary» CRUD prestito/libro) rather than described in the textual templates.

Feature	**Status**	**Coverage**	**Evidence**
User interface and interaction design principles	Present	86% (6/7)	Section 2.4 and figures 7–13 provide UI mockups for key interfaces such as Registrazione Utente, Login, Prenotazione Libro, Pagina Commenti, Aggiunta Libro, Conclusione Prestito, and Catalogo Volumi, showing structured input fields and role-specific screens. The Page Navigation diagram (fig. 6) and description in 2.3 demonstrate navigation elements and flows, and UC-2 plus the Prenotazione Libro mockup and controller code show a step-by-step reservation process. Validation mechanisms and dynamic UI updates are documented in sections 3.7.2 and 3.8 (e.g., alerts on invalid email, login errors, and updating lists after prenotaLibro). Distinct user roles and their functionalities are also reflected in separate views like User Home and Library Home.
Separation of concerns in software architecture	Present	100% (5/5)	Sections 1.2 and 3 describe a layered architecture with distinct packages View, Controller, Business Logic (Service), Model, and ORM/DAO, and figure 15 (Diagramma dei Pacchetti) plus figures 16–20 show their relationships. Section 3.6.3 and code snippets (figs. 23–25) document how controllers call services, which in turn call DAOs, and the roles of each component are clearly explained (e.g., View for UI, Controller as mediator, Business Logic for complex logic, ORM/DAO for persistence). The text emphasizes modularity, maintainability, and separation of responsibilities.

Feature	Status	Coverage	Evidence
Data Access Object (DAO) pattern	Present	86% (6/7)	Section 1.2 and 3.2 explicitly state the use of the DAO pattern, with separate DAOs such as UserDAO, BookDAO/LibroDAO, LoanDAO/PrestitoDAO, EdizioneDAO, OperaDAO, VolumeDAO, InfoCommDAO, each encapsulating SQL queries (e.g., login, prenotaLibro, annullaPrestito, ottieniIsbn, operaDisponibile) inside DAO classes. The ORM class diagram (fig. 17) shows CRUD-like methods (aggiungi*, modifica*, cancella*, caricaCatalogo) and demonstrates abstraction of database access. Patterns for managing data access and dependencies are discussed in 3.2 and 3.6.4, and unit tests in 3.7.1 target DAO methods, but there is no explicit DAO interface type defined; DAOs are concrete classes rather than interfaces.

5 Summary Table

Category	HIGH	MEDIUM	LOW	Total
Architecture	0	1	0	1
Requirements	1	7	0	8
Testing	2	3	1	6
Total	**3**	**11**	**1**	**15**

6 Issue Details

6.1 Architecture (1 issues)

ISS-001 — MEDIUM [100%] — Page 34 The narrative claims that Session is instantiable multiple times to support concurrent users, but the implementation shows only static fields and methods, effectively making Session a global singleton shared by all users. This is an architectural inconsistency between the described design and the actual code, and it also affects testing: tests that rely on Session state may interfere with each other and do not reflect realistic multi-user behavior.

> *La classe Session è instanziabile più volte, così da consentire e l'accesso a più Utenti in contemporanea, ognuno con una propria sessione attribuita.*

Recommendation: Align the description and implementation of Session. For a desktop single-user application, clarify in the text that Session is intentionally implemented as a static singleton and adjust the sentence above. If you want to keep the idea of multiple sessions (e.g., for future web or multi-window scenarios), refactor Session to be an instance class managed per user context instead of using static fields. In both cases, when writing tests that depend on

Session, ensure that you reset Session state (userEmail, utente) in @BeforeEach/@AfterEach to avoid cross-test interference and document this in the test section.

6.2 Requirements (8 issues)

UC-2

ISS-002 — HIGH [100%] — Page 14 The requirement that a user can have at most 3 active loans is stated in the Domain Model description, but it is not present in the UC-2 Effettua prestito basic flow (only a generic "limite di prestiti" is mentioned) and is not clearly enforced in the DAO/Service logic. The prenotaLibro DAO method only checks for an available volume and inserts a PRESTITO; there is no visible query that counts active loans per user. The UI error alert lists many possible reasons in a single generic message, but there is no clear mapping from specific validation failures (e.g. exceeding 3 loans) to distinct alternative flows or tests. This is a critical business rule for the library and should be explicitly modeled and verifiable.

> *Ogni Prestito è collegato a un Utente e a un Volume, l'entità Volume può essere associata a più prestiti in base alle istanze (copie) disponibili, mentre un Utente può avere fino a 3 Prestiti attivi contemporaneamente.*

Recommendation: Make the "max 3 active loans" rule explicit and consistent across requirements, logic, and tests: 1) In UC-2, specify the exact limit in the Description or in step 5 (e.g. "Il sistema verifica che l'utente non abbia già 3 prestiti attivi"). Add an alternative flow 5b: "L'utente ha già 3 prestiti attivi: il sistema rifiuta la richiesta e mostra un messaggio specifico". 2) In the DAO/Service layer, add a query that counts active loans for Session.getUtente() (e.g. restituito = false) and returns a specific error/boolean when the limit is exceeded, before inserting a new PRESTITO. 3) Split the generic error alert in prenotaLibro() into more precise branches, or at least log/encode the specific cause so that it can be tested. 4) Add a JUnit test that creates 3 active loans for a user and verifies that a 4th prenotaLibro() call fails for that user. This will ensure the business rule is actually enforced and demonstrably correct.

See also: TST-001

UC-1

ISS-007 — MEDIUM [100%] — Page 5 All four documented use cases (UC-1..UC-4) lack explicit preconditions and postconditions, and their alternative flows are incomplete. For example, UC-1 does not state the precondition that the user is not already logged in, nor the postcondition that a new UTENTE record is created in the database. UC-2 does not specify what happens if the user is not authenticated, or what the final state of PRESTITO and VOLUME is. UC-3 and UC-4 similarly omit system state changes and boundary conditions. This makes it harder to verify correctness and to map business rules (e.g. max 3 prestiti, uniqueness of books) to the flows.

> *UC-1 Registrazione Description Registrare un nuovo utente Main Actor Utente Basic Course 1. L'utente apre la pagina di registrazione (MCK 1, fig. 7) 2. L'utente inserisce i dati personali (CF, email, password, ecc.) 3. L'utente preme 'registrati' 4. Il sistema verifica che i dati inseriti siano corretti 5. Il sistema reindirizza l'utente alla pagina di login (MCK 2, fig. 8) Alternative Course 4a. Il sistema comunica che i dati inseriti non sono corretti o che l'utente è già registrato e permette di ripetare*

Recommendation: For each existing UC (UC-1..UC-4), add explicit sections for Preconditions and Postconditions, and enrich the Alternative Course. Examples: - UC-1 Registrazione: Preconditions: user is not authenticated; email not yet registered. Postconditions (success): a new UTENTE row is created with valid fields; the user is redirected to login. Alternative flows:

4a1) invalid field X -> show specific error; 4a2) email already registered -> show dedicated message. - UC-2 Effettua prestito: Preconditions: user authenticated; at least one volume available for the selected ISBN. Postconditions (success): a PRESTITO row is inserted; the chosen VOLUME.stato becomes 'in prestito'; num_rinnovi initialized; the loan appears in the user's active loans list. Add alternative flows for: user not logged in, no available copies, user already has a loan for that volume/opera, user has reached 3 active loans. Apply the same pattern to UC-3 and UC-4, clearly stating how the database entities change and what the UI must show after success or failure.

See also: TST-002

General Issues

ISS-003 — MEDIUM [100%] — Page 4 The business rule that each user can leave at most one comment per opera and one per volume is clearly stated in prose, but it is not reflected in the use-case templates (no UC for "effettua commento" is specified) and there is no explicit requirement or description of how the system enforces this constraint (e.g. DB uniqueness, validation logic, error message). This makes the rule hard to verify and easy to violate in implementation.

Ogni utente può lasciare al massimo un commento per opera e uno per volume.

Recommendation: Introduce a dedicated use case for commenting (e.g. "UC-X: Effettua commento su opera" and "UC-Y: Effettua commento su volume") that includes: - Preconditions: user authenticated; for volume comments, user has at least one PRESTITO for that volume. - Basic flow: user opens comments page, writes comment, submits; system checks if the user already has a comment for that opera/volume. - Alternative flow: if a comment already exists, system rejects the operation and shows a specific error. Also document how the rule is enforced technically (e.g. unique constraint on (id_utente, id_edizione) in COMMENTO_EDIZIONE and on (id_utente, id_prestito) or (id_utente, id_volume) in COMMENTO_VOLUME, or equivalent logic in DAO/Service). This will make the requirement testable and traceable.

ISS-004 — MEDIUM [100%] — Page 4 The document states that all operations in the use-case diagram are only possible after login, but the Page Navigation diagram shows that from the Login View the user can navigate directly to Comments View, Opera Comments View, Volume Comments View and Catalogue View without passing through a successful login. This is a contradiction between the high-level requirement and the navigation design, and it is unclear which pages are public and which require authentication.

Tutte le operazioni riportate nel diagramma dei casi d'uso sottostante sono da considerarsi effettuabili solo dopo aver effettuato il log-in.

Recommendation: Clarify the access control rules and align diagrams and text. Decide which pages are accessible without authentication (e.g. only catalogue browsing and reading comments, or nothing). Then: 1) Update section 2.1 to distinguish clearly between operations that require login (e.g. prenota, prolunga, commenta volume, gestione profilo, CRUD libro) and those that are public (e.g. sola visualizzazione catalogo/commenti, if intended). 2) Adjust the Page Navigation diagram (Figura 6) so that flows from Login View to protected pages are only reachable after a successful login, or add a separate "public home" node for anonymous users. 3) Optionally, add a short non-functional requirement about authorization (e.g. "Solo utenti autenticati possono effettuare prestiti, commentare volumi, modificare il profilo; la sola consultazione del catalogo è pubblica").

ISS-005 — MEDIUM [100%] — Page 4 The narrative in section 2.1 describes several operations with non-trivial business rules (modifica account, cancellazione account, estensione prestito, cancellazione prestito, approvazione implicita del prestito da parte del sistema/biblioteca), but there are no corresponding detailed requirements or use cases. For example, it is not specified what happens to existing PRESTITO and COMMENTO records when an account is deleted, or what constraints apply to extending a loan (max number of renewals, date limits). This leaves important lifecycle and consistency aspects under-specified.

> *L'Utente può modificare le informazioni relative al proprio account o cancellarlo del tutto, può richiedere, estendere o cancellare un prestito per un libro a scelta, la prenotazione arriverà poi (se approvata dal sistema) alla biblioteca che predisporrà il libro per il ritiro in loco.*

Recommendation: For each of these operations, add at least a short use case or requirement bullet list that clarifies the business rules: - Modifica account: which fields can be changed; validation rules; postcondition (UTENTE row updated, related PRESTITO/COMMENTO unchanged). - Cancellazione account: preconditions (e.g. no active loans, or what happens if there are); postconditions (UTENTE deleted; ON DELETE CASCADE on COMMENTO and PRESTITO, or logical deactivation); what the user sees after deletion. - Estensione prestito: max number of renewals (num_rinnovi), how the new due date is computed, whether the volume must still be in stato 'in prestito', and what happens if the limit is reached. - Cancellazione prestito: whether it is allowed only before ritiro, or also after; how VOLUME.stato and PRESTITO.restituito are updated. You do not need long templates, but a few precise rules per operation will make the behavior clear and testable.

ISS-006 — MEDIUM [90%] — Page 4 The rules for commenting (no loan required for opera comments, loan required for volume comments) are described in natural language but are not captured in any formal requirement, use case, or test. There is no UC describing the precondition "utente ha un prestito per questo volume" for volume comments, and no test that verifies the system blocks a volume comment when the user has no corresponding PRESTITO. This gap makes it unclear whether the implementation actually enforces these important business constraints.

> *Ogni utente può lasciare al massimo un commento per opera e uno per volume. ... Per lasciare un commento su di un'opera non è necessario aver ottenuto quel libro in prestito, mentre lo è per i commenti riguardanti gli specifici volumi.*

Recommendation: Model and test the comment rules explicitly: 1) Add preconditions to the new comment use cases (see REQ-004): - UC-X Effettua commento su opera: no prestito required. - UC-Y Effettua commento su volume: precondition "esiste almeno un PRESTITO attivo o concluso per quel volume associato all'utente". 2) In InfoCommDAO/Service, implement checks that: - For volume comments, verify via a query that Session.getUtente() has at least one PRESTITO for the target volume (or prestito id) before inserting COMMENTO_VOLUME. 3) Add at least one JUnit or TestFX test that attempts to comment a volume without a loan and asserts that the operation fails and the correct error message is shown. Document these checks briefly in section 3.8.1 under "Operazioni non consentite" so the behavior is traceable from requirements to code and tests.

ISS-008 — MEDIUM [100%] — Page 5 Use cases are only specified for 4 scenarios (registrazione, effettua prestito, aggiunge libro, conclude prestito), while the use-case diagram and the textual description in 2.1 mention many more operations (login, modifica account, elimina account, prolunga prestito, cancella prestito, effettua commento su opera/volume, modifica libro, etc.). This leaves several important business functions without any textual use-case description (no basic flow, no alternative/error flows, no pre/post conditions), which weakens traceability and completeness of the requirements.

> *Di seguito sono descritti 4 dei casi d'uso più importanti.*

Recommendation: Extend the Use-Case Templates section so that every use case shown in the diagram in Figura 1 and described in section 2.1 has at least a minimal textual specification. For each missing use case (e.g. login, modifica account, elimina account, prolunga prestito, cancella prestito, effettua commento su opera, effettua commento su volume, modifica libro), add: (1) a short Description, (2) Main Actor, (3) Basic Course with numbered steps, (4) Alternative Course(s) for main error/exception situations, and (5) explicit preconditions and postconditions. This does not need to be extremely detailed, but every operation that appears in the diagram or is implemented in code should have a corresponding textual use case.

ISS-009 — MEDIUM [100%] — Page 22 The documented tests cover only a subset of the main business functions: prenotazione/annullamento prestito, recupero ISBN, verifica disponibilità, login errato, prenotazione libro da UI, prolungamento prestito, and email validation. There are no tests shown for registration success, account modification, account deletion, comment creation (opera/volume), book CRUD by the library, or the "max 3 prestiti" rule. Given that these are explicitly mentioned as key use cases and business rules, the lack of corresponding tests reduces confidence that the implementation matches the requirements.

> *La tabella seguente riepiloga i test automatici riportati nel dettaglio nelle sezioni successive. I test coprono sia sia la logica applicativa con JUnit (sez. 3.7.1), sia l'interfaccia utente tramite TestFX (sez. 3.7.2). Pur non rappresentando la totalità delle classi e dei metodi testati, questi esempi evidenziano i principali casi d'uso, inclusi gli scenari di errore e i comportamenti limite.*

Recommendation: Extend the test suite (even with a few representative tests) to cover the most critical missing requirements: - Registration success: a JUnit or TestFX test that registers a valid user and verifies that a UTENTE row is created and login works. - Account modification and deletion: tests that update user data and delete an account, verifying DB state (UTENTE and related COMMENTO/PRESTITO) and UI behavior. - Comment creation: tests for adding a comment on an opera and on a volume, including the case where the user already has a comment (should fail) and, for volume comments, the case without a loan (should fail). - Max 3 loans: a test that attempts to create a 4th active loan and asserts that the operation is rejected. You can briefly document these additional tests in section 3.7 (even as summaries) to show traceability between the main requirements/use cases and the implemented tests.

6.3 Testing (6 issues)

UC-2

TST-001 — HIGH [100%] — Page 5 Error and alternative flows for loan-related use cases are handled in the code (alerts for non-available books, exceeded loan limits, concurrent reservations) but are not covered by automated tests. The JUnit tests for LoanDAO and BookDAO focus on successful reservation/cancellation and availability counting, and the TestFX tests for prenotazione and proroga prestito only verify success messages. There are no tests that assert the correct behavior when the book is not available, when the user exceeds the maximum number of loans, or when the same book is reserved concurrently.

> *Alternative Course 4a. Il sistema comunica che il libro non è disponibile 5a. Il sistema comunica che l'utente ha superato il limite massimo temporaneo di prenotazioni consentite*

Recommendation: For all loan-related scenarios (UC-2 Effettua prestito and the implicit flows around loan limits and availability), add negative-path tests. At unit level, create JUnit tests that set up the database so that: (a) no available volume exists for a given ISBN and assert that prenotaLibro() returns false; (b) the user already has 3 active loans and assert that prenotaLibro() fails. At UI level, add TestFX tests that attempt to reserve a non-available book or a fourth loan and verify that the error dialog shown in Figura 46 appears with the expected text. This will demonstrate that the alternative courses 4a and 5a are correctly implemented.

See also: ISS-002

UC-1

TST-002 — HIGH [100%] — Page 22 There is no explicit traceability between the defined use cases (UC-1 Registrazione, UC-2 Effettua prestito, UC-3 Aggiunge libro, UC-4 Conclude Prestito) and the implemented tests. The test section lists tests by technical function (LoanDAO, BookDAO, login UI, registration UI) but never maps them back to specific UC identifiers or requirements. This makes it hard for a reviewer to verify that each critical use case is actually covered end-to-end.

> *Pur non rappresentando la totalità delle classi e dei metodi testati, questi esempi evidenziano i principali casi d'uso, inclusi gli scenari di errore e i comportamenti limite.*

Recommendation: Add a small traceability table that, for each main UC (at least UC-1, UC-2, UC-3, UC-4), lists which JUnit and/or TestFX test methods cover its basic and alternative flows. For example: a row for UC-2 Effettua prestito referencing testPrenotaEAnnullaPrestito(), testPrenotaLibro (UI), and operaDisponibile() tests; a row for UC-1 Registrazione referencing testEmailNonValidaMostraErrore() and any positive registration test. If some UCs currently have no tests, explicitly mark them and add at least one positive-path test and, where relevant, one error-path test for each.

See also: ISS-007

UC-4

TST-003 — MEDIUM [100%] — Page 6 The use case UC-4 Conclude Prestito is described and there is a mockup (MCK.6) showing the librarian concluding a loan, but no corresponding JUnit or TestFX tests are documented. Existing tests cover prenotazione, annullamento, and proroga prestito, but not the librarian-side conclusion flow that should update the volume state back to 'disponibile' and update the UI.

> *UC-4 Conclude Prestito Description Terminare un prestito effettuato dall'utente Main Actor Bibliotecario Basic Course 1. Bibliotecario seleziona il prestito nella lista dei prestiti attivi 2. Clicca su termina prestito (MCK 6, fig: 12) 3. Il sistema rende il libro nuovamente disponibile agli utenti*

Recommendation: Add tests specifically for the loan conclusion flow. At DAO level, create a JUnit test that: (a) creates a loan for a volume; (b) calls the method that concludes the loan (e.g., concludePrestito or equivalent); (c) asserts that the prestito row is marked as restituito and that the corresponding volume.stato is 'disponibile'. At UI level, add a TestFX test that opens the librarian home (OpUserController / MCK.6), selects an active loan, clicks "Concludi prestito", confirms the dialog, and verifies that the loan disappears from the active list and that the success message is shown.

UC-3

TST-004 — MEDIUM [100%] — Page 6 The use case for adding a book (UC-3) and the related operations on Opera/Edizione/Volume are central to the librarian role, but there are no tests shown for LibroDAO.aggiungiLibro(), OperaDAO/EdizioneDAO/VolumeDAO, or the AggiungiLibroController / AggiungiLibro view. The only DAO tests concern loans and availability;

the only UI tests concern login, reservation, loan extension, and registration email validation. This leaves the whole book-creation workflow (including duplicate detection in step 4/4a) without explicit automated coverage.

> *UC-3: Aggiunge libro Description Aggiungere un nuovo libro al catalogo Main Actor Bibliotecario Basic Course 1. Bibliotecario apre il catalogo libri 2. Apre la pagina di aggiunta libro (MCK 5, fig. 11) 3. Inserisce le informazioni del libro da aggiungere 4. Il sistema controlla che non sia già stato inserito 5. Il sistema lo aggiunge correttamente Alternative Course 4a. Il libro è già presente nella collezione e il sistema lo notifica*

Recommendation: Introduce at least a minimal set of tests for the book management workflow. At unit level, add JUnit tests for LibroDAO.aggiungiLibro(), OperaDAO.aggiungiOpera(), EdizioneDAO.aggiungiEdizione(), and VolumeDAO.aggiungiVolume() that: (a) insert a new book and verify it appears in the catalog; (b) attempt to insert a duplicate and verify the DAO returns false or throws the expected exception. At UI level, add a TestFX test that opens the AggiungiLibro view (MCK.5), fills in valid data, clicks "Salva opera" and verifies that the book appears in the catalog list, plus a test that tries to add a duplicate and checks that the notification for case 4a is shown.

General Issues

TST-005 — MEDIUM [100%] — Page 33 The document describes several system error-handling behaviors (empty catalog, generic system error, database connection failure) and user error alerts, but there are no automated tests that verify these error paths. All TestFX examples focus on specific user errors (login fail, invalid email) and success flows; there is no test that simulates a failing BookDAO.caricaCatalogo() or a database connection error and asserts that the correct alert is shown and the UI remains in a safe state.

> *Viene qui sopra mostrata una classica gestione dell'errore nel caso in cui una query di lettura nel database fallisca e finisca in eccezione. Qualora l'oggetto catalogo (in cui viene salvato il risultato della query dal BookDAO) sia vuoto del tutto o non contenga nessun Volume, il sistema mostra un alert di errore che notifica l'Utente del temporaneo errore; quindi riprova nuovamente a caricare il catalogo richiamando il metodo aggiornaCatalogo. Se invece il problema fosse nel flusso del programma e pure questo metodo dovesse finire in eccezione, un errore più generico viene mostrato.*

Recommendation: For the most important error-handling paths, add targeted tests. For example, use mocking or a test double for BookDAO to force caricaCatalogo() to return null or throw an exception, then write a TestFX test that calls the UI action that triggers stampaCatalogo() and verifies that the "Errore di Caricamento" or "Errore di Sistema" dialog appears as described. Similarly, add a unit test for DatabaseConnection.getConnection() that uses an invalid URL or credentials in a test configuration and asserts that a SQLException with the expected message is thrown. This will demonstrate that your documented error-handling logic actually works.

TST-006 — LOW [100%] — Page 34 Field validation is implemented with several regex-based methods (email, CF, password, phone, date range) and is used in registration and profile management, but only the invalid email case is covered by a UI test. There are no tests that check positive cases (all fields valid) or other negative cases (invalid CF, password too weak, phone number out of range, date of birth outside allowed range). This leaves a significant part of the validation logic unverified.

> *Di seguito vengono mostrate le espressioni regolari (REGEX) e i metodi di validazione usati per verificare la correttezza nell'operazione di Registrazione.*

Recommendation: Extend your validation tests beyond the single invalid-email scenario. At unit level, add JUnit tests for the validation methods (isValidEmail, isValidCf, isValidPassword, isValidPhone, isValidDate) with multiple valid and invalid examples. At UI level, add at least one TestFX test that fills the registration form with fully valid data and verifies that no validation dialog appears and that the user is redirected to the login page, plus one or two tests for other invalid fields (e.g., wrong CF format, weak password) checking that the corresponding alert is shown. This will give much stronger evidence that your REGEX and validation rules behave as intended.

7 Priority Recommendations

The following actions are considered priority:

1. **ISS-002** (p. 14): Make the "max 3 active loans" rule explicit and consistent across requirements, logic, and tests: 1) In UC-2, specify the exact limit in the Descripti...
2. **TST-001** (p. 5): For all loan-related scenarios (UC-2 Effettua prestito and the implicit flows around loan limits and availability), add negative-path tests.
3. **TST-002** (p. 22): Add a small traceability table that, for each main UC (at least UC-1, UC-2, UC-3, UC-4), lists which JUnit and/or TestFX test methods cover its basic ...

8 Traceability Matrix

Of 14 traced use cases: 5 fully covered, 0 without design, 9 without test.

ID	Use Case	Design	Test	Gap
UC-1	Registrazione	✓	✓	—
UC-2	Effettua prestito	✓	✓	—
UC-3	Aggiunge libro	✓	×	UC-3 Aggiunge libro is implemented via AggiungiLibroController, LibroService and related DAO/Model classes, but no unit ...
UC-4	Conclude Prestito	✓	×	UC-4 Conclude Prestito has UI and DAO/Service support but no explicit JUnit or TestFX test for concluding a loan is desc...
UC-5	Login	✓	✓	—
UC-6	Modifica account	✓	×	UC-6 Modifica account (including update and delete) is supported by ProfiloController and UtenteService/UtenteDAO, but n...
UC-7	Elimina account	✓	×	UC-7 Elimina account has clear controller/service/DAO support but no automated tests for account deletion are shown.

ID	Use Case	Design	Test	Gap
UC-8	Modifica libro	✓	×	UC-8 Modifica libro is implemented via ModificaLibroController and related services/DAOs, but no tests for editing or de...
UC-9	Effettua commento su opera	✓	×	UC-9 Effettua commento su opera has DAO/service/controller and UI pages, but no unit or UI tests for adding, viewing, or...
UC-10	Effettua commento su volume	✓	×	UC-10 Effettua commento su volume is designed and implemented but lacks explicit automated tests for volume comments.
UC-11	Visualizza catalogo opere	✓	×	UC-11 Visualizza catalogo opere is fully implemented but there are no documented tests specifically asserting correct ca...
UC-12	Visualizza catalogo volumi	✓	×	UC-12 Visualizza catalogo volumi and related volume management are designed, but no tests for volume listing, add, edit,...
UC-13	Prolunga prestito	✓	✓	—
UC-14	Cancella prestito	✓	✓	—

9 Terminological Consistency

Found **10** terminological inconsistencies (5 major, 5 minor).

Group	Variants found	Severity	Suggestion
Data access layer naming	"DAO", "Data Access Object", "ORM", "ORM Tool", "package ORM", "DAO Libro", "LoanDAO", "BookDAO", "UserDAO", "PrestitoDAO", "UtenteDAO", "LibroDAO", "InfoCommDAO", "EdizioneDAO", "OperaDAO", "VolumeDAO"	MAJOR	Use a single, consistently named DAO layer; preferably standardize on either the Italian plural "DAO" or the English singular/plural form, but avoid mixing "DAO" and "ORM" as separate concepts when they both denote the data access layer.
Business layer naming	"Business Logic", "package Business Logic", "Service", "classi Service", "layer service"	MAJOR	Use one consistent term for the business layer; for example, always use "Business Logic" (as a layer name) and avoid alternating with "Service" as if it were a different concept.

Group	Variants found	Severity	Suggestion
Library actor naming	"Biblioteca", "attore Biblioteca", "Bibliotecario", "Login as Library", "Library Home", "OpUserController" (described as "home page a disposizione della biblioteca"), "OpUtente" (view for library operations)	MAJOR	Use a single, consistent term for the library-side actor; for example, standardize on "Biblioteca" or on "Bibliotecario" depending on whether the actor is the institution or the librarian, and update all use cases and diagrams accordingly.
Book / work entity naming	"libro", "Libro", "Book", "Opera", "opera", "Catalogo Libri", "Catalogo Volumi", "Informazioni Opera", "Aggiungi nuovo libro", "CRUD libro", "DAO Libro", "BookDAO"	MAJOR	Use one consistent term for the book entity; for example, use "Libro" for the conceptual entity in the domain and UI, and reserve "Opera" strictly for the specific domain concept if needed, clearly explaining the distinction.
Loan vs reservation naming	"prestito", "Prestito", "prenotazione", "prenota", "Prenotazione Libro", "Summary Prestito", "CRUD prestito", "lista dei prestiti", "prenotazioni attive"	MAJOR	Use a single, consistent term for the loan/reservation concept; for example, standardize on "Prestito" and clearly distinguish it from any separate "prenotazione" concept, or explicitly define both if they are different states.
User actor naming	"Utente", "utente", "User", "UserDAO", "UserService", "OpUserController", "OpUtente"	MINOR	Use a single, consistent term for the user-side actor; for example, always use "Utente" and avoid mixing with the English "User" in class or package names unless you decide to fully Anglicize all such identifiers.
Authentication operation naming	"login", "log-in", "Login", "Accedi", "Accedi al tuo account", "Registrazione", "registrazione", "Registrati", "Crea il tuo account", "RegistrationController", "RegistrazioneController"	MINOR	Standardize the naming of the login/registration operations; for example, consistently use "login" and "registrazione" (or their English equivalents) across use cases, methods, and UI labels.
Catalog view naming	"Catalogo Libri", "Catalogo Volumi", "catalogo", "Visualizza Catalogo", "VisualizzaCatalogo", "stampaCatalogo"	MINOR	Use a single, consistent term for the catalog views; for example, distinguish clearly between "Catalogo Libri", "Catalogo Volumi" and "VisualizzaCatalogo" only if they represent different views, otherwise unify the terminology.

Group	**Variants found**	**Severity**	**Suggestion**
Comment feature naming	"Commento", "Commenti", "Pagina Commenti", "InfoComm", "InfoCommController", "InfoCommDAO", "InfoCommService", "Vedi Recensioni"	MINOR	Use a single, consistent term for the comment-related components; for example, standardize on either "Commento" or "Comm" and align DAO/Service/View names accordingly.
Home page naming	"HomePage", "HomePageController", "User Home", "Library Home", "OpUserController", "OpUtente"	MINOR	Standardize the naming of the home/main pages; for example, use "HomePage" plus a clear qualifier (e.g., "HomePageUtente", "HomePageBiblioteca") instead of mixing "HomePage", "User Home", "Library Home", and "OpUtente".