BIBLIOTECA

Progetto per il corso ISPW

Davide Di Francesco

Matricola: 0255559

INDICE

1. Software Requirement Specification

1.1 Introduzione

Questo documento descrive il sistema "Biblioteca", un gestionale di biblioteca con interfaccia JavaFX.

Il documento definisce:

- funzionalità offerte ai diversi ruoli (Admin, Bibliotecario, Utente finale con tessera)
- requisiti funzionali e non funzionali
- principali flussi operativi (prestiti, gestione catalogo, gestione utenti)
- architettura logica e pattern di design adottati

Scopo: fornire una base unica sia per la valutazione ISPW sia per la manutenzione futura.

1.2 Panoramica del sistema

Il sistema consente di gestire una biblioteca fisica.

Ci sono tre ruoli:

- Admin
  - Vede gli utenti
  - Imposta/modifica credenziali di accesso (username, password, ruolo)
  - Abilita o disabilita utenti tramite data di scadenza della tessera

- Bibliotecario
  - Gestisce il catalogo libri (aggiunta, modifica, rimozione)
  - Gestisce i prestiti (apertura prestito, registrazione restituzione)
  - Gestisce gli utenti (creazione nuovo utente, modifica dati anagrafici, cancellazione)
  - Importa/esporta CSV del catalogo e degli utenti

- Utente
  - Può consultare il catalogo
  - Può vedere solo le copie disponibili
  - Può visualizzare il proprio profilo (dati personali, stato tessera)
  - Può consultare i propri prestiti (attivi e storici)
  - Può esportare i propri prestiti in CSV

La UI è a tab (Home, Catalogo, Prestiti, Utenti, Profilo, I miei prestiti) e la disponibilità delle tab cambia in base al ruolo, usando SessionContext.isAdmin(), isBibliotecario(), isUtente().

Il sistema supporta due modalità di persistenza:

- File/JSON locale tramite DAO come JsonUtenteDAO, che salva utenti e credenziali in utenti.json con salvataggio incrementale e utenti di default auto-generati.
- DB relazionale (MySQL) tramite DAO DB. La scelta avviene in fase di avvio tramite wizard iniziale (StartupDialog + StartupResult), che chiede impostazioni e credenziali e configura SessionContext di conseguenza.

1.3 Requisiti hardware e software

- Obbligatorio: Java SDK (17+)
- Obbligatorio: JavaFX
- Obbligatorio: Server MySQL/MariaDB
- Obbligatorio: Filesystem scrivibile
- Consigliato: IntelliJ Idea + Supporto Maven
- Consigliato: 4GB+ RAM

1.4 Sistemi correlati, pro e contro

- **Biblioteche di Roma**: sistema unificato che gestisce tutte le biblioteche pubbliche inserite nel sistema OPAC della provincia di Roma. Consente di cercare libri nel database, effettuare prenotazioni, spostare libri da una biblioteca all'altra per consentire il ritiro in quella più vicina, visualizzare le novità e gli eventi nelle varie biblioteche. Non consente tuttavia di avere la visuale di tutti i libri presenti in una

determinata biblioteca, tipo catalogo, e non ha visuale sulle copie disponibili in una determinata biblioteca ma solo di una generica “disponibilità al prestito”.

- **OPAC SBN**: è il catalogo del Servizio Bibliotecario Nazionale. Contiene la lista di tutti i libri, documenti, sistemi multimediali gestiti dalle varie biblioteche pubbliche del territorio italiano. Consente di navigare all’interno del catalogo, ricercare e visualizzare in quale biblioteca/sito è fisicamente o digitalmente presente. Richiede tuttavia una registrazione apposita per la prenotazione o la richiesta di un determinato supporto, che varia a seconda del tipo di supporto e alla piattaforma di riferimento.

2. User Stories

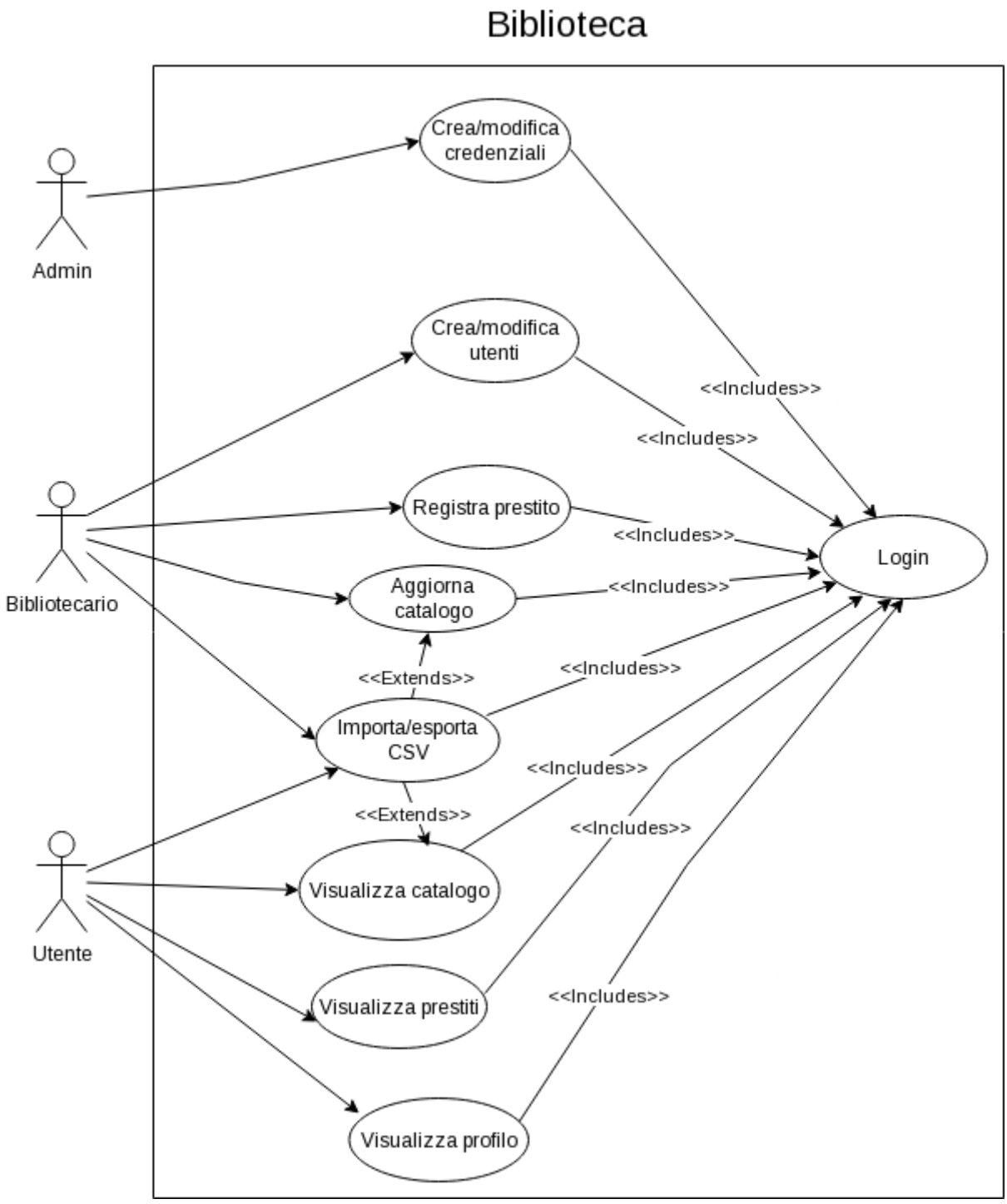
2.1 User Stories

- 2.1.1 US-1: Come Utente, voglio vedere i miei prestiti attivi e passati, così posso controllare cosa devo restituire.
- 2.1.2 US-2: Come Bibliotecario, voglio aggiungere/modificare/rimuovere libri dal catalogo, così posso mantenere il catalogo aggiornato.
- 2.1.3 US-3: Come Admin, voglio creare e aggiornare le credenziali di accesso di un utente, così posso consentire loro l’accesso al sistema.

2.2 Requisiti funzionali

- 2.2.1 FR-1: Il sistema deve calcolare automaticamente le copie disponibili sottraendo i prestiti attivi dal totale copie.
- 2.2.2 FR-2: Il sistema deve gestire l'anagrafica utente e la validità della tessera.
- 2.2.3 FR-3: Il sistema deve registrare la data di prestito e permettere la chiusura inserendo la data di restituzione.

2.3 Use Case Diagram



2.4 Use Case: passi interni

- 2.4.1 UC-1: Registrare un prestito (Bibliotecario)
  - 1. Il Bibliotecario apre la sezione "Prestiti"

    - 2. Il sistema mostra la tabella dei prestiti (loansTable) filtrabile.
    - 3. Il Bibliotecario sceglie "Registra Prestito".
    - 4. Il sistema mostra la lista dei libri disponibili, calcolando per ciascun libro se ci sono copie libere (libriDisponibili()).
    - 5. Il Bibliotecario seleziona un libro.
    - 6. Il sistema mostra la lista utenti attivi (tessera non scaduta) (SelectUserDialog).
    - 7. Il Bibliotecario seleziona l'utente.
    - 8. Il sistema mostra una finestra di conferma prestito (PrestitoDialog).
    - 9. Il Bibliotecario conferma.
    - 10. Il sistema chiama ui.registerLoan(...).
    - 11. Se l'esito è OK, il sistema:
        - aggiunge il prestito
        - aggiorna la vista prestiti (aggiornaPrestiti())
        - aggiorna la disponibilità copie (aggiornaCatalogoLibri())
        - mostra messaggio "Prestito registrato con successo."
    - 12. Se l'utente è inattivo, il sistema mostra "Utente non attivo" e non registra il prestito.
    - 13. Se il DB fallisce, il sistema mostra "Impossibile registrare il prestito."
- Extensions:
    - 4a. Nessun libro disponibile → il sistema mostra errore "Nessun libro disponibile".
    - 6a. L'utente selezionato ha tessera scaduta → esito UTENTE_INATTIVO.
- 2.4.2 UC-2: Gestire le credenziali utente (Admin)

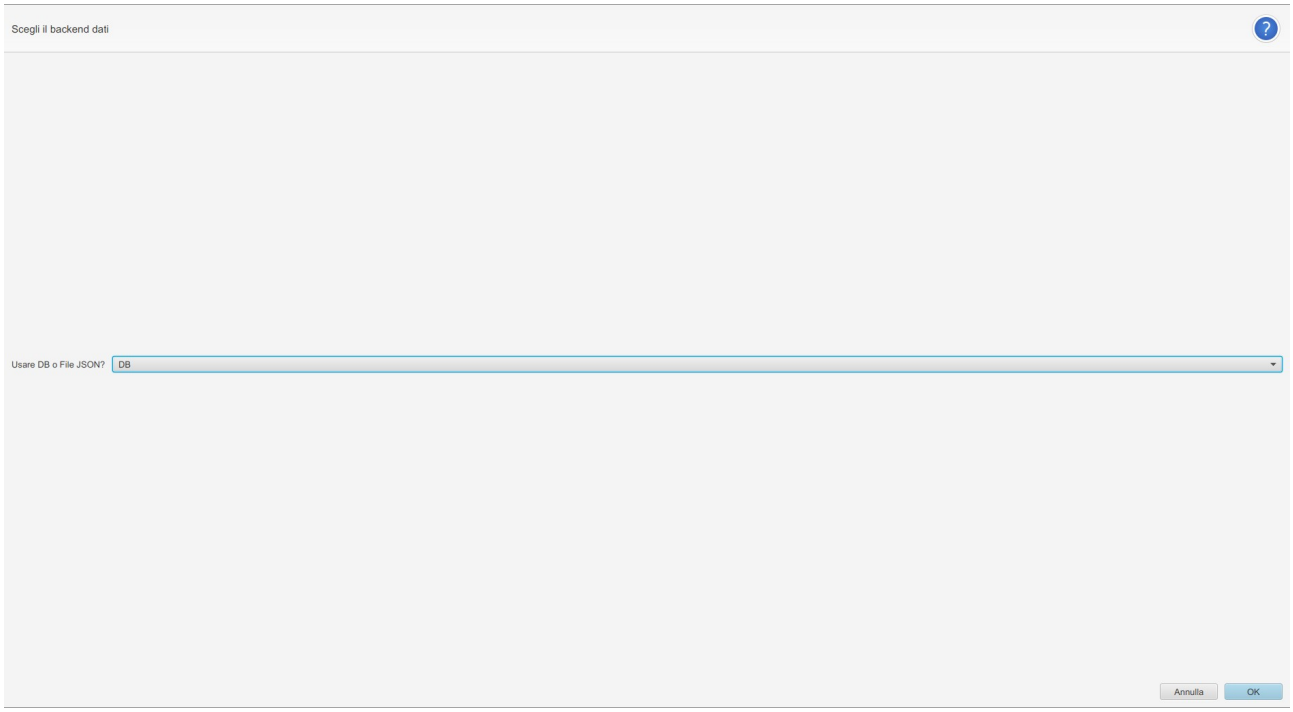
    - L’Admin apre la sezione "Utenti".
    - Il sistema mostra la tabella utenti, con colonne Username e Password mascherata ("*****").
    - L’Admin seleziona un utente.
    - L’Admin sceglie "Crea/Modifica credenziali".
    - Il sistema apre CredentialsDialog, precompilando lo username esistente se presente.
    - L’Admin inserisce username e password.
    - Il sistema decide:
        - se l’utente aveva già credenziali → chiama ui.updateCredentials(...)
        - altrimenti → chiama ui.createCredentials(...)
    - Se l’operazione va a buon fine:
        - il sistema aggiorna la lista utenti (aggiornaUtenti())
        - mostra "Credenziali salvate."
    - Se fallisce:
        - mostra "Impossibile salvare credenziali."
- Extensions:
    - 3a. Nessun utente selezionato → il sistema mostra "Seleziona un utente per associare le credenziali".
    - 1a. Se il ruolo corrente NON è Admin, il sistema rifiuta subito con "Solo Admin può gestire le credenziali." (ensureBibliotecarioOrAdmin() / check di ruolo diretto in handleCredentials()).

3. Storyboards

Le storyboards illustrano i vari processi e le funzioni dell'applicazione Biblioteca, suddivise anche nei vari utenti che possono utilizzarla.

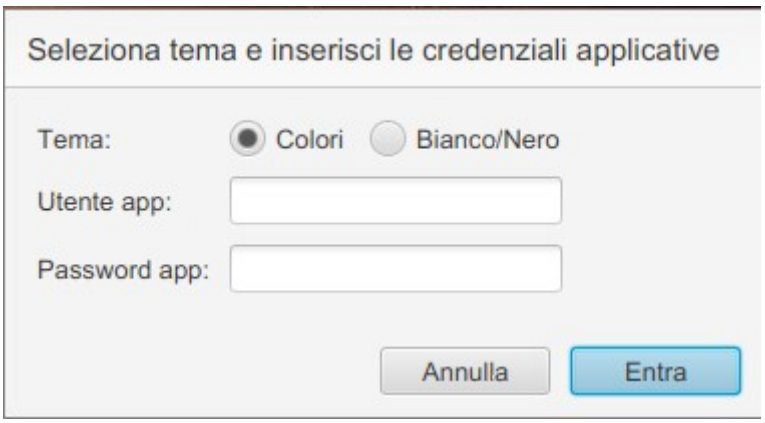
- 3.1 Login (tutti gli utenti)
    - 3.1.1 Scelta del backend

Attraverso questa schermata, qualsiasi utente può scegliere se utilizzare come backend il DB o i file locali JSON.



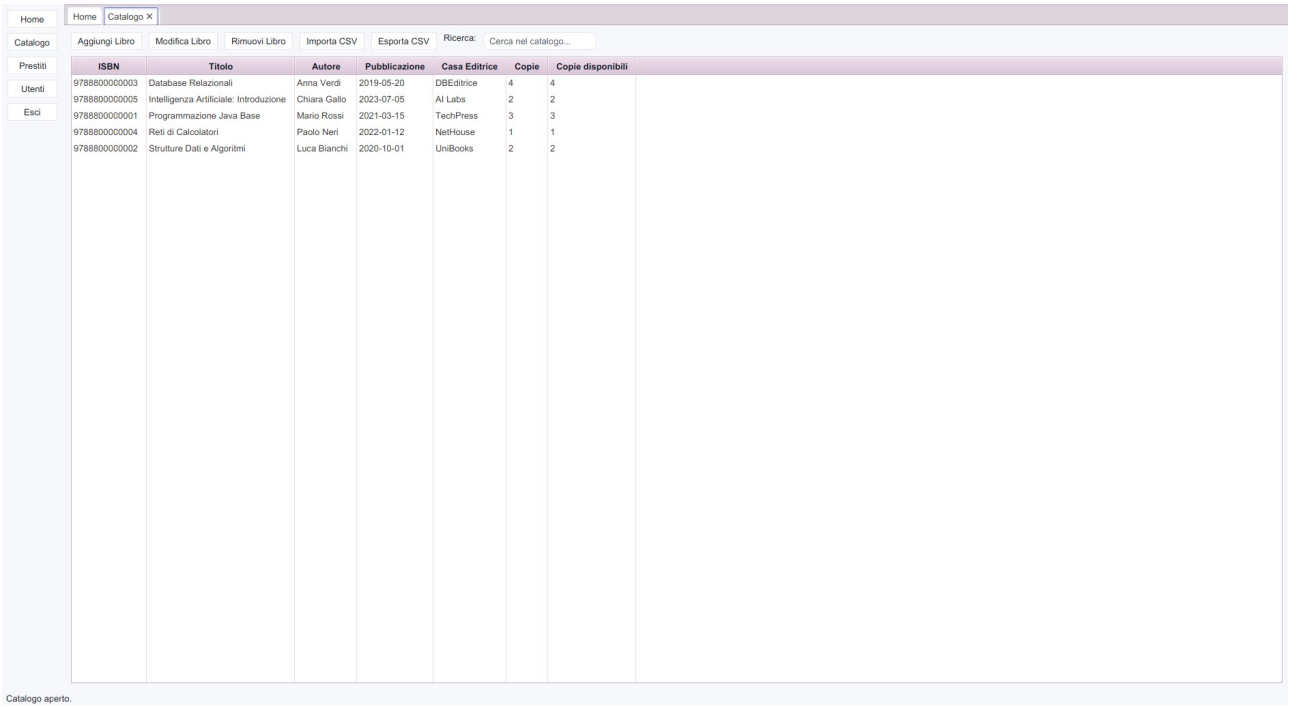
    - 3.1.2 Inserimento delle credenziali

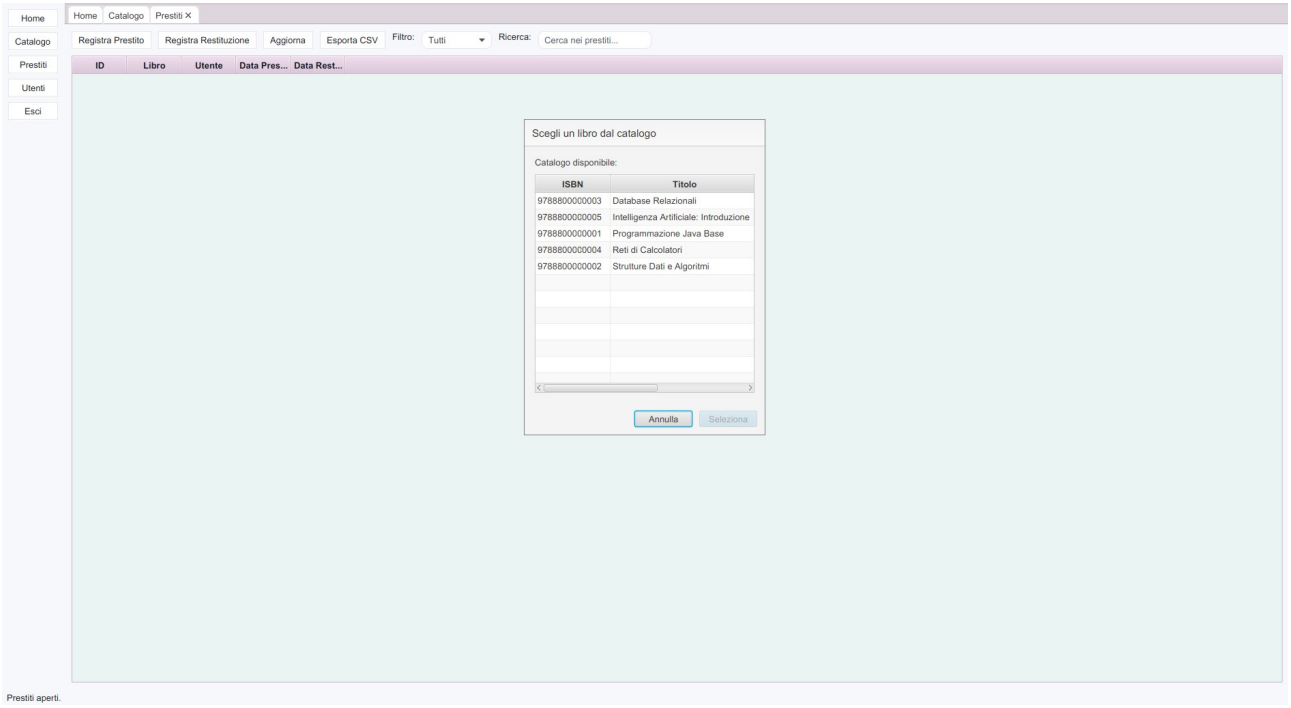
In questa schermata, qualsiasi utente può inserire le proprie credenziali e scegliere il tema a colori o in bianco e nero.



- 3.2 Bibliotecario
    - 3.2.1 Visualizzazione/Aggiornamento catalogo

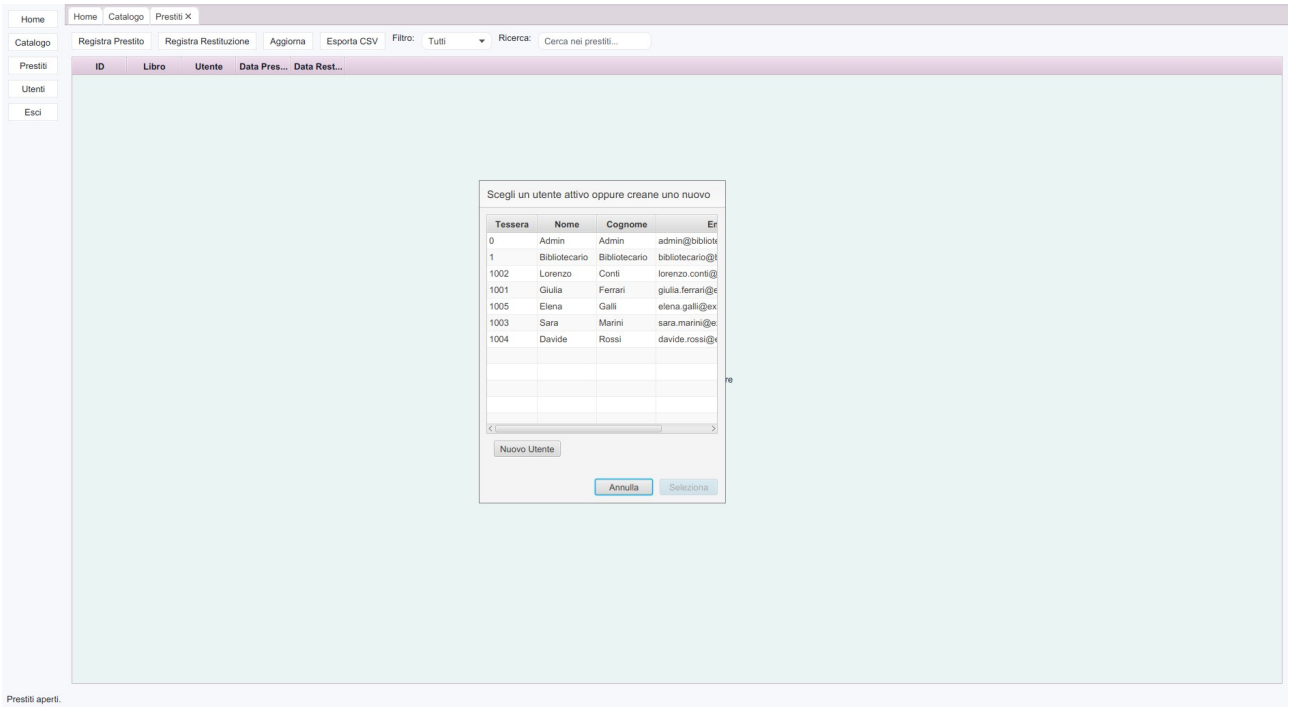
In questa sezione, il bibliotecario può inserire, rimuovere o modificare i libri presenti nel catalogo della biblioteca.





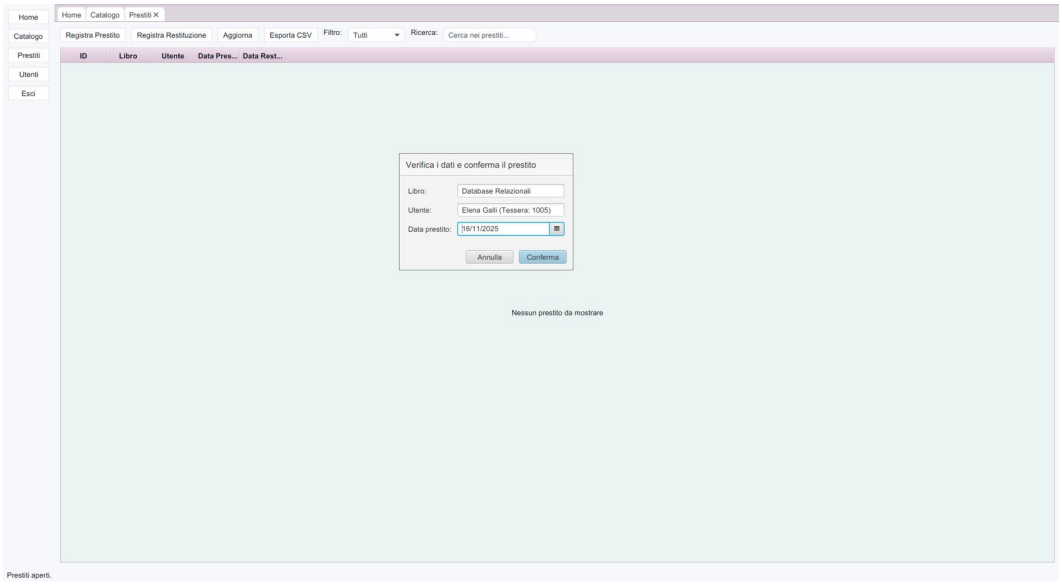
- 3.2.2 Inserimento/rimozione utenti

Nella sezione utenti, il bibliotecario può inserire nuovi utenti assegnando un numero di tessera, oppure può rimuovere utenti già presenti.



- 3.2.3 Registrazione prestiti

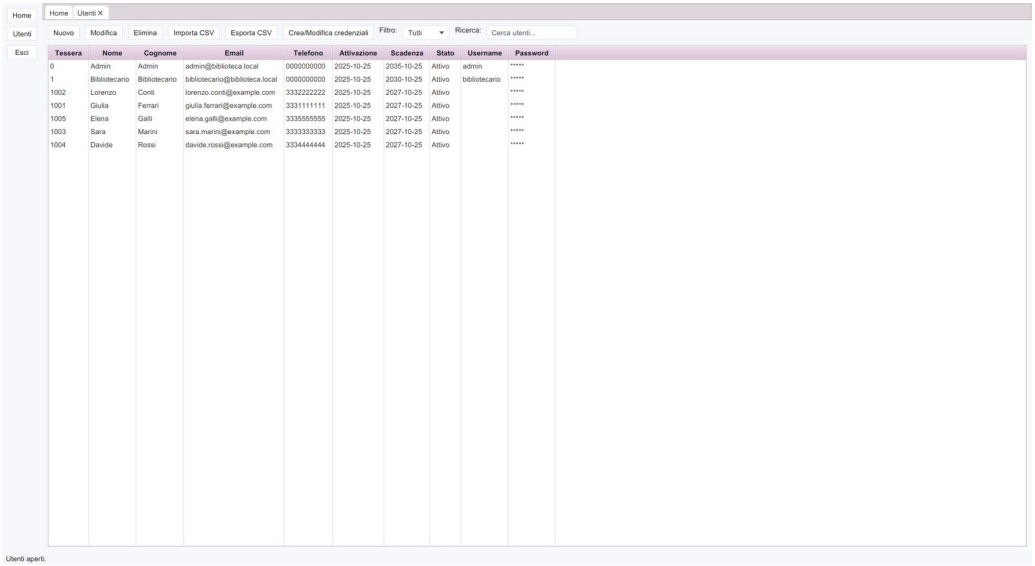
Nella sezione prestiti, il bibliotecario può registrare un nuovo prestito oppure può registrare la restituzione di un prestito già attivo.



- 3.3 Admin

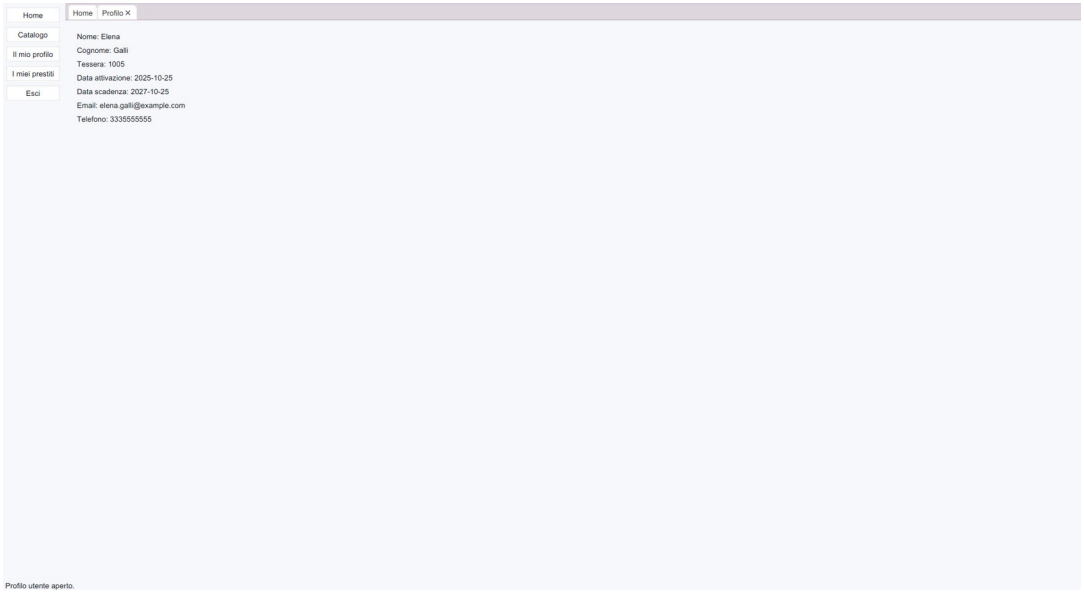
  - 3.3.1 Creazione/modifica credenziali

L’amministratore può creare o modificare le credenziali di accesso di ogni utente creato dal bibliotecario, per permettere loro l’accesso al database.



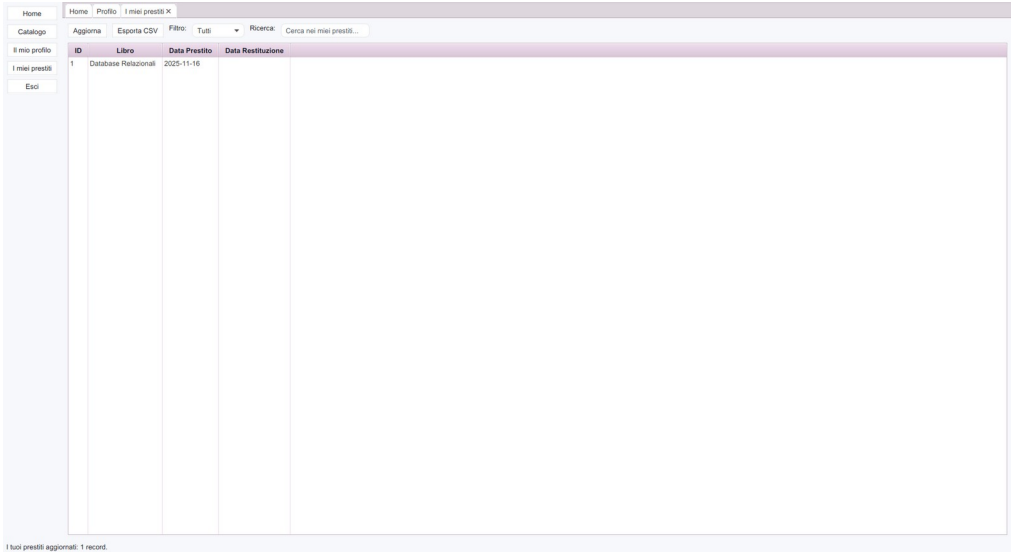
- 3.4 Utente
    - 3.4.1 Visualizzare profilo

Ogni utente ha la possibilità di visionare il proprio profilo, dove trova i propri dati personali relativi alla sua registrazione nella biblioteca.



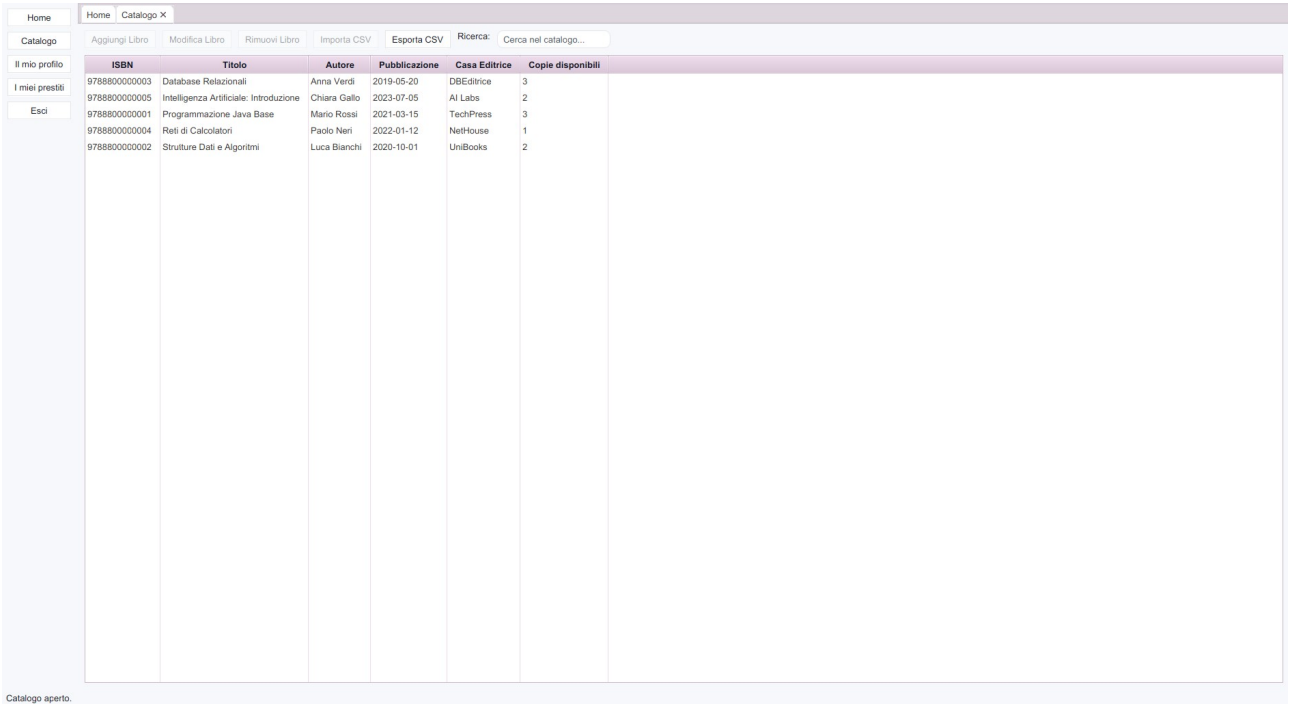
- 3.4.2 Visualizzare prestiti

Nella sezione prestiti, l’utente può visionare quali sono i suoi prestiti attivi o già restituiti.



- 3.4.3 Visualizzare catalogo

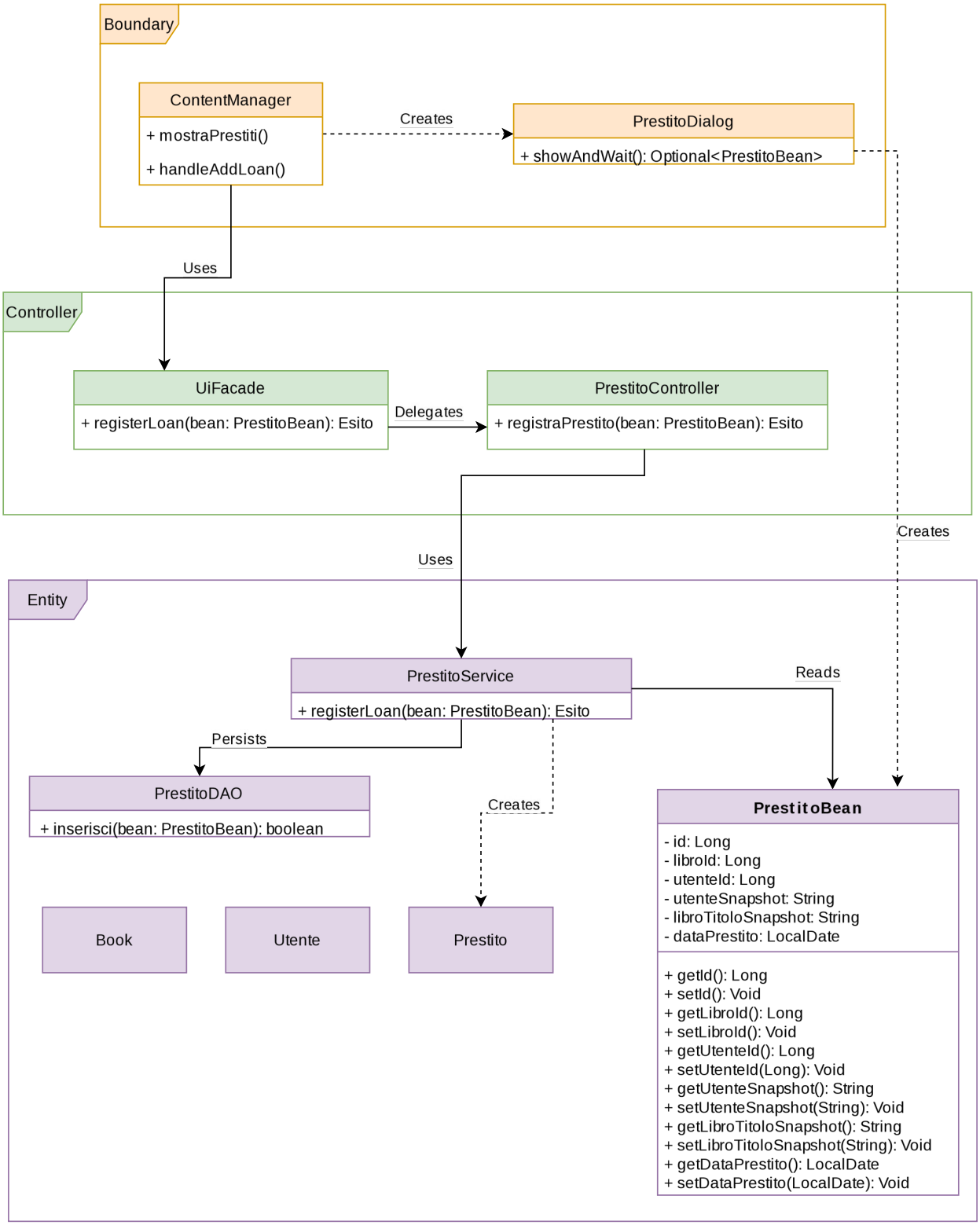
L’utente ha la possibilità di visionare il catalogo dei libri (senza poterlo modificare), per vedere i libri e le copie ancora disponibili.



4. Design

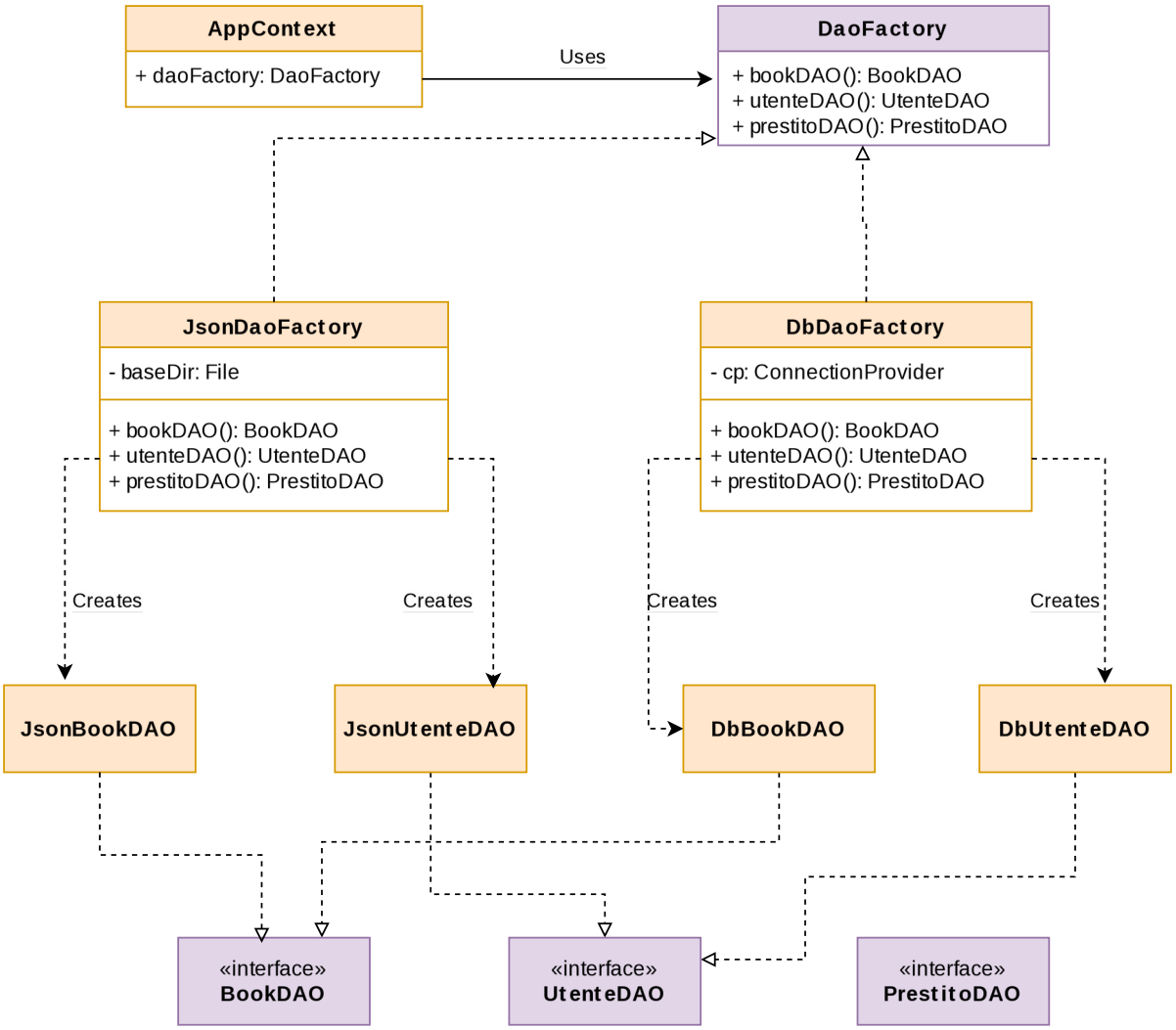
- 4.1 VOPC

VOPC -Registrazione Prestito

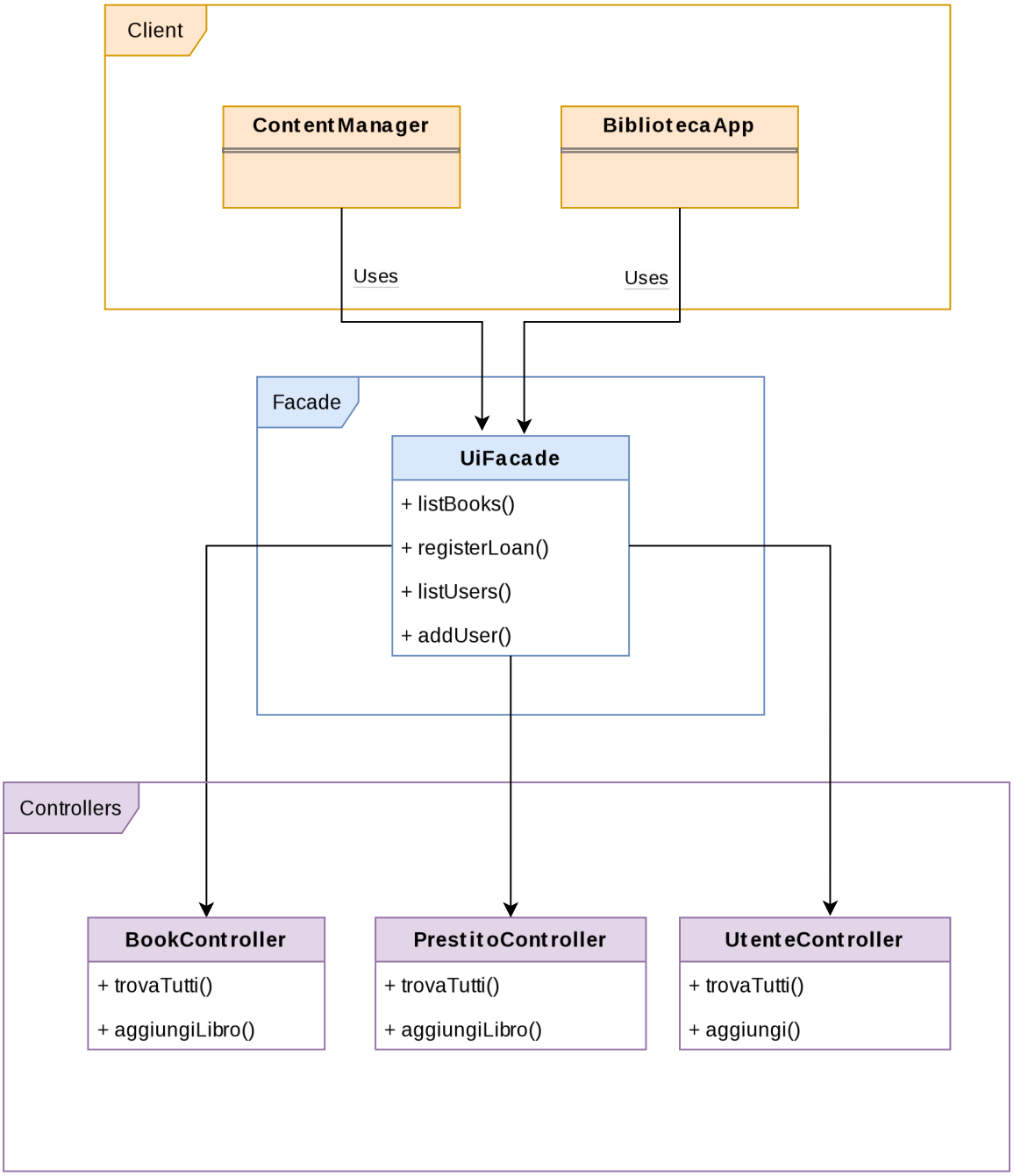


- 4.2 Design Diagrams

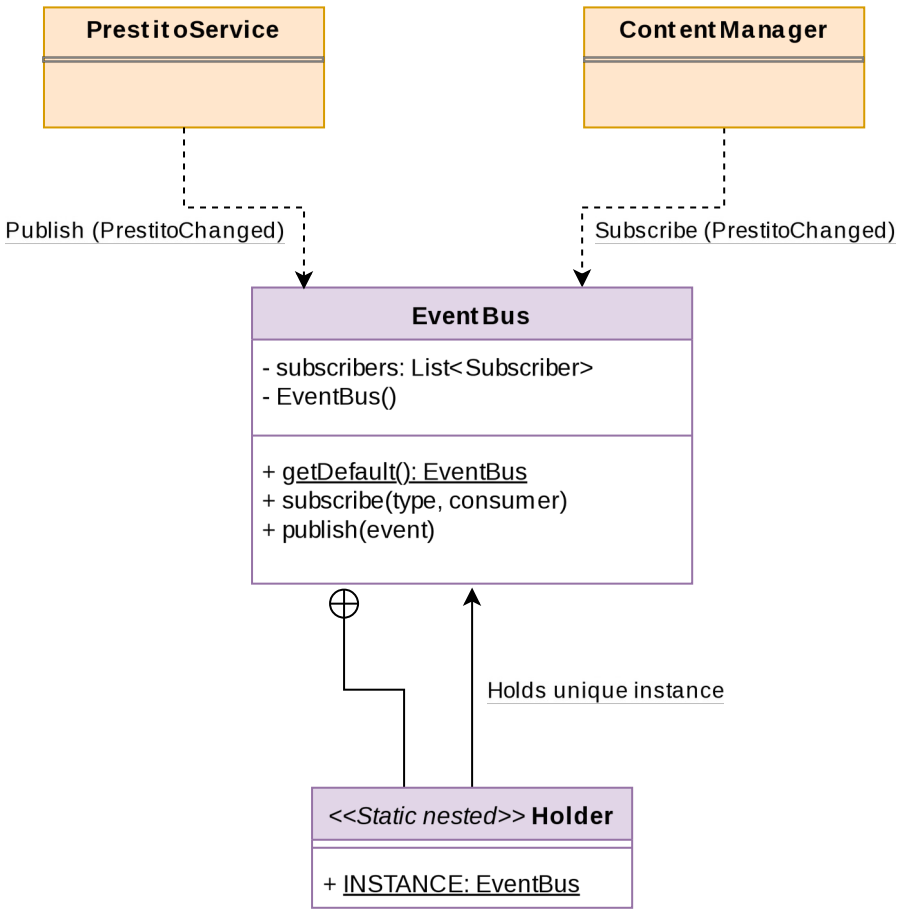
Design-Level: Abstract Factory



Design-level: Facade



Design-level: Singleton



- 4.3 Patterns

Nel progetto sono stati adottati diversi design pattern appartenenti alla catalogazione GoF, con l'obiettivo di migliorare la separazione delle responsabilità e ridurre l'accoppiamento tra i componenti.

Il primo pattern riconoscibile è il **Facade**, implementato dalla classe UiFacade. Essa fornisce un unico punto di accesso alle funzionalità applicative esposte verso l'interfaccia utente: le classi della UI non interagiscono direttamente con i singoli service o controller. In questo modo l'interfaccia grafica rimane isolata dai dettagli interni della logica di business, semplificando il codice lato UI e rendendo più agevoli eventuali modifiche alla struttura interna dei servizi senza impattare sulle viste.

Un secondo pattern GoF utilizzato è l'**Abstract Factory**, realizzato tramite la classe DaoFactory (e le sue implementazioni concrete). La

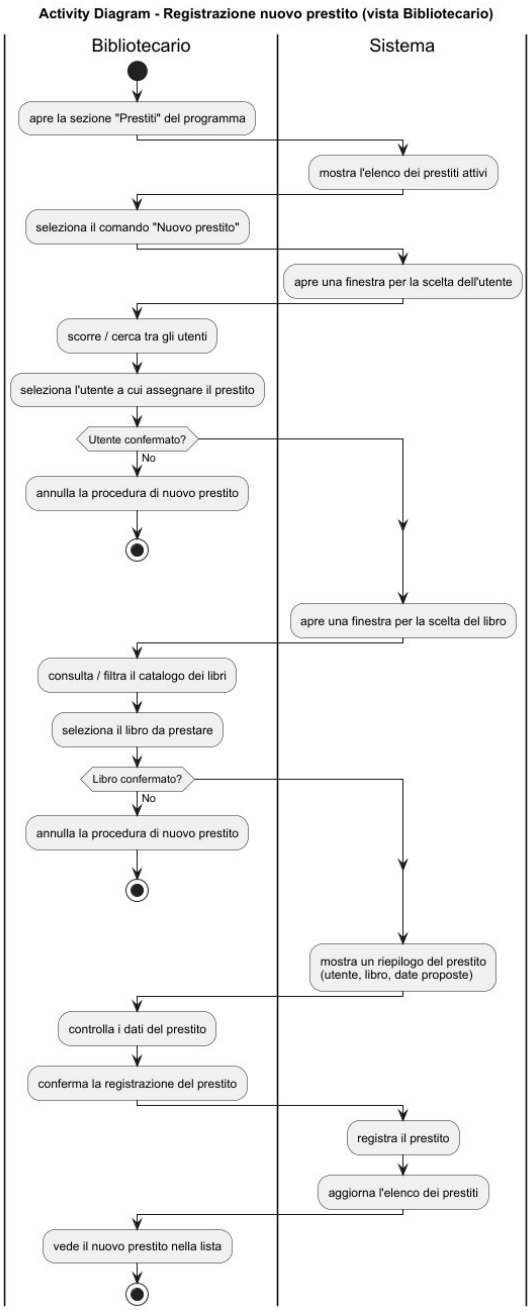
factory si occupa di creare le istanze dei vari DAO (BookDAO, PrestitoDAO, UtenteDAO), incapsulando la logica di costruzione e la scelta dell'implementazione concreta. Questo consente di cambiare la tecnologia di persistenza (ad esempio passando da un certo tipo di backend ad un altro) intervenendo in un unico punto, senza modificare il codice dei servizi che utilizzano i DAO. L'uso di una Abstract Factory riduce quindi la dipendenza diretta dai costruttori delle classi concrete e rende l'architettura più flessibile ed estendibile.

È inoltre possibile individuare l'adozione del pattern **Strategy** nella gestione della connessione al database, attraverso l'interfaccia ConnectionProvider e le sue diverse implementazioni. Il modo in cui viene ottenuta e configurata la connessione al DB è delegato ad oggetti intercambiabili che implementano la stessa interfaccia. Questo permette di sostituire agevolmente la strategia di connessione (ad esempio passando da una configurazione di test ad una di produzione) senza modificare il codice dei DAO o dei servizi che accedono ai dati. Il pattern Strategy facilita quindi la configurabilità del sistema e supporta scenari diversi (produzione, test, ecc.) con un impatto minimo sul codice applicativo.

Nel progetto è presente anche il pattern **Singleton**, implementato tramite la classe EventBus. EventBus rappresenta il punto unico di pubblicazione e sottoscrizione degli eventi applicativi: tutti i componenti che devono notificare o ricevere cambiamenti passano attraverso la stessa istanza condivisa. L'uso del Singleton evita la creazione di più bus di eventi scollegati tra loro e semplifica la gestione centralizzata delle notifiche, riducendo la necessità di passare esplicitamente riferimenti al bus tra le varie parti dell'applicazione.

- 4.4 Activity Diagram

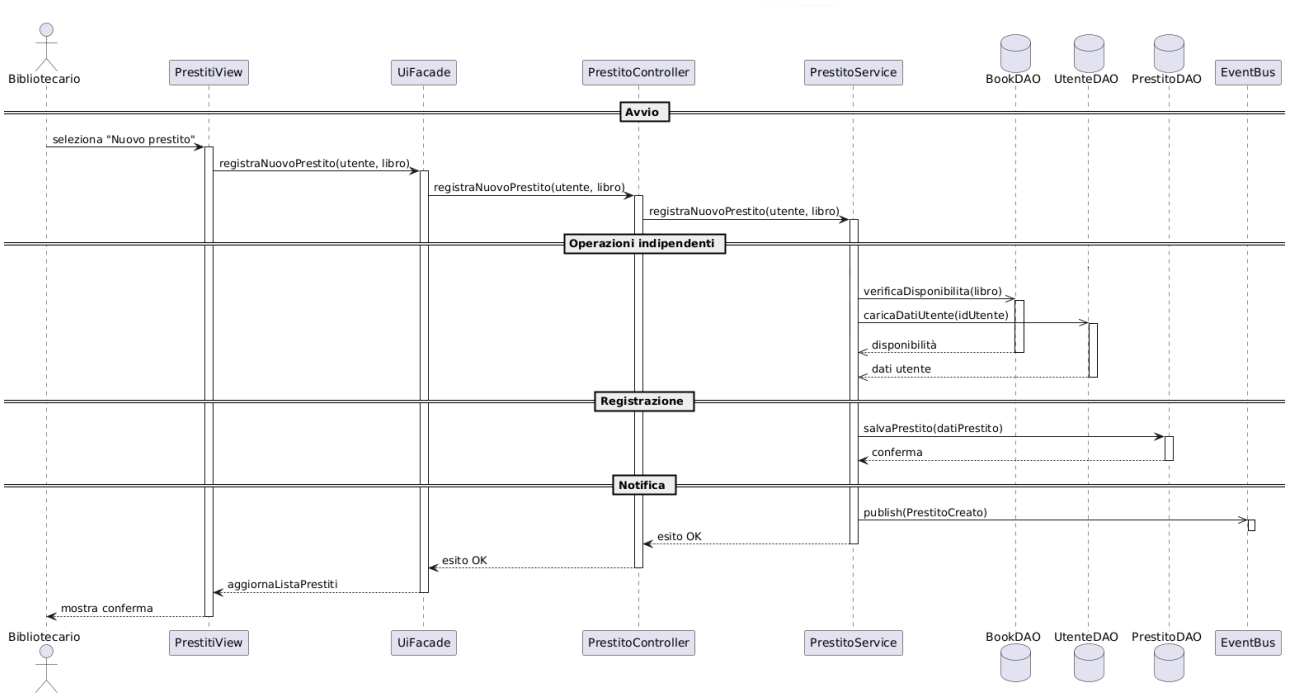
Essendoci multiple operazioni disponibili, ho scelto la registrazione di un nuovo prestito da parte del Bibliotecario.



Il grafico è anche disponibile come file (ActivityDiagramPrestito.png) per una migliore visualizzazione.

- 4.5 Sequence Diagram

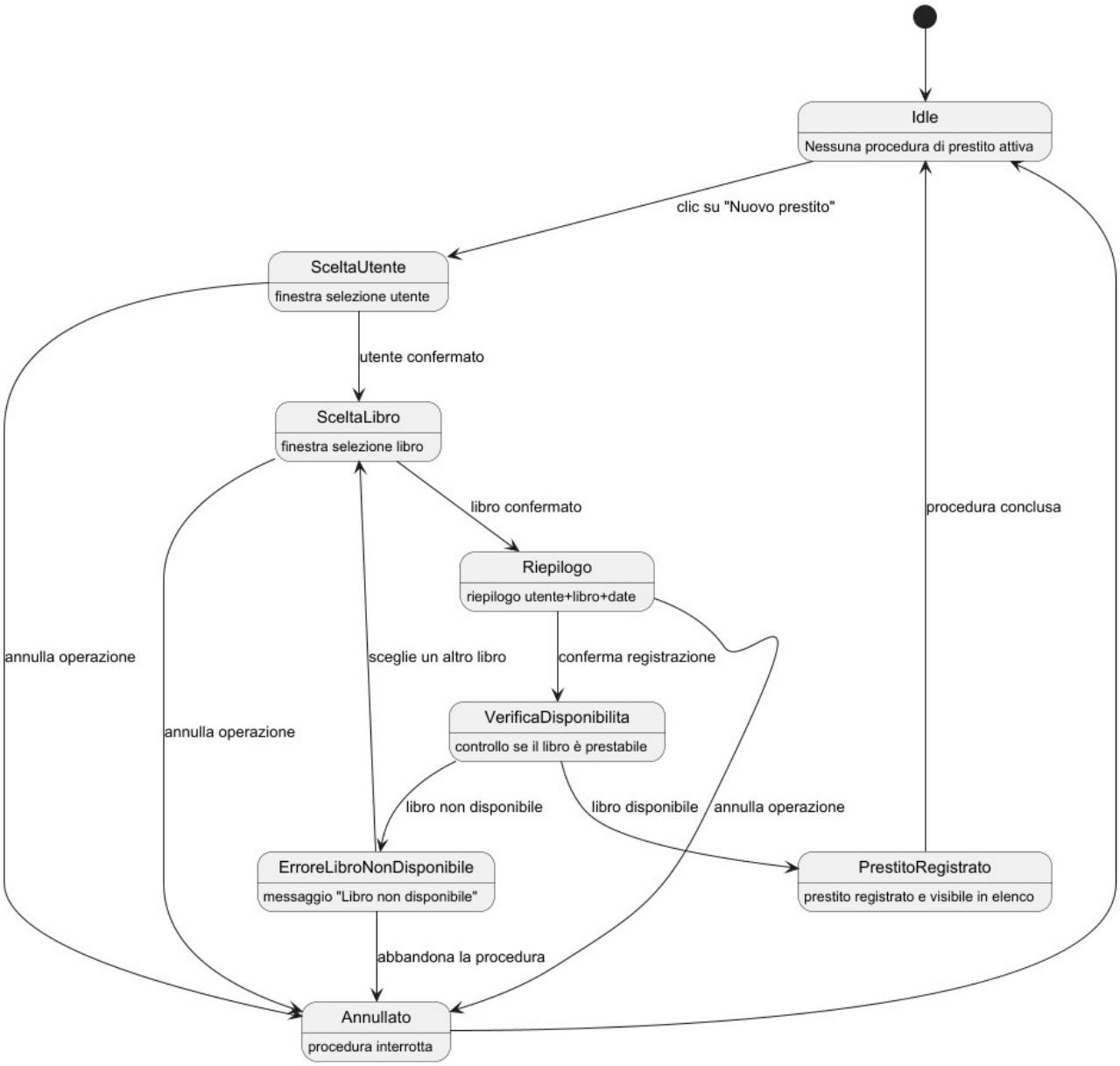
Di seguito il Sequence Diagram, sempre per l’attività di registrazione di un nuovo prestito da parte del Bibliotecario. Il diagramma è disponibile come file (SequenceDiagramPrestito.png) per una migliore visualizzazione.



- 4.6 State Diagram

Infine, lo State Diagram, di nuovo per la stessa attività di registrazione di un nuovo prestito da parte del Bibliotecario. Il diagramma è disponibile come file (StateDiagramPrestito.png) per una migliore visualizzazione.

State Diagram - Procedura "Nuovo prestito" (Bibliotecario)



5. Testing

Il progetto prevede una batteria di test automatici basati su JUnit 5, orientati principalmente alla verifica dell'accesso ai dati e dell'autenticazione.

Per la configurazione dell'ambiente di test sono state introdotte due classi di supporto nel package it.biblioteca.testutil:

- **TestConnectionProvider** implementa l'interfaccia ConnectionProvider e fornisce le connessioni JDBC al database di test tramite DriverManager.
- **TestDbSetup** contiene il metodo statico resetSchema(Connection), che si occupa di ricreare completamente lo schema di test "biblioteca": vengono droppate e ricreate le tabelle principali (utenti, libri, prestiti, credenziali, ecc.) e vengono inseriti alcuni dati iniziali (un amministratore, un bibliotecario, un utente normale, e almeno un libro di esempio). In questo modo ogni test parte da uno stato del database noto e ripetibile.

Su questa base sono stati definiti tre test principali:

1. **BookDaoDbTest** (package it.biblioteca.dao.db)

Questo test verifica il corretto funzionamento della classe DbBookDAO, che implementa l'interfaccia BookDAO. Nel metodo di setup (@BeforeEach) viene creata un'istanza di TestConnectionProvider e viene richiamato TestDbSetup.resetSchema per riportare il database allo stato iniziale prima di ogni test. Il test crudLibro_funzionante copre l'intero ciclo di vita di un libro:

- creazione di un oggetto Book con dati di esempio;
- lettura di tutti i libri presenti;
- aggiornamento del titolo e del numero di copie;
- eliminazione del libro

Questo test garantisce che le operazioni CRUD sul catalogo dei libri siano correttamente mappate sul database.

2. **PrestitoFlowDbTest** (package it.biblioteca.dao.db)

Questo test verifica il flusso completo di registrazione e chiusura di un prestito utilizzando l'implementazione DbPrestitoDAO.
Nel test registraEChiudiPrestito:

- viene creato un PrestitoBean che riferisce il libro e l'utente inseriti dal setup;
- si invoca dao.inserisci(bean) e si verifica che l'operazione vada a buon fine;
- si leggono tutti i prestiti (dao.trovaTutti()) controllando che ne esista uno solo;
- si verifica che il prestito risulti tra quelli attivi (dao.trovaPrestitiAttivi());
- si chiude il prestito tramite dao.chiudiPrestito(id, dataRestituzione);
- si controlla che, dopo la chiusura, non vi siano più prestiti attivi e che la data di restituzione memorizzata nel database corrisponda a quella specificata nel test.

Questo test assicura che il flusso di gestione dei prestiti (inserimento, ricerca, chiusura) sia coerente e correttamente persistito.

3. **AuthServiceDbTest** (package it.biblioteca.security)

Questo test verifica il corretto funzionamento del servizio di autenticazione AuthService in combinazione con il database di test. Sono definiti due casi di test principali:

- loginAdmin_ok_e_wrongPassword_fail: verifica che una chiamata AuthService.authenticate("admin", "admin") produca un risultato positivo (ok() = true) o negativo (ok() = false);
- loginUnknownUser_fail: verifica che il tentativo di autenticazione di un utente inesistente restituisca sempre un risultato negativo (ok() = false).

In questo modo viene validata sia la corretta integrazione con la tabella delle credenziali, sia la logica dell'esito di autenticazione.

6. Gestione delle eccezioni

La gestione delle eccezioni nel progetto è stata mantenuta intenzionalmente semplice: negli strati più bassi (accesso al database, operazioni CSV, ecc.) le eccezioni tecniche vengono intercettate tramite blocchi try/catch e convertite in valori di ritorno (ad esempio boolean o liste vuote) oppure in messaggi di errore da propagare verso i livelli superiori.
Nello strato applicativo e di interfaccia grafica, gli errori vengono mostrati all'utente tramite finestre di Alert e aggiornamenti della barra di stato, evitando la chiusura anomala dell'applicazione. Non è stata introdotta una gerarchia complessa di eccezioni personalizzate: si è preferito un approccio pragmatico, basato sulle eccezioni standard e sulla traduzione degli errori in feedback chiari per l'utente.

7. Persistenza e DB

L'applicazione supporta due modalità di persistenza dei dati. In questo modo la struttura logica dei dati rimane la stessa, ma cambia il supporto fisico e il livello di formalizzazione (file JSON vs database relazionale).

- 7.1 File/JSON

Nel backend a file, i dati sono salvati in una cartella (db/json) sotto forma di tre file JSON principali: un file per i libri (books.json), uno per gli utenti (utenti.json) e uno per i prestiti (prestiti.json).
Essi sono gestiti tramite le apposite classi inserite nella cartella dao/json.

- 7.2 MySQL

Nel backend MySQL, invece, le stesse informazioni sono organizzate in uno schema relazionale con tabelle distinte per utenti, libri, prestiti e credenziali di accesso, collegate tra loro tramite chiavi esterne. Il DB è configurabile automaticamente tramite il file Biblioteca.sql presente all'interno della cartella DB nella root del progetto.

8. Qualità del codice/Sonar/Rule Compliance

Il progetto, così come richiesto dalle specifiche, è stato testato dal sistema SonarQube Cloud. I code smells sono stati tutti risolti e i risultati della verifica sono disponibili al seguente link: https://sonarcloud.io/summary/new_code?id=VanackSabbadium_BibliotecaISPW&branch=main

9. Codice

Il codice del progetto è disponibile al repository GitHub: https://github.com/VanackSabbadium/BibliotecaISPW

10. Presentazione video

La presentazione video è disponibile al seguente URL, presente nel repository GitHub e scaricabile: https://github.com/VanackSabbadium/BibliotecaISPW/blob/main/BibliotecaISPW.mpeg