

UNIVERSITÀ DEGLI STUDI DI PARMA

Dipartimento di Informatica

Corso di Ingegneria del Software

Anno Accademico 2025/2026

Progetto 6: UniprBooks

Mercatino Libri Usati UNIPR

Piattaforma P2P per la compravendita di libri di testo universitari

Team T24

Studente	Matricola	Ruolo
Andrea Belli	361054	Database
Nna Minkousse Kenneth James	366361	Frontend
Samuele Premori	361939	Backend

Gennaio 2026

Indice

1	Introduzione	4
1.1	Il Team	4
1.2	Quick Start	4
1.3	Gestione Dati e Database	4
2	Processo di Sviluppo	5
2.1	Metodologia	5
2.1.1	A. Sprint Planning	5
2.1.2	B. Development Loop	5
2.1.3	C. Sprint Review	5
2.2	Organizzazione del Team	6
2.2.1	Database	6
2.2.2	Backend	7
2.2.3	Frontend	7
2.3	Branch Strategy	7
2.3.1	Branch Structure	7
2.3.2	Regole	7
2.4	Testing	7
3	Analisi dei Requisiti	8
3.1	Requisiti Non Funzionali	8
3.2	Requisiti Funzionali	8
3.3	Attori di Sistema Principali	8
3.3.1	Visitatore	8
3.3.2	Utente	9
3.4	User Stories	9
3.4.1	[US01] Registrazione Utente	9
3.4.2	[US02] Login Utente	9
3.4.3	[US03] Visualizzazione	9
3.4.4	[US04] Pubblicazione Annuncio	9
3.4.5	[US05] Acquisto Diretto	9
3.4.6	[US06] Ricerca Avanzata	10
3.4.7	[US07] Messaggistica tra Utenti	10
3.5	Matrice di Tracciabilità	10
4	Architettura del Sistema	11
4.1	Pattern MVP	11
4.1.1	View	11
4.1.2	Presenter	11
4.1.3	Model	11
4.2	Schema Dati	11
4.2.1	Modello Entità-Relazione	11
4.2.2	Interfaccia al Database	12
4.3	Sicurezza	15

4.3.1	Hashing delle Password	15
4.3.2	JSON Web Token (JWT)	15
4.3.3	Messaggi di Errore	16
4.3.4	Validazione Input	16
4.4	Architettura Frontend Presenter	17
4.4.1	ApiService (Servizio Statico)	17
4.4.2	AuthPresenter	17
4.4.3	BooksPresenter	18
4.4.4	ChatPresenter	18
4.4.5	User (Model)	18
4.4.6	Relazioni	18
5	Contratti di Interfaccia	20
5.1	Specifiche API	20
5.1.1	POST /register	20
5.1.2	POST /login	21
5.1.3	GET /books	21
5.1.4	GET /my-books	22
5.1.5	POST /books	22
5.1.6	PUT /books	23
5.1.7	DELETE /books	24
5.1.8	POST /purchase	25
5.1.9	GET /purchases	25
5.1.10	GET /sales	26
5.1.11	GET /conversations	27
5.1.12	GET /messages?userId={id}	27
5.1.13	POST /messages	28
5.2	Interfaccia Database	28
5.2.1	Entity Classes	28
5.2.2	Metodi Gestione Utenti	29
5.2.3	Metodi Gestione Libri	29
5.2.4	Metodi Gestione Transazioni	30
5.2.5	Metodi Gestione Messaggi	30
6	Sprint Log	31
6.1	Sprint 1	31
6.1.1	Task	31
6.2	Sprint 2	31
6.2.1	Task	31
7	Use Cases	32
7.1	UC01 - Registrazione Utente	32
7.1.1	Panoramica	32
7.1.2	Flussi di Eventi	32
7.1.3	Activity Diagram	34
7.1.4	Criteri di Accettazione e Testing	34
7.1.5	Specifiche Tecniche	35
7.2	UC02 - Login Utente	37
7.2.1	Panoramica	37
7.2.2	Flussi di Eventi	38
7.2.3	Activity Diagram	39
7.2.4	Criteri di Accettazione	39

7.2.5	Piano di Test Manuale	40
7.2.6	Design Tecnico	40
7.3	UC03 - Visualizzazione Lista Libri	42
7.3.1	Panoramica	42
7.3.2	Flussi di Eventi	43
7.3.3	Activity Diagram	44
7.3.4	Criteri di Accettazione	44
7.3.5	Piano di Test Manuale	45
7.3.6	Design Tecnico	45
7.4	UC04 - Pubblicazione Annuncio	47
7.4.1	Panoramica	47
7.4.2	Flussi di Eventi	47
7.4.3	Activity Diagram	49
7.4.4	Criteri di Accettazione	50
7.4.5	Piano di Test Manuale	50
7.4.6	Design Tecnico	51
7.5	UC05 - Acquisto Diretto	53
7.5.1	Panoramica	53
7.5.2	Flussi di Eventi	54
7.5.3	Activity Diagram	55
7.5.4	Criteri di Accettazione	55
7.5.5	Piano di Test Manuale	56
7.5.6	Design Tecnico	56
7.6	UC06 - Ricerca Avanzata	58
7.6.1	Panoramica	58
7.6.2	Flussi di Eventi	58
7.6.3	Criteri di Accettazione	59
7.6.4	Piano di Test Manuale	59
7.6.5	Design Tecnico	60
7.7	UC07 - Messaggistica tra Utenti	60
7.7.1	Panoramica	60
7.7.2	Flussi di Eventi	61
7.7.3	Activity Diagram	62
7.7.4	Criteri di Accettazione	62
7.7.5	Piano di Test Manuale	63
7.7.6	Design Tecnico	63

Capitolo 1

Introduzione

1.1 Il Team

Studente	Matricola	Ruolo Principale
Andrea Belli	361054	Database
Nna Minkousse Kenneth James	366361	Frontend
Samuele Premori	361939	Backend

Tabella 1.1: Composizione del Team T24

1.2 Quick Start

1. Clonare il repository
2. Configurare l'ambiente

```
1 cp .env.example .env  
2
```

3. Avviare i container

```
1 docker-compose up -d  
2
```

4. Accedere all'applicazione

- Frontend: <http://localhost:8080>
- Backend API: <http://localhost:8081>

1.3 Gestione Dati e Database

- Rimuovere il Volume:

```
1 docker-compose down -v  
2
```

- Avvio Build forzata Immagine:

```
1 docker-compose up -d --build  
2
```

Capitolo 2

Processo di Sviluppo

2.1 Metodologia

Il processo adottato è **Iterativo** ed **Incrementale**. Ogni Sprint trasforma un sottoinsieme di Requisiti in un incremento software funzionante, testato e documentato.

2.1.1 A. Sprint Planning

- **Analisi dei Requisiti:** Revisione dei requisiti ed eventuale aggiornamento.
- **Selezione User Stories:** Spostamento delle user stories nello Sprintlog.
- **Analisi:** Le user stories selezionate vengono dettagliate in `docs/features/` tramite:
 - Lo Use Case Diagram, con tabella Use Case per dettaglio ed eventuali Activity Diagram.
 - I Criteri di Accettazione.
 - La strategia per i Test di Integrazione manuali.
- **Design Session:** Definizione dei contratti:
 - Specifica JSON delle API.
 - Specifica metodi interfaccia database.
- **Task Assignment:** Suddivisione dei compiti, creazione dei branch `feature/` e creazione della issue relativa alla user story.

2.1.2 B. Development Loop

- **Detailed Design:** Prima della codifica, ogni sviluppatore progetta il proprio componente nella cartella della feature:
 - Diagrammi delle Classi.
 - Diagrammi ER specifici.
- **Coding & Unit Testing:** Scrittura del codice sorgente e degli eventuali Unit Test.
- **Pull Request:** Lo sviluppatore apre una PR verso il ramo di integrazione (`develop`).
- **Merge:** Se la review è positiva e i test passano, il codice viene integrato in `develop`.

2.1.3 C. Sprint Review

- **Integration Test:** Verifica completa del flusso basata sui **Criteri di Accettazione**.

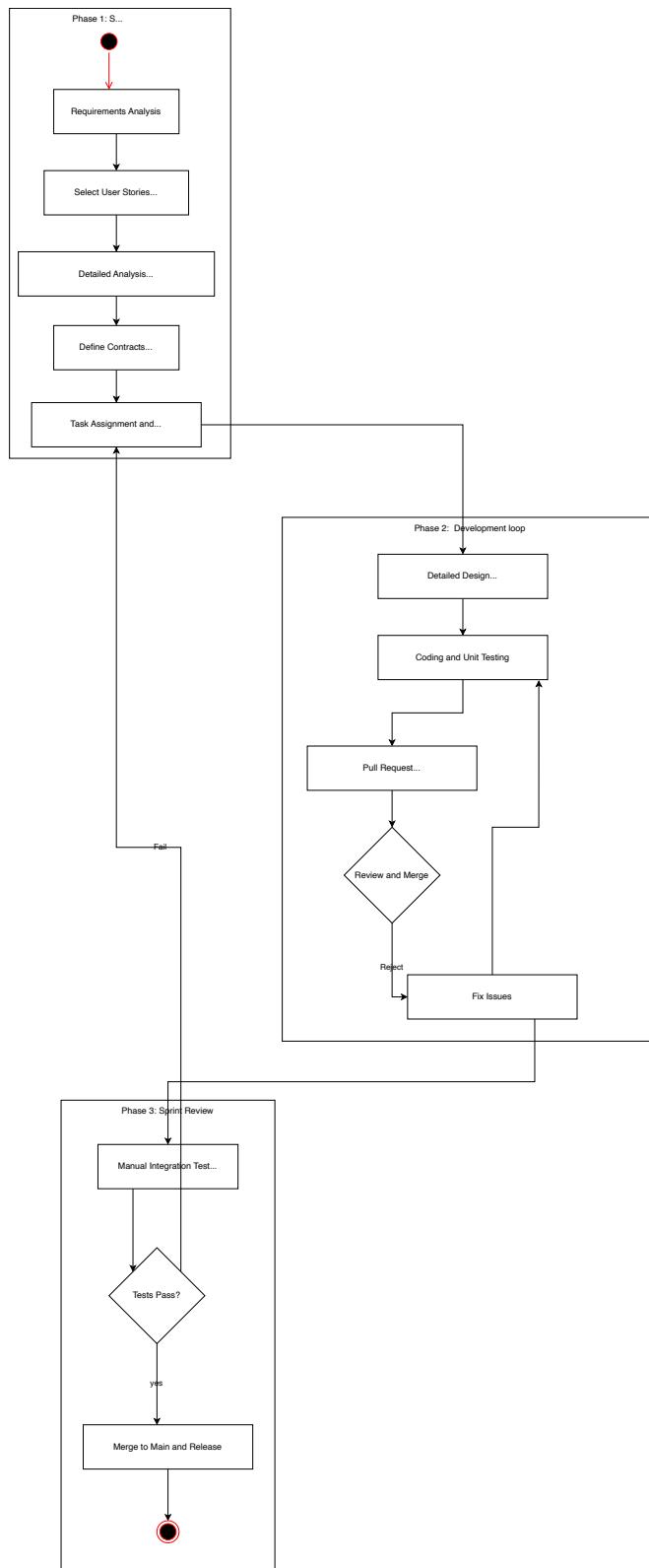


Figura 2.1: Activity Diagram del Processo di Sviluppo

2.2 Organizzazione del Team

2.2.1 Database

- **Responsabile:** Andrea Belli

- **Responsabilità:** gestione Database, gestione classe interfaccia Database.

2.2.2 Backend

- **Responsabile:** Samuele Premori
- **Responsabilità:** Logica business, esposizione API, test di unità.

2.2.3 Frontend

- **Responsabile:** Nna Minkousse Kenneth James
- **Responsabilità:** Interfaccia utente, integrazione API, test manuali.

2.3 Branch Strategy

2.3.1 Branch Structure

- **main:** codice stabile e testato
- **dev:** ramo di integrazione continua delle feature
- **feature/T{ID}:** ramo riguardante la task specifica, singolo per sviluppatore.

2.3.2 Regole

- Vietato il push diretto su **main**.
- Il push diretto su **dev** avviene solo per modifiche della documentazione o modifiche tra gli sprint approvate in maniera unanime.
- Ogni ramo **feature** si chiude con un pull request su **dev**.
- Ogni pull request da **dev** a **main** deve contenere la documentazione aggiornata e il tag relativo alla versione.

2.4 Testing

La strategia di testing è divisa in:

- **Unit Testing:**
 - Applicato alle classi del Backend.
 - Obbligatorio passare i test prima della PR.
 - Framework: PHPUnit.
- **Integration Testing:**
 - Eseguito manualmente seguendo i file di collaudo compilati a inizio sprint.

Capitolo 3

Analisi dei Requisiti

Scopo del Progetto: Applicazione Web per la compravendita di libri universitari.

- Non verrà implementato un sistema di pagamento.
- Non verrà implementato un meccanismo di password recovery.
- La gestione per accordarsi sul prezzo, su un diverso pagamento e sulla consegna del libro, sono attualmente lasciate alla gestione tra utenti.

3.1 Requisiti Non Funzionali

- **RNF01:** Implementazione dell'architettura attraverso il pattern Model-View-Presenter (MVP).
- **RNF02:** Backend implementato in PHP puro.
- **RNF03:** Frontend implementato in HTML, CSS e Javascript.
- **RNF04:** Il frontend comunica con il backend solo tramite API RESTful CRUD.
- **RNF05:** Il Backend non deve generare codice HTML.
- **RNF06:** Le password non devono essere salvate in chiaro.

3.2 Requisiti Funzionali

- **RF01:** Login e Registrazione degli utenti.
- **RF02:** Visualizzazione lista libri.
- **RF03:** Caricamento annuncio.
- **RF04:** Acquisto diretto.
- **RF05:** Ricerca avanzata per ISBN, corso, docente.
- **RF06:** Messaggistica tra utenti.

3.3 Attori di Sistema Principali

3.3.1 Visitatore

Utente non autenticato, può:

1. Visualizzare tutti gli annunci.
2. Effettuare il login.

3. Effettuare la registrazione.

3.3.2 Utente

Utente autenticato, può:

1. Visualizzare annunci pubblicati da altri.
2. Aggiungere, modificare e rimuovere i propri annunci.
3. Visualizzare lo storico dei libri venduti.
4. Visualizzare lo storico degli acquisti.
5. Effettuare il logout.

3.4 User Stories

3.4.1 [US01] Registrazione Utente

*Come Visitatore,
Voglio registrarmi,
Affinché il sistema mi permetta di accedere.*

Analisi: UC01 (Capitolo 7)

3.4.2 [US02] Login Utente

*Come Visitatore,
Voglio effettuare il login,
Affinché il sistema mi permetta di compiere operazioni.*

Analisi: UC02 (Capitolo 7)

3.4.3 [US03] Visualizzazione

*Come Utente o Visitatore,
Voglio vedere la lista dei libri (esclusi i miei),
Affinché possa trovare testi da acquistare.*

Analisi: UC03 (Capitolo 7)

3.4.4 [US04] Pubblicazione Annuncio

*Come Utente,
Voglio caricare un libro (Titolo, Prezzo, Foto, ISBN, Docente, Corso),
Affinché sia visibile e acquistabile dagli altri utenti.*

Analisi: UC04 (Capitolo 7)

3.4.5 [US05] Acquisto Diretto

*Come Utente,
Voglio poter acquistare un libro,
Affinché il libro vada nello storico dei miei acquisiti, nello storico annunci del venditore e spariscia dalla bacheca.*

Analisi: UC05 (Capitolo 7)

3.4.6 [US06] Ricerca Avanzata

*Come Utente o Visitatore,
Voglio poter cercare libri per ISBN, corso o docente,
Affinché possa trovare rapidamente i testi di mio interesse.*

Analisi: UC06 (Capitolo 7)

3.4.7 [US07] Messaggistica tra Utenti

*Come Utente,
Voglio poter inviare e ricevere messaggi da altri utenti,
Affinché possa accordarmi su prezzo e consegna dei libri.*

Analisi: UC07 (Capitolo 7)

3.5 Matrice di Tracciabilità

	RF01	RF02	RF03	RF04	RF05	RF06
UC01	X					
UC02		X				
UC03			X			
UC04				X		
UC05					X	
UC06						X
UC07						X

Tabella 3.1: Matrice di Tracciabilità Use Cases - Requisiti Funzionali

Capitolo 4

Architettura del Sistema

4.1 Pattern MVP

Il sistema adotta un'architettura **Model-View-Presenter (MVP)**.

4.1.1 View

- Definisce la struttura della pagina e gli stili.
- Espone gli elementi del DOM (bottoni, form) che vengono manipolati dal Presenter.

4.1.2 Presenter

- Intercetta gli eventi utente dalla View.
- Invoca il Model remoto (tramite Fetch API).
- Riceve i dati (JSON) e aggiorna la View manipolando il DOM.
- Gestisce la logica di presentazione.

4.1.3 Model

- Rappresenta lo stato del sistema e la Business Logic pura.
- Espone un'interfaccia RESTful (JSON).
- Riceve comandi dal Presenter.
- Valida le regole di business (Sicurezza, Integrità).
- Restituisce i dati aggiornati o errori.

4.2 Schema Dati

Questa sezione descrive la modellazione dei dati del sistema.

4.2.1 Modello Entità-Relazione

Il modello logico dei dati si basa su quattro entità principali: **User**, **Book**, **Transaction** e **Message**.

Le principali scelte progettuali includono:

- **Integrità Referenziale:** L'uso di chiavi esterne (`seller_id`) con vincolo `ON DELETE CASCADE` assicura che alla rimozione di un utente vengano rimossi automaticamente anche i suoi annunci, prevenendo dati orfani.

- Sicurezza:** Le password non vengono salvate in chiaro, ma sotto forma di hash.
- Tipi di dato:** Per il prezzo è stato scelto il tipo DECIMAL per garantire precisione nelle operazioni monetarie, evitando gli errori di arrotondamento tipici dei numeri in virgola mobile (FLOAT).
- Storico Prezzi:** Nella tabella transactions viene salvato una copia del prezzo al momento dell'acquisto. Questo garantisce che lo storico ordini rimanga corretto anche se il prezzo originale del libro venisse modificato in futuro.

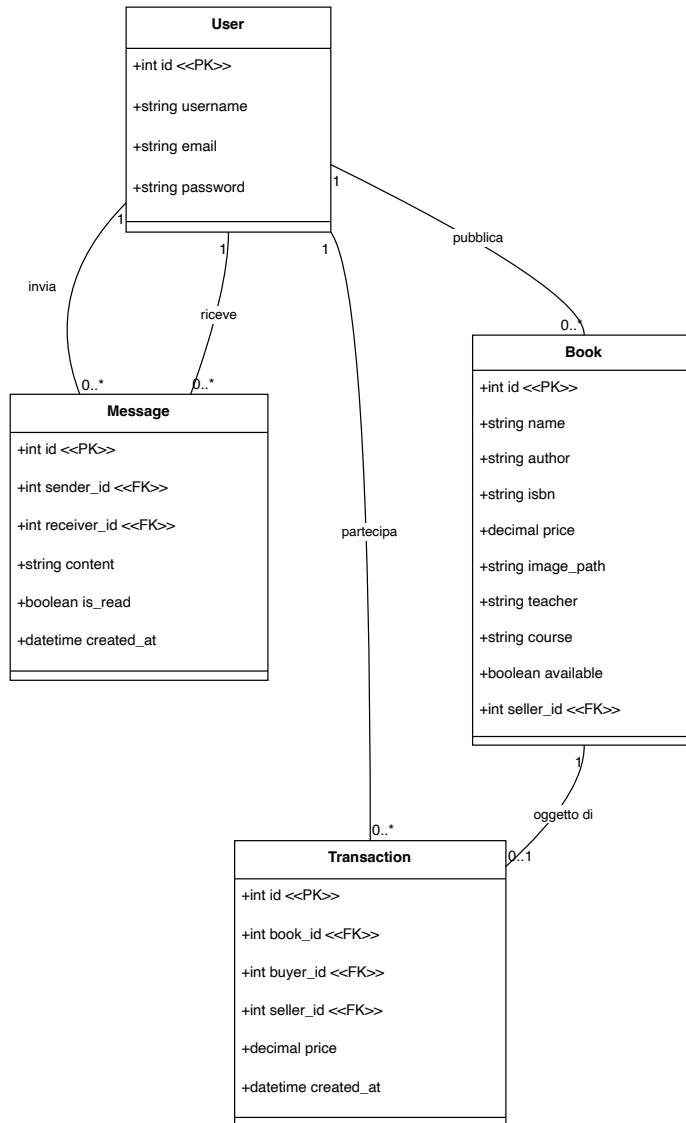


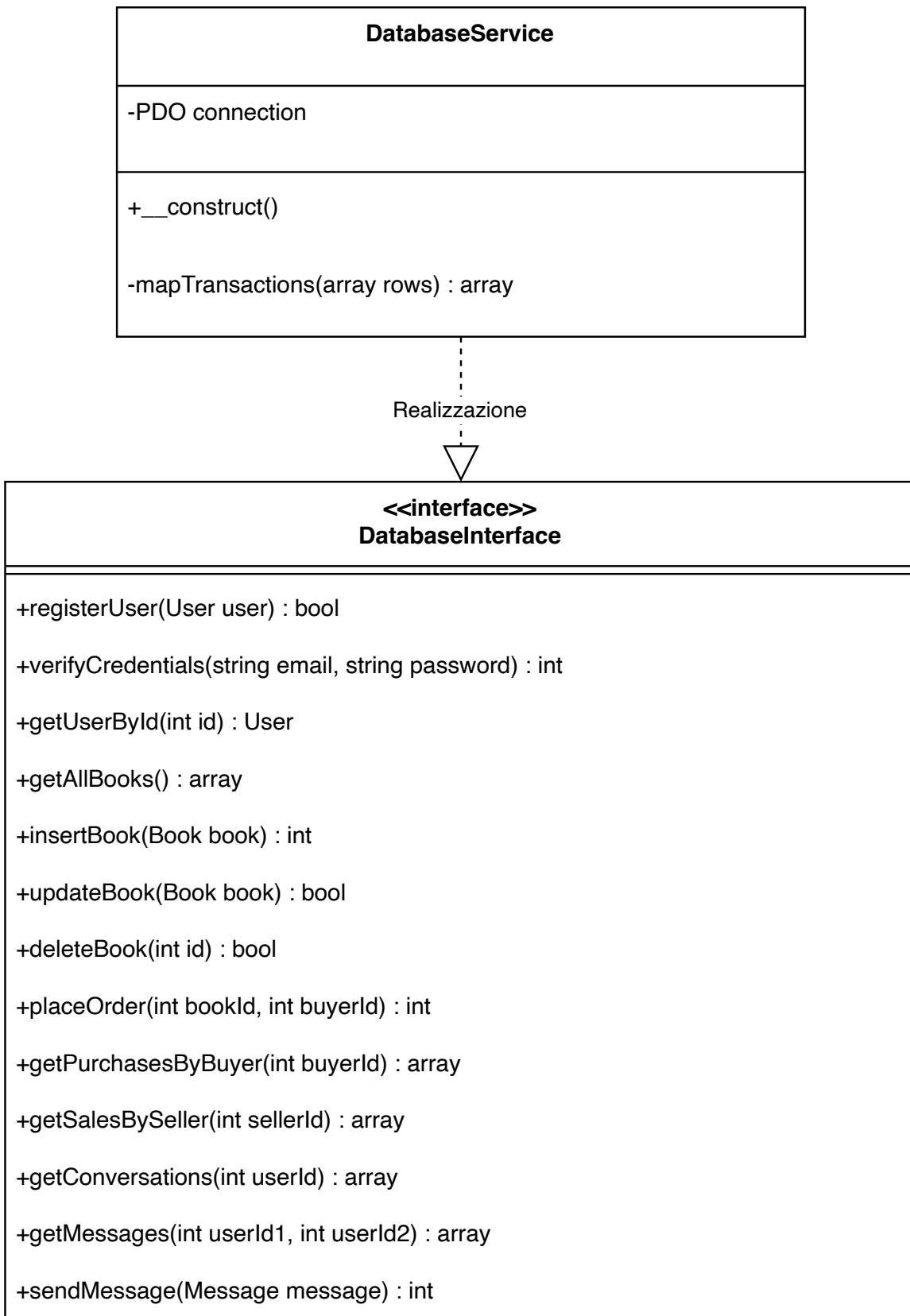
Figura 4.1: Diagramma Entità-Relazione

4.2.2 Interfaccia al Database

Per la gestione della persistenza è stato adottato il design pattern **Data Access Object (DAO)**, con l'obiettivo di isolare completamente la logica di accesso ai dati dal resto dell'applicazione. Questa scelta permette ai Controller di interagire con il database attraverso metodi ad alto livello, ignorando i dettagli implementativi del linguaggio SQL o del driver specifico utilizzato.

L'architettura si articola su due componenti principali:

- **DatabaseInterface (Contratto):** Un'interfaccia PHP che definisce formalmente le operazioni CRUD (Create, Read, Update, Delete) e di autenticazione disponibili nel sistema.
- **DatabaseService (Implementazione):** La classe concreta che realizza l'interfaccia utilizzando l'estensione **PDO (PHP Data Objects)**. Questa classe incapsula internamente la connessione al database, gestita in modo sicuro tramite variabili d'ambiente per evitare l'esposizione di credenziali nel codice sorgente.

**Figura 4.2:** Class Diagram - Database Interface

- Encapsulation:** Il driver PDO è dichiarato come proprietà privata all'interno del **DatabaseService**, impedendo accessi non autorizzati alla connessione dall'esterno della classe.
- Prepared Statements:** Ogni interazione con il database avviene tramite l'uso di query

preparate; questa pratica è essenziale per neutralizzare il rischio di attacchi **SQL Injection**, garantendo che i dati di input (come email o ISBN) siano trattati correttamente dal driver.

3. **Iniezione delle Dipendenze:** L'istanza di `DatabaseService` viene creata centralmente nel router (`index.php`) e passata ai controller tramite i loro costruttori. Questo approccio facilita la manutenzione e assicura che l'intera richiesta HTTP utilizzi una singola connessione persistente.
4. **Data Mapping:** Il servizio si occupa di trasformare i risultati grezzi delle query (array associativi) in oggetti di tipo `Book` o `User`, permettendo ai controller di operare su entità tipizzate e coerenti con il dominio dell'applicazione.

4.3 Sicurezza

Questa sezione definisce le specifiche di sicurezza per l'autenticazione e l'autorizzazione del sistema.

4.3.1 Hashing delle Password

Le password degli utenti non vengono mai salvate in chiaro nel database (RNF06).

Aspetto	Specifiche
Funzione	<code>password_hash()</code> di PHP
Algoritmo	<code>PASSWORD_BCRYPT</code>
Verifica	<code>password_verify()</code> (gestita dall'interfaccia DB)

Nota: *L'hashing viene eseguito nel backend (`AuthController`) prima di passare l'oggetto `User` all'interfaccia database.*

4.3.2 JSON Web Token (JWT)

Il sistema utilizza JWT per la gestione delle sessioni utente e l'autorizzazione agli endpoint protetti.

Aspetto	Specifiche
Libreria	<code>firebase/php-jwt</code>
Algoritmo	HS256 (HMAC-SHA256)
Chiave Segreta	Variabile d'ambiente <code>JWT_SECRET</code>
Scadenza Token	8 ore (28800 secondi)

Struttura del Token:

```
1 Header.Payload.Signature
```

Payload del Token:

```
1 {
2   "sub": 123,
3   "email": "utente@esempio.it",
4   "iat": 1737450000,
5   "exp": 1737478800
6 }
```

Campo	Descrizione
sub	ID dell'utente (subject)
email	Email dell'utente
iat	Timestamp di emissione (issued at)
exp	Timestamp di scadenza (expiration)

4.3.3 Messaggi di Errore

Per prevenire attacchi di enumerazione utenti, i messaggi di errore relativi all'autenticazione sono generici:

Scenario	Messaggio Restituito
Email non registrata	“Invalid credentials”
Password errata	“Invalid credentials”
Token mancante/invalido	“Unauthorized”
Token scaduto	“Unauthorized”

Nota: *Non viene mai rivelato se un'email è registrata o meno nel sistema.*

4.3.4 Validazione Input

Campo	Regole di Validazione
email	Formato email valido, non vuoto
username	Non vuoto
password	Minimo 8 caratteri

4.4 Architettura Frontend Presenter

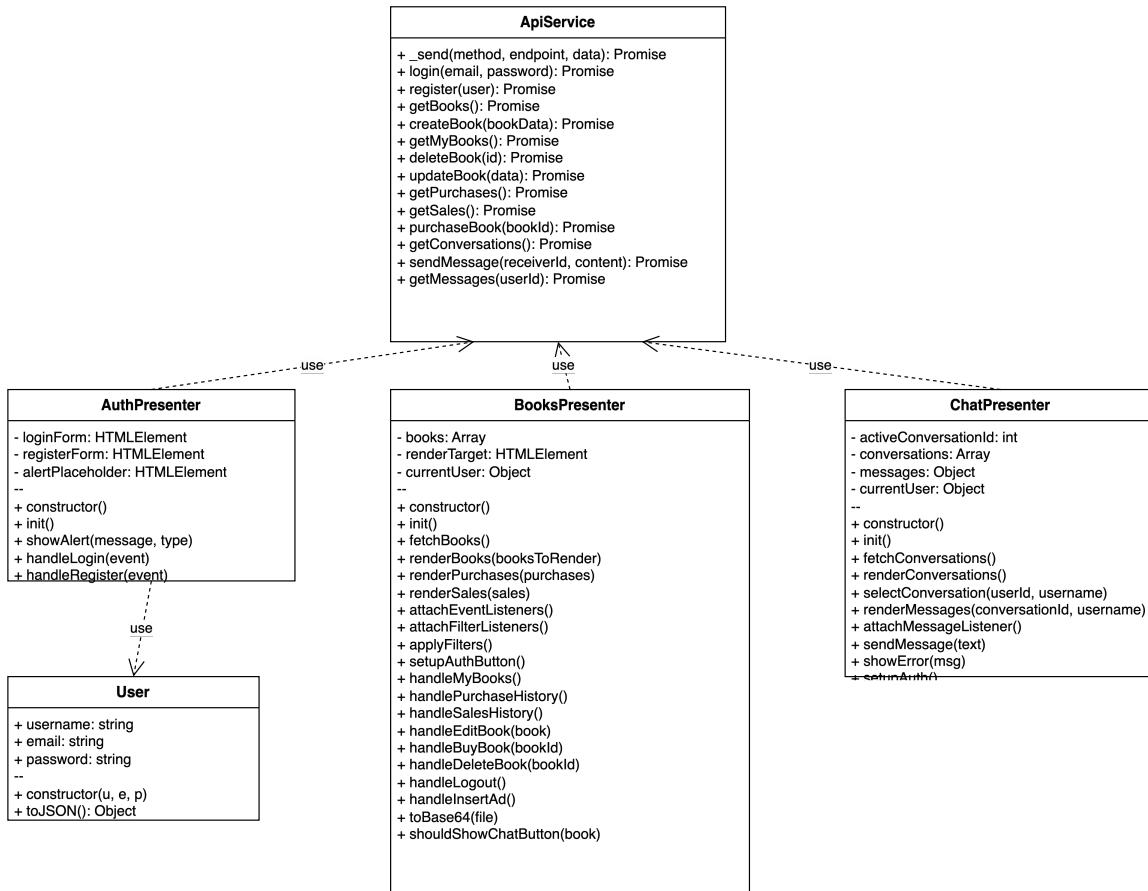


Figura 4.3: Class Diagram dei Presenter Frontend

Questa sezione spiega la struttura e le relazioni delle classi frontend come rappresentate nel diagramma delle classi. L'architettura segue una variante del pattern **Model-View-Presenter (MVP)**, dove i “Presenter” gestiscono la logica UI e la comunicazione con le API, mentre la “View” viene manipolata direttamente tramite il DOM.

4.4.1 ApiService (Servizio Statico)

Ruolo: Hub centrale di comunicazione per l'intero frontend.

- **Natura:** Consiste in metodi statici che gestiscono le richieste AJAX (`$.ajax`).
- **Responsabilità:**
 - Gestisce il Token JWT (aggiungendo l'header `Authorization`).
 - Espone gli endpoint per Login, Registrazione, operazioni sui Libri (Get, Create, Update, Delete), Acquisti e Messaggistica.
 - **Astrazione:** Nasconde i dettagli HTTP sottostanti ai Presenter.

4.4.2 AuthPresenter

Ruolo: Gestisce i flussi di Autenticazione Utente.

- **Responsabilità principali:**

- Ascolta i form di Login e Registrazione.
- Chiama `ApiService.login()` e `ApiService.register()`.
- **Gestione JWT:** Analizza il token JWT ricevuto per estrarre le informazioni utente (`parseJwt`) e lo memorizza nel `localStorage`.
- Reindirizza l'utente alla dashboard in caso di successo.

4.4.3 BooksPresenter

Ruolo: Controller principale per l'interfaccia del Marketplace dei Libri.

- **Responsabilità principali:**

- **Dashboard:** Recupera e renderizza tutti i libri specifici (`fetchBooks`, `renderBooks`).
- **Filtri:** Implementa la logica di filtraggio (Strategy Pattern con `CompositeFilter`, `GeneralSearchFilter`, ecc.) per cercare libri per titolo, ISBN, docente o corso.
- **I Miei Libri:** Visualizza i libri appartenenti all'utente loggato (`handleMyBooks`).
- **Operazioni CRUD:** Gestisce la creazione (`handleInsertAd`), l'aggiornamento del prezzo (`handleEditBook`) e l'eliminazione (`handleDeleteBook`) degli annunci.
- **Commercio:** Gestisce il processo di acquisto (`handleBuyBook`) e visualizza lo Storico Acquisti (`renderPurchases`) e lo Storico Vendite (`renderSales`).
- **Navigazione:** Gestisce il menu dropdown utente incluso il logout.

4.4.4 ChatPresenter

Ruolo: Gestisce il Sistema di Messaggistica.

- **Responsabilità principali:**

- **Conversazioni:** Recupera e lista le conversazioni attive (`fetchConversations`).
- **Messaggistica:** Visualizza lo storico chat (`renderMessages`) e invia nuovi messaggi (`sendMessage`).
- **Deep Links:** Gestisce i parametri URL (es. `?userId=X`) per aprire una chat specifica direttamente dalla card di un libro.
- **Aggiornamenti UI:** Formatta i timestamp e distingue tra messaggi inviati e ricevuti.

4.4.5 User (Model)

Ruolo: Semplice Data Transfer Object (DTO).

- **Responsabilità:**

- Incapsula i dati di registrazione utente (`username`, `email`, `password`) prima dell'invio all'API.

4.4.6 Relazioni

- **Dipendenza:** Tutti i Presenter (`AuthPresenter`, `BooksPresenter`, `ChatPresenter`) dipendono da `ApiService` per eseguire le operazioni di rete.
- **Associazione:** `AuthPresenter` utilizza la classe `User` per strutturare i dati di registrazione.

- **Flusso:**

1. BooksPresenter (Book Card) → Richiesta alla Chat (`window.location.href`).
2. ChatPresenter si inizializza → Controlla i parametri URL → Apre la Conversazione.

Capitolo 5

Contratti di Interfaccia

5.1 Specifiche API

In questa sezione vengono definiti gli endpoint del sistema. Tutte le richieste e le risposte utilizzano il formato **JSON**. Per le operazioni di scrittura e protezione dei dati, il sistema adotta lo standard **JWT** (JSON Web Token).

Nota: *Autenticazione:* Gli endpoint che richiedono autorizzazione devono includere l'header:
Authorization: `Bearer <token_jwt>`

Nota: *Risposta Errore Autorizzazione:* Se il token è mancante, invalido o scaduto, gli endpoint protetti restituiscono:

```
1 {
2   "status": "error",
3   "message": "Unauthorized"
4 }
```

5.1.1 POST /register

Registrazione di un nuovo utente.

Request Body:

```
1 {
2   "email": "utente@esempio.it",
3   "username": "mario_rossi",
4   "password": "password"
5 }
```

Response (Success):

```
1 {
2   "status": "success",
3   "message": "User registered successfully"
4 }
```

Response (Error - Input non valido):

```
1 {
2   "status": "error",
3   "message": "Invalid input"
4 }
```

5.1.2 POST /login

Autenticazione e rilascio del token.

Request Body:

```

1 {
2   "email": "utente@esempio.it",
3   "password": "password"
4 }
```

Response (Success):

```

1 {
2   "status": "success",
3   "message": "Login successful",
4   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
5 }
```

Response (Error - Input non valido):

```

1 {
2   "status": "error",
3   "message": "Invalid input"
4 }
```

Response (Error - Credenziali errate):

```

1 {
2   "status": "error",
3   "message": "Invalid credentials"
4 }
```

Nota: Sicurezza: Il messaggio “*Invalid credentials*” è volutamente generico per prevenire attacchi di enumerazione utenti. Non viene mai rivelato se l’email è registrata o meno.

5.1.3 GET /books

Recupero della lista di tutti i libri disponibili.

Auth: Opzionale (JWT)

Nota: Filtro Identità: Se l’utente è autenticato (JWT valido), il sistema esclude automaticamente i libri pubblicati dall’utente stesso dalla lista restituita.

Response (Success):

```

1 {
2   "status": "success",
3   "data": [
4     {
5       "id": 1,
6       "name": "Ingegneria del Software",
7       "author": "Ian Sommerville",
8       "isbn": "978-8871926284",
9       "imagePath": "data:image/jpeg;base64,/9j/4AAQ...",
10      "teacher": "Prof. Bagnara",
11      "course": "Informatica",
```

```

12     "price": 35.00,
13     "sellerId": 10,
14     "sellerUsername": "username",
15     "available": true
16   }
17 ]
18 }
```

Nota: *Immagine:* Il campo `imagePath` contiene l'immagine codificata in formato base64 data URI, pronta per essere utilizzata direttamente come attributo `src` di un tag ``. Se il libro non ha un'immagine associata, il campo sarà una stringa vuota.

Response (Error - Errore server):

```

1 {
2   "status": "error",
3   "message": "Server error"
4 }
```

5.1.4 GET /my-books

Recupera la lista di tutti i libri messi in vendita dall'utente correntemente loggato.

Auth: Obbligatoria (JWT)

Response (Success):

```

1 {
2   "status": "success",
3   "data": [
4     {
5       "id": 5,
6       "name": "Sistemi Operativi",
7       "author": "Silberschatz",
8       "isbn": "978-1118063330",
9       "imagePath": "data:image/jpeg;base64,...",
10      "teacher": "Prof. Veltri",
11      "course": "Informatica",
12      "price": 40.00,
13      "sellerId": 10,
14      "sellerUsername": "username",
15      "available": true
16    }
17  ]
18 }
```

5.1.5 POST /books

Inserimento di un nuovo libro nel catalogo.

Auth: JWT Required

Request Body:

```

1 {
2   "name": "Sistemi Operativi",
3   "author": "Silberschatz",
4   "isbn": "978-1122334455",
```

```

5   "image": "data:image/jpeg;base64,/9j/4AAQ...",
6   "teacher": "Prof. Bagnara",
7   "course": "Informatica",
8   "price": 28.00
9 }
```

Nota: Il campo `sellerId` viene estratto automaticamente dal JWT (claim `sub`) e non deve essere specificato nel body.

Nota: Upload Immagine: Il campo `image` è opzionale e accetta dati in formato `base64` (con o senza prefisso `data URI`). Formati supportati: `JPEG`, `PNG`, `WebP`. Dimensione massima: `5MB`.

Response (Success):

```

1 {
2   "status": "success",
3   "id": 105,
4   "message": "Libro messo in vendita con successo"
5 }
```

Response (Error - Input non valido):

```

1 {
2   "status": "error",
3   "message": "Invalid input"
4 }
```

Response (Error - Formato immagine non valido):

```

1 {
2   "status": "error",
3   "message": "Invalid file format"
4 }
```

Response (Error - File troppo grande):

```

1 {
2   "status": "error",
3   "message": "File too large"
4 }
```

5.1.6 PUT /books

Aggiornamento di un libro esistente. L'`id` è obbligatorio.

Auth: JWT Required

Nota: Ownership: L'utente può modificare solo i propri libri. Il sistema verifica che `book.sellerId` corrisponda all'ID dell'utente autenticato.

Request Body:

```

1 {
2   "id": 172,
3   "price": 25.00,
4   "available": false
5 }
```

Response (Success):

```

1 {
2   "status": "success",
3   "message": "Book updated"
4 }
```

Response (Error - Input non valido):

```

1 {
2   "status": "error",
3   "message": "Invalid input"
4 }
```

Response (Error - Libro non trovato):

```

1 {
2   "status": "error",
3   "message": "Book not found"
4 }
```

Response (Error - Non proprietario):

```

1 {
2   "status": "error",
3   "message": "Forbidden"
4 }
```

5.1.7 DELETE /books

Rimozione di un libro.

Auth: JWT Required

Nota: Ownership: L'utente può eliminare solo i propri libri. Il sistema verifica che `book.sellerId` corrisponda all'ID dell'utente autenticato.

Request Body: {"id": 78}

Response (Success):

```

1 {
2   "status": "success",
3   "message": "Book deleted"
4 }
```

Response (Error - Input non valido):

```

1 {
2   "status": "error",
3   "message": "Invalid input"
4 }
```

Response (Error - Libro non trovato):

```

1 {
2   "status": "error",
3   "message": "Book not found"
4 }
```

Response (Error - Non proprietario):

```

1 {
2   "status": "error",
3   "message": "Forbidden"
4 }
```

5.1.8 POST /purchase

Effettua l'acquisto di un libro.

Auth: JWT Required

Request Body:

```

1 {
2   "bookId": 78
3 }
```

Response (Success):

```

1 {
2   "status": "success",
3   "message": "Purchase completed successfully",
4   "orderId": 501,
5   "sellerEmail": "venditore@email.it"
6 }
```

Response (Error - Libro non trovato):

```

1 {
2   "status": "error",
3   "message": "Book not found"
4 }
```

Response (Error - Libro già venduto):

```

1 {
2   "status": "error",
3   "message": "Book already sold"
4 }
```

Response (Error - Auto-acquisto):

```

1 {
2   "status": "error",
3   "message": "Cannot purchase your own book"
4 }
```

5.1.9 GET /purchases

Recupera lo storico degli acquisti dell'utente autenticato.

Auth: JWT Required

Response (Success):

```

1 {
2   "status": "success",
3   "data": [
4     {
5       "orderId": 501,
6       "book": {
7         "id": 78,
8         "name": "Ingegneria del Software",
9         "author": "Ian Sommerville",
10        "price": 35.00
11      },
12      "sellerUsername": "mario_rossi",
13      "purchaseDate": "2026-01-25T14:30:00Z"
14    }
15  ]
16 }

```

Response (Error - Errore server):

```

1 {
2   "status": "error",
3   "message": "Server error"
4 }

```

5.1.10 GET /sales

Recupera lo storico delle vendite dell'utente autenticato.

Auth: JWT Required

Response (Success):

```

1 {
2   "status": "success",
3   "data": [
4     {
5       "orderId": 501,
6       "book": {
7         "id": 78,
8         "name": "Ingegneria del Software",
9         "author": "Ian Sommerville",
10        "price": 35.00
11      },
12      "buyerUsername": "luigi_verdi",
13      "saleDate": "2026-01-25T14:30:00Z"
14    }
15  ]
16 }

```

Response (Error - Errore server):

```

1 {
2   "status": "error",
3   "message": "Server error"
4 }

```

5.1.11 GET /conversations

Recupera la lista delle conversazioni dell'utente autenticato.

Auth: JWT Required

Response (Success):

```

1 {
2   "status": "success",
3   "data": [
4     {
5       "userId": 5,
6       "username": "luigi_verdi",
7       "lastMessage": "Ok, ci vediamo domani",
8       "lastMessageDate": "2026-01-25T14:30:00Z"
9     }
10   ]
11 }
```

Response (Error - Errore server):

```

1 {
2   "status": "error",
3   "message": "Server error"
4 }
```

5.1.12 GET /messages?userId={id}

Recupera i messaggi scambiati con un utente specifico.

Auth: JWT Required

Query Params: userId (required)

Response (Success):

```

1 {
2   "status": "success",
3   "data": [
4     {
5       "id": 1,
6       "senderId": 5,
7       "receiverId": 10,
8       "content": "Ciao, il libro e' ancora disponibile?",
9       "createdAt": "2026-01-25T14:00:00Z"
10    }
11   ]
12 }
```

Response (Error - Utente non trovato):

```

1 {
2   "status": "error",
3   "message": "User not found"
4 }
```

5.1.13 POST /messages

Invia un nuovo messaggio a un utente.

Auth: JWT Required

Request Body:

```

1 {
2   "receiverId": 5,
3   "content": "Ciao, sono interessato al libro"
4 }
```

Response (Success):

```

1 {
2   "status": "success",
3   "message": "Message sent",
4   "messageId": 42
5 }
```

Response (Error - Destinatario non trovato):

```

1 {
2   "status": "error",
3   "message": "User not found"
4 }
```

Response (Error - Messaggio a se stesso):

```

1 {
2   "status": "error",
3   "message": "Cannot message yourself"
4 }
```

Response (Error - Contenuto vuoto):

```

1 {
2   "status": "error",
3   "message": "Message content required"
4 }
```

Nota: Ricerca Avanzata (RF05): La funzionalità di ricerca per ISBN, corso e docente è implementata interamente lato Frontend mediante filtri JavaScript sui dati già recuperati tramite `GET /books`. Non sono previsti endpoint dedicati.

5.2 Interfaccia Database

5.2.1 Entity Classes

Oggetti PHP rappresentanti i dati scambiati tra i vari moduli. Il campo `id` è nullable per poter creare nuovi oggetti.

```

1 class User {
2   public ?int $id;
3   public string $username;
4   public string $email;
```

```

5     public string $password;
6 }
7
8 class Book {
9     public ?int $id;
10    public string $name;
11    public string $author;
12    public string $isbn;
13    public string $imagePath;
14    public string $teacher;
15    public string $course;
16    public float $price;
17    public int $sellerId;
18    public bool $available;
19 }
20
21 class Transaction {
22     public ?int $id;
23     public int $bookId;
24     public int $buyerId;
25     public int $sellerId;
26     public DateTime $createdAt;
27 }
28
29 class Message {
30     public ?int $id;
31     public int $senderId;
32     public int $receiverId;
33     public string $content;
34     public DateTime $createdAt;
35 }

```

5.2.2 Metodi Gestione Utenti

- `registerUser(User $user): bool` – Esegue l'insert del nuovo utente.
- `verifyCredentials(string $email, string $password): ?int` – Ritorna l'ID utente se le credenziali sono corrette (con verifica `password_verify`), altrimenti `null`.
- `getUserById(int $id): ?User` – Ritorna l'utente con l'ID specificato, oppure `null` se non trovato.

Nota: Il metodo `getUserById` è stato aggiunto per supportare il recupero dell'email del venditore in `POST /purchase`.

5.2.3 Metodi Gestione Libri

- `getAllBooks(): array` – Ritorna un array di oggetti `Book` disponibili.
- `insertBook(Book $book): int` – Inserisce un libro e ritorna l'ID generato.
- `updateBook(Book $book): bool` – Aggiorna i campi modificati.
- `deleteBook(int $id): bool` – Rimuove logicamente o fisicamente il record.

5.2.4 Metodi Gestione Transazioni

- `placeOrder(int $bookId, int $buyerId): int` – Crea un record nella tabella ordini, aggiorna `available = false` sul libro e ritorna l'ID ordine.
- `getPurchasesByBuyer(int $buyerId): array` – Ritorna lo storico acquisti dell'utente.
- `getSalesBySeller(int $sellerId): array` – Ritorna lo storico vendite dell'utente.

5.2.5 Metodi Gestione Messaggi

- `getConversations(int $userId): array` – Ritorna la lista delle conversazioni con ultimo messaggio.
- `getMessages(int $userId1, int $userId2): array` – Ritorna i messaggi tra due utenti ordinati cronologicamente.
- `sendMessage(Message $message): int` – Inserisce un messaggio e ritorna l'ID generato.

Capitolo 6

Sprint Log

6.1 Sprint 1

Periodo: 11/01/26 - 25/01/26

User Stories: US01, US02, US03, US04

6.1.1 Task

ID	Descrizione	Assegnatario	Stato	US
T01	Progettazione Schema SQL e Classi Entity	Andrea	Done	Global
T02	Sviluppo classe interfaccia Database	Andrea	Done	US03, US04
T03	Logica di Backend e Sicurezza: API Auth, Hashing, JWT	Samuele	Done	US01, US02
T04	Sviluppo API Catalogo e Upload immagini	Samuele	Done	US03, US04
T05	Struttura pagina autenticazione e registrazione	Kenneth	Done	US01, US02
T06	Struttura bacheca annunci e inserimento annuncio	Kenneth	Done	US03, US04

Tabella 6.1: Task dello Sprint 1

6.2 Sprint 2

Periodo: 25/01/26 - 28/01/26

User Stories: US05, US06, US07

6.2.1 Task

ID	Descrizione	Assegnatario	Stato	US
T07	Progettazione schema tabelle transactions e messages	Andrea	Done	US05, US07
T08	Sviluppo metodi interfaccia DB per acquisti e messaggi	Andrea	Done	US05, US07
T09	Implementazione API Acquisto (POST /purchase)	Samuele	Done	US05
T10	Implementazione API Messaggistica (CRUD messages)	Samuele	Done	US07
T11	Struttura pagina dettaglio libro con pulsante Acquista	Kenneth	Done	US05
T12	Implementazione barra di ricerca avanzata e filtri	Kenneth	Done	US06
T13	Struttura pagina messaggi e chat tra utenti	Kenneth	Done	US07

Tabella 6.2: Task dello Sprint 2

Capitolo 7

Use Cases

7.1 UC01 - Registrazione Utente

7.1.1 Panoramica

Descrizione: Consente a un visitatore di creare un nuovo account nel sistema, permettendogli successivamente di accedere alle funzionalità riservate.

Attori	Visitatore
Pre-condizioni	Il visitatore non è autenticato e si trova nella pagina di registrazione
Post-condizioni	L'account è creato nel DB e il visitatore può fare login

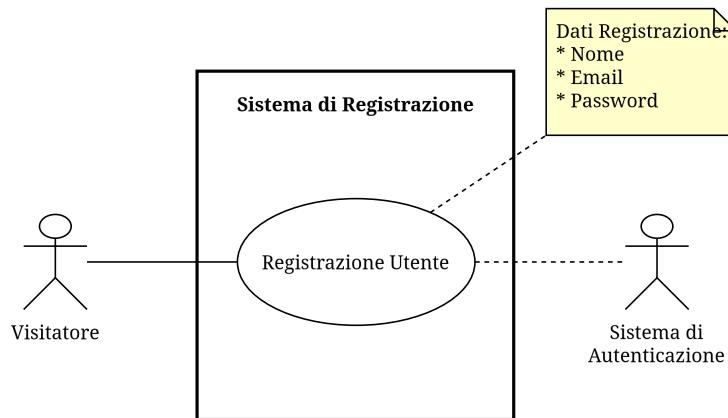


Figura 7.1: Use Case Diagram - UC01 Registrazione

7.1.2 Flussi di Eventi

Flusso Principale

1. Il **Visitatore** accede alla pagina di registrazione.
2. Il sistema mostra il modulo (Nome, Email, Password).
3. Il **Visitatore** compila i campi e conferma.
4. Il sistema verifica che l'email non sia già registrata.
5. Il sistema valida il formato dei dati (es. password strong).
6. Il sistema crea il nuovo account nel database.
7. Il sistema reindirizza al login con messaggio di successo.

Flussi Alternativi**A1: Email già registrata**

1. Il sistema rileva che l'email esiste.
2. Il sistema mostra errore: "Email già in uso".
3. Il flusso termina (l'utente deve riprovare).

A2: Dati non validi

1. Il sistema rileva formato errato.
2. Il sistema evidenzia i campi in rosso.
3. L'utente corregge e si torna al passo 3.

7.1.3 Activity Diagram

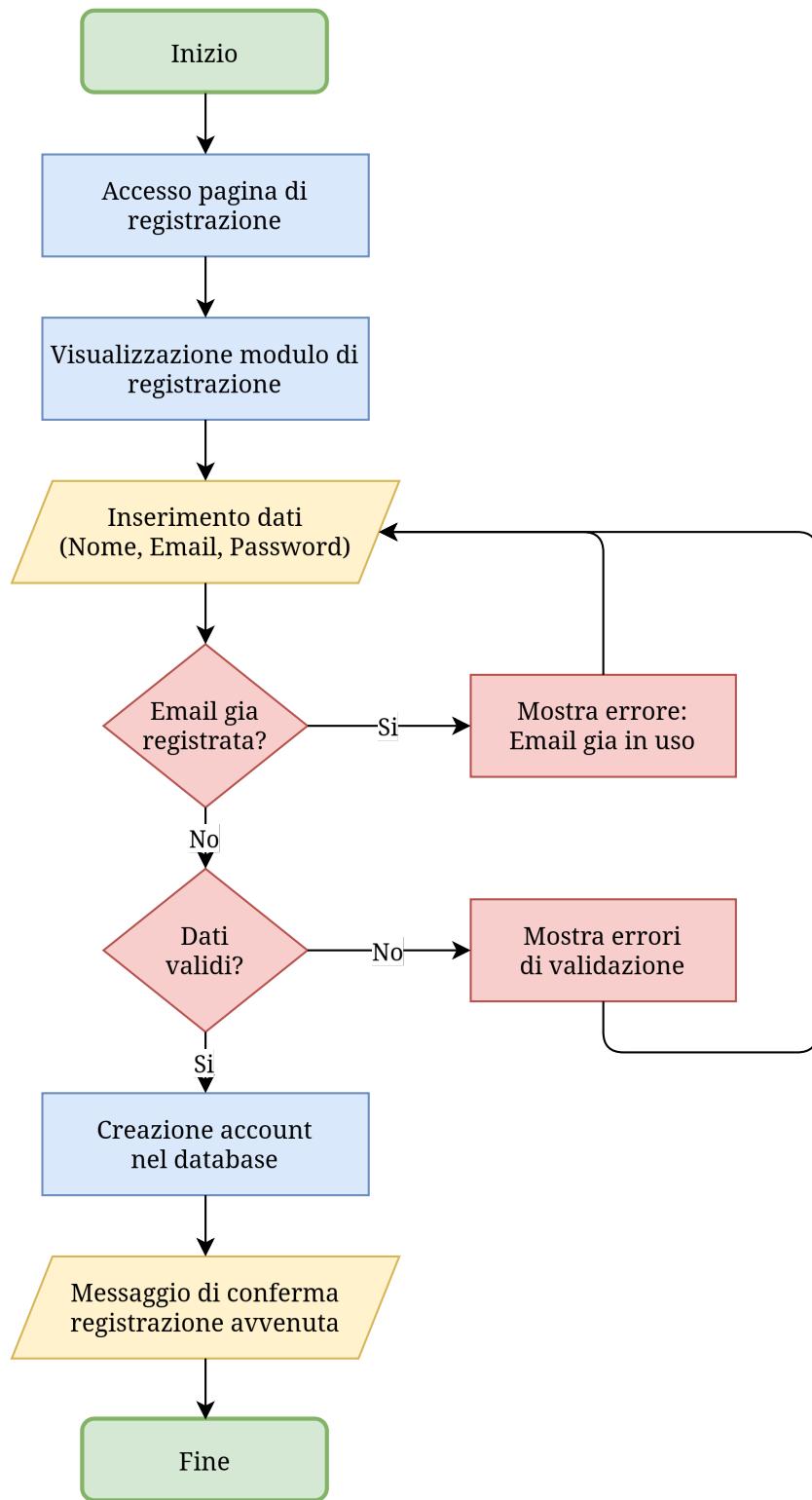


Figura 7.2: Activity Diagram - UC01 Registrazione

7.1.4 Criteri di Accettazione e Testing

- La password deve avere almeno 8 caratteri.
- Se la registrazione ha successo, l'utente non viene loggato automaticamente ma va al login.

- La mail deve contenere “@” e un dominio valido.

ID Test	Azione	Risultato Atteso	Valida
T01	Invia form vuoto	Bordi rossi sui campi	✓
T02	Invia mail esistente	Messaggio errore specifico	✓
T03	Invia dati corretti	Redirect a /login	✓

Tabella 7.1: Piano di Test - UC01

7.1.5 Specifiche Tecniche

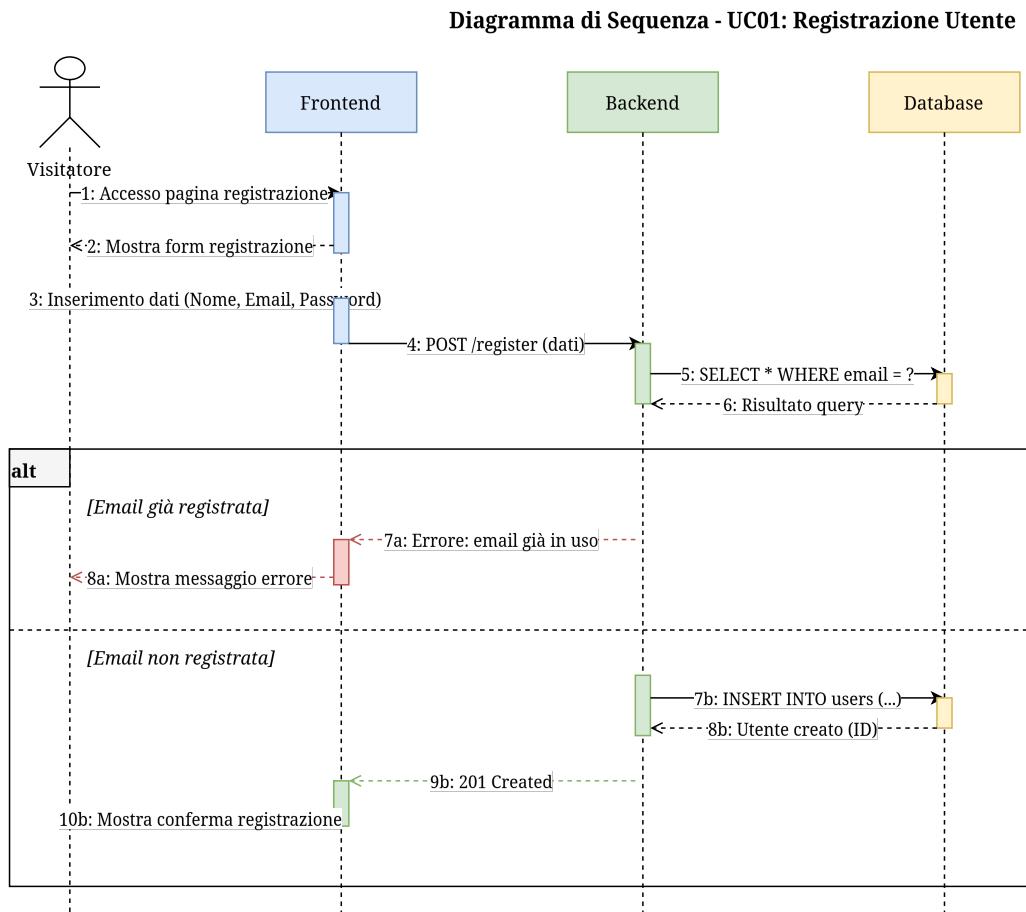


Figura 7.3: Sequence Diagram - UC01

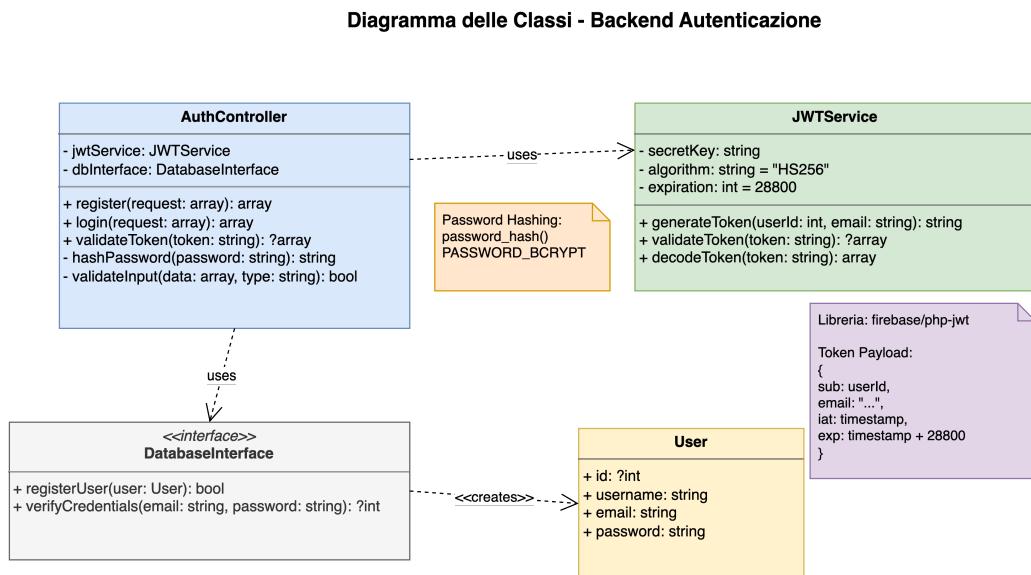
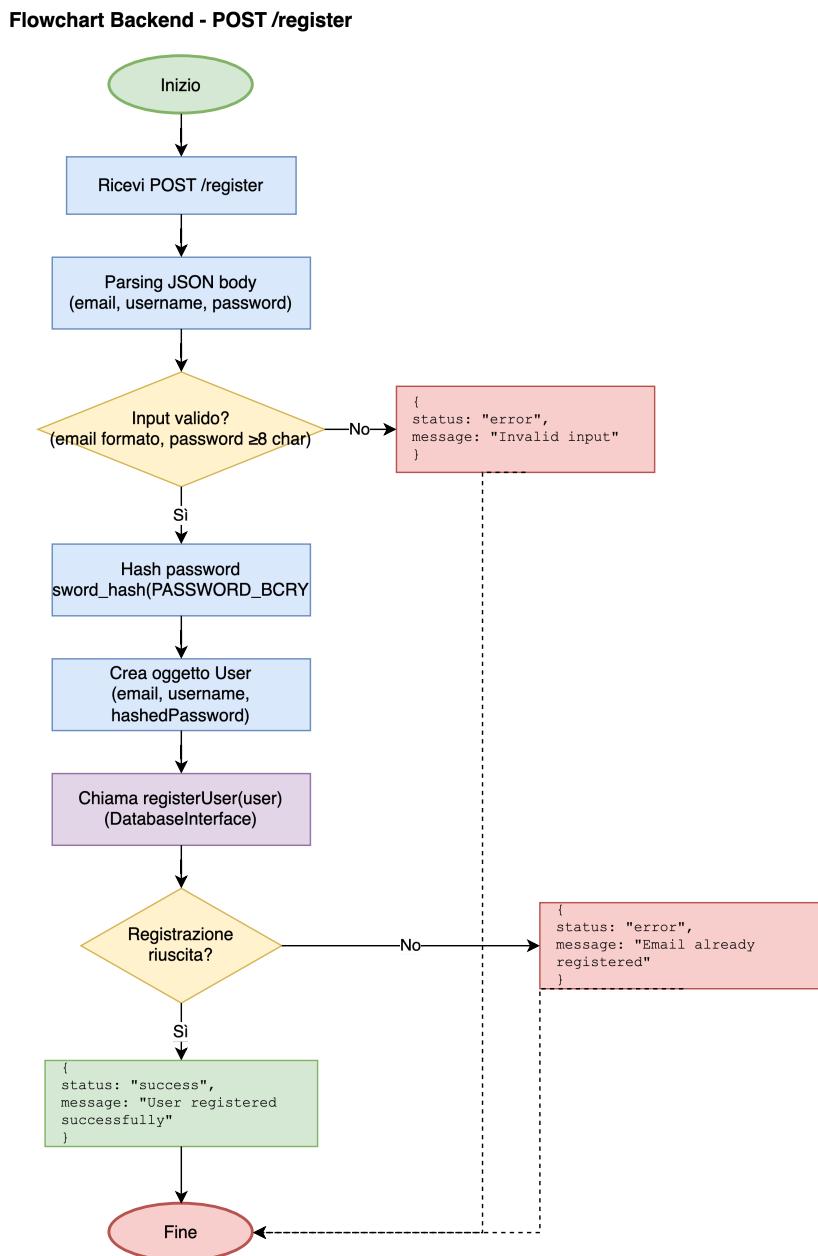


Figura 7.4: Class Diagram Backend - Autenticazione

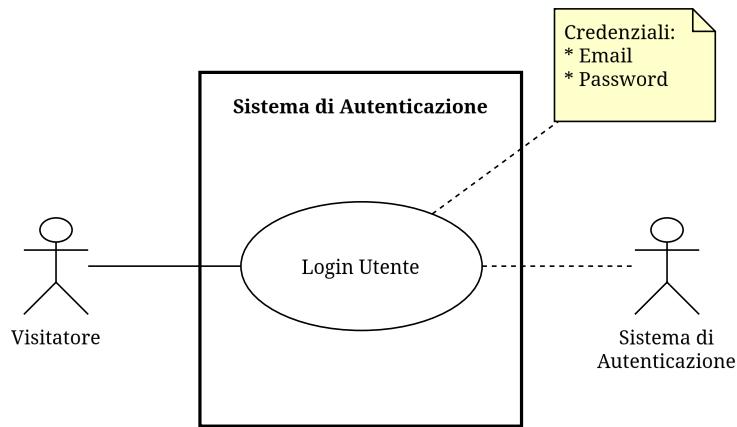
**Figura 7.5:** Backend Flowchart - UC01

7.2 UC02 - Login Utente

7.2.1 Panoramica

Descrizione: Consente a un visitatore registrato di autenticarsi nel sistema per accedere alle funzionalità riservate agli utenti (come la pubblicazione di annunci o l'acquisto).

Attori	Visitatore (Registrato)
Pre-condizioni	Il visitatore possiede un account e non è già autenticato
Post-condizioni	Il sistema crea una sessione/token e l'utente viene autenticato

**Figura 7.6:** Use Case Diagram - UC02 Login

7.2.2 Flussi di Eventi

Flusso Principale

1. Il **Visitatore** accede alla pagina di login.
2. Il sistema mostra il modulo (Email e Password).
3. Il **Visitatore** inserisce le proprie credenziali e conferma.
4. Il sistema valida il formato dei dati (lato Frontend).
5. Il sistema verifica la corrispondenza delle credenziali nel database (lato Backend).
6. Il sistema genera un token di sessione.
7. Il sistema reindirizza l'utente alla Dashboard/Home con messaggio di successo.

Flussi Alternativi

A1: Credenziali errate o Account inesistente

1. Il sistema rileva che l'email non esiste o la password è errata.
2. Il sistema mostra un messaggio di errore generico: "Email o password non corretti".
3. Il flusso riprende dal punto 3.

Nota: Si usa un messaggio generico per evitare che malintenzionati scoprano quali email sono registrate.

A2: Formato dati non valido

1. Il sistema rileva che la mail non è nel formato corretto o i campi sono vuoti.
2. Il sistema blocca l'invio e segnala i campi da correggere.

7.2.3 Activity Diagram

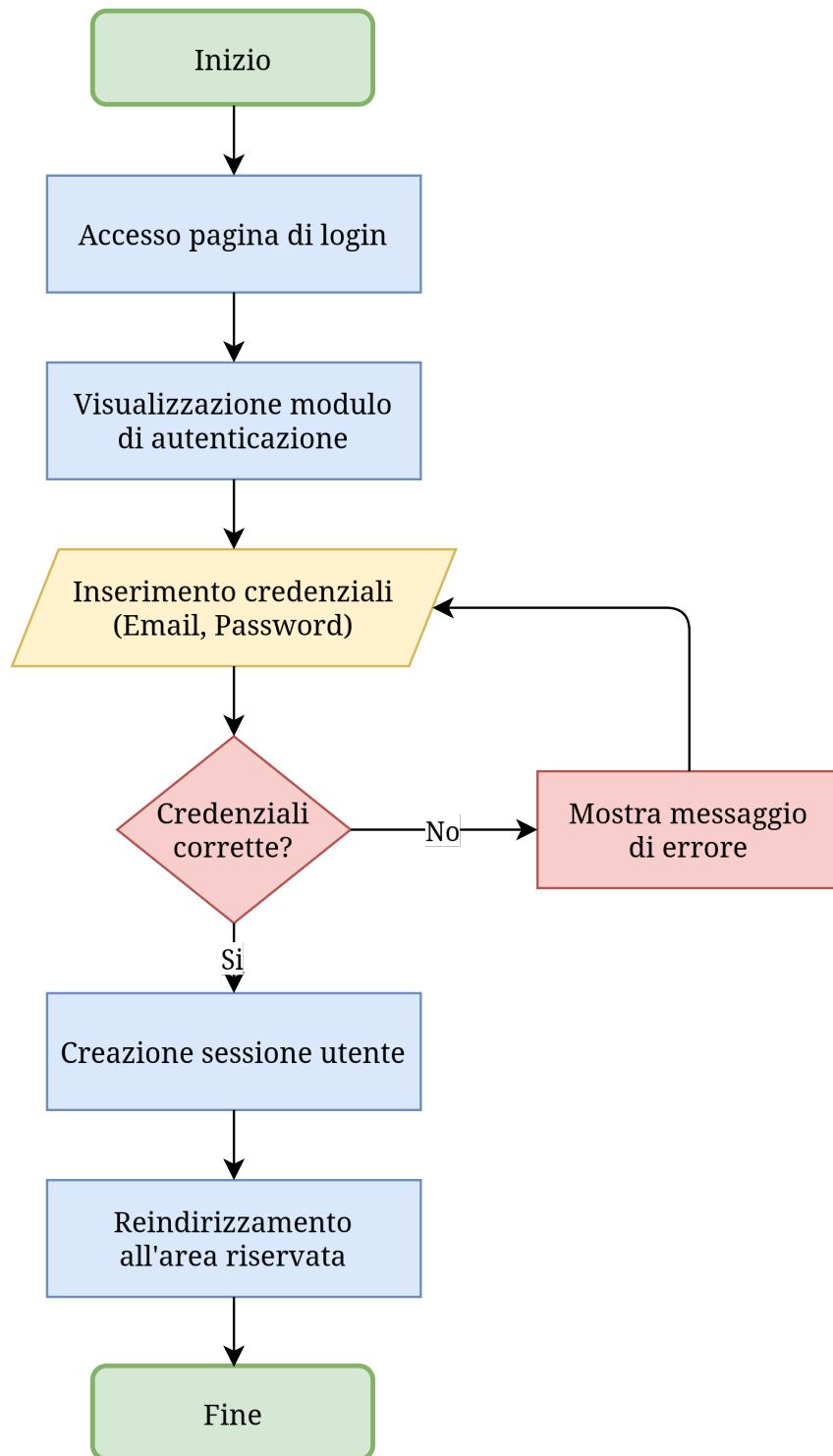


Figura 7.7: Activity Diagram - UC02 Login

7.2.4 Criteri di Accettazione

- **Sicurezza:** Le password non devono mai apparire in chiaro nel modulo.
- **Sessione:** Al login effettuato, il sistema deve memorizzare lo stato di “Loggato”.

- **Feedback:** In caso di errore, i campi non devono essere resettati per permettere la correzione veloce.

7.2.5 Piano di Test Manuale

ID	Azione	Risultato Atteso	Valida
T01	Inserire email non registrata	Messaggio di errore generico	✓
T02	Inserire email corretta ma password errata	Messaggio di errore generico	✓
T03	Lasciare il campo password vuoto	Tasto “Login” disabilitato o errore	✓
T04	Inserire credenziali corrette	Reindirizzamento alla Home	✓

Tabella 7.2: Piano di Test - UC02

7.2.6 Design Tecnico

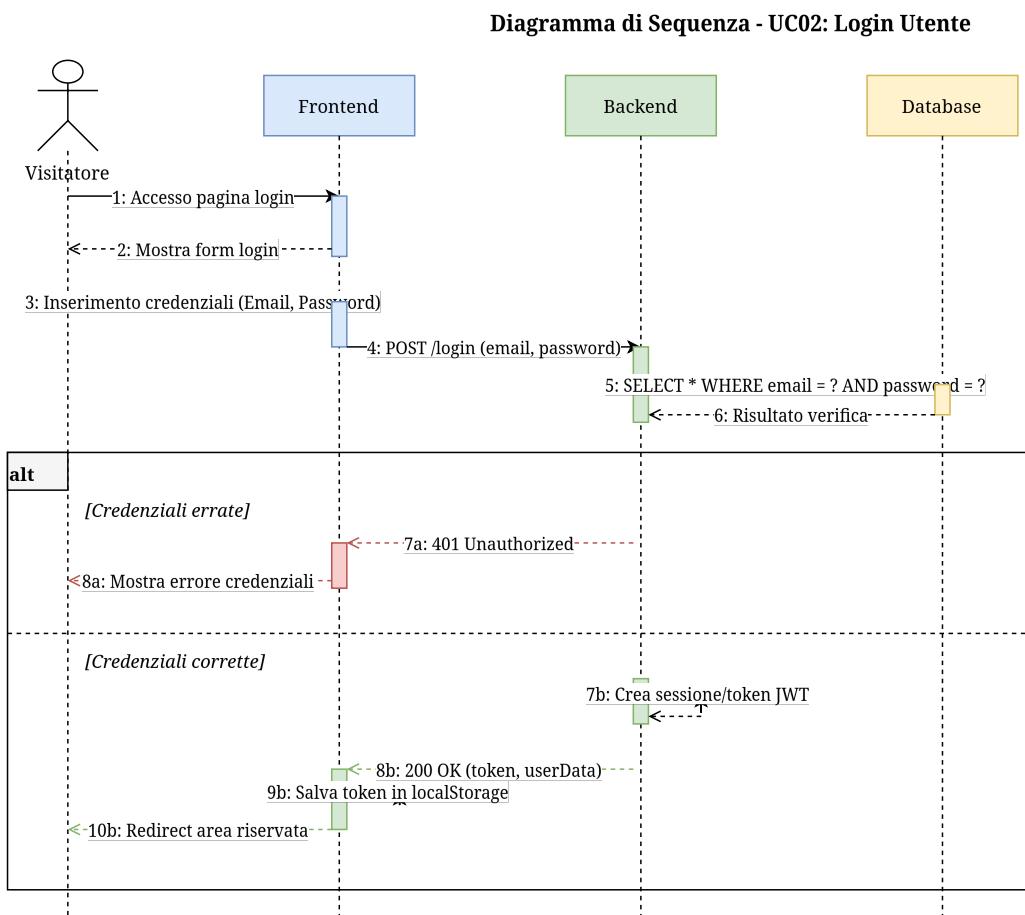
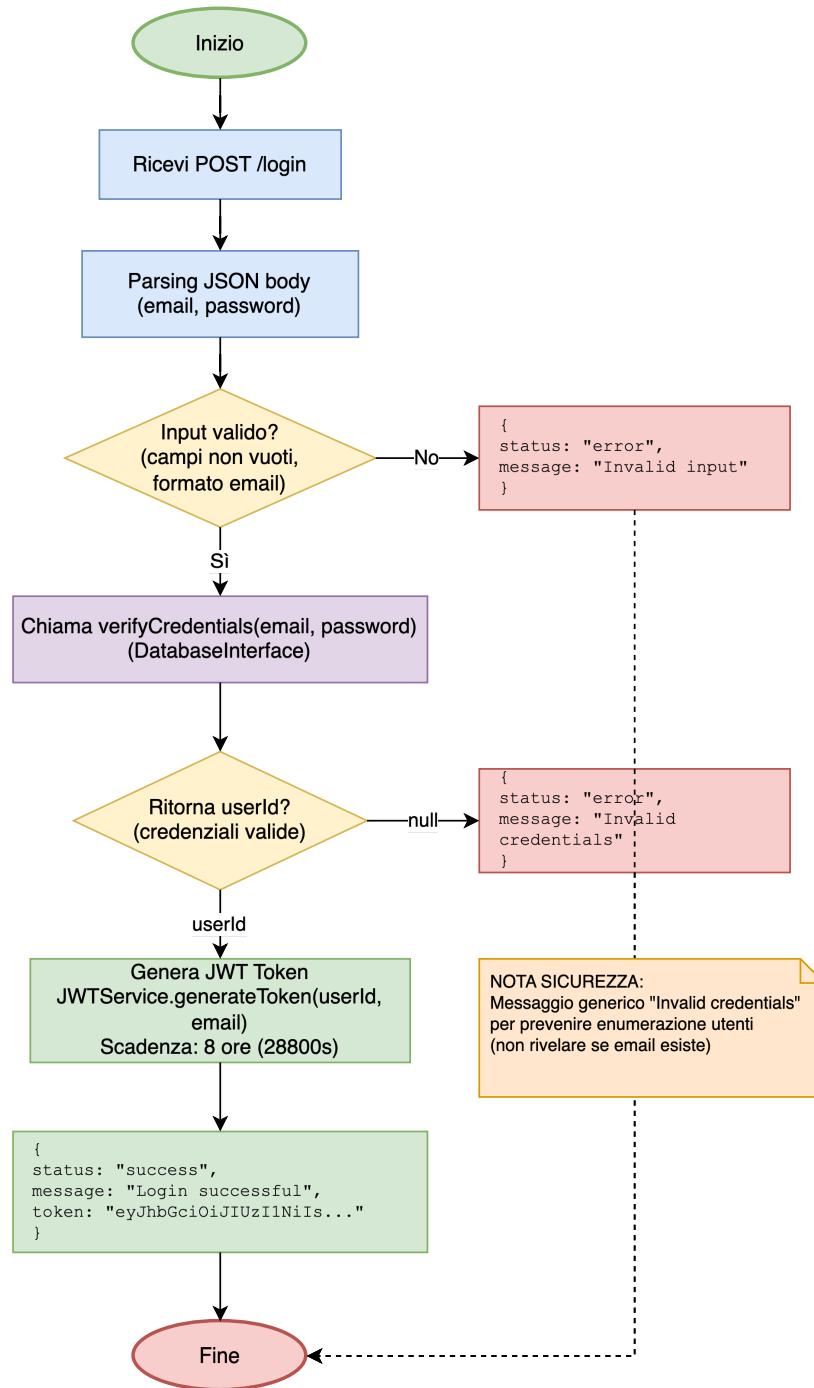


Figura 7.8: Sequence Diagram - UC02

Flowchart Backend - POST /login

Figura 7.9: Backend Flowchart - UC02

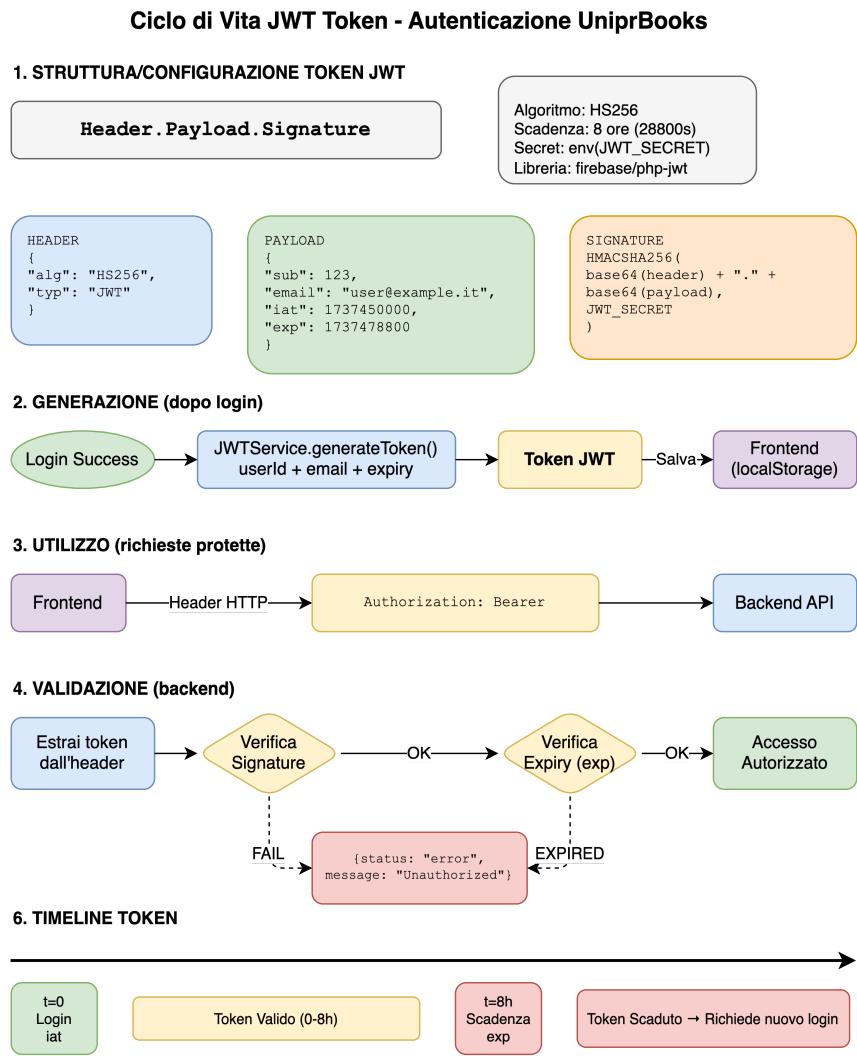


Figura 7.10: Ciclo di Vita JWT Token

7.3 UC03 - Visualizzazione Lista Libri

7.3.1 Panoramica

Descrizione: Consente a un Visitatore o a un Utente Autenticato di consultare la bacheca degli annunci disponibili. Se l'utente è loggato, il sistema nasconde automaticamente i suoi annunci.

Attori	Visitatore, Utente Autenticato, Database Annunci
Pre-condizioni	L'utente accede alla pagina "Bacheca" o "Home"
Post-condizioni	Il sistema mostra i libri filtrati disponibili per l'acquisto

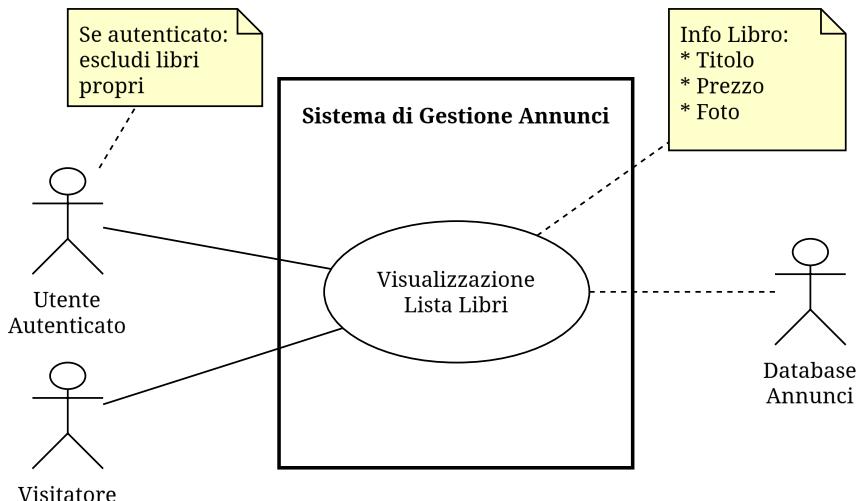


Figura 7.11: Use Case Diagram - UC03 Visualizzazione

7.3.2 Flussi di Eventi

Flusso Principale

1. L'**Utente/Visitatore** accede alla bacheca dei libri.
2. Il sistema (Presenter) richiede la lista degli annunci al Model (API).
3. Il sistema (Backend) recupera i libri dal database.
4. **Filtro Identità:** Se l'utente è autenticato, il sistema esclude dalla lista gli annunci creati dall'utente stesso.
5. Il sistema mostra la lista ordinata (es. dal più recente) con: Titolo, Prezzo, Foto e Autore.
6. L'**Utente/Visitatore** visualizza i risultati.

Flussi Alternativi

A1: Nessun libro disponibile

1. Il database non restituisce risultati.
2. Il sistema mostra un messaggio: "Al momento non ci sono libri disponibili. Torna più tardi!".

A2: Errore caricamento immagini

1. Se la foto di un annuncio non è reperibile, il sistema mostra un'immagine segnaposto (placeholder) di default.

7.3.3 Activity Diagram

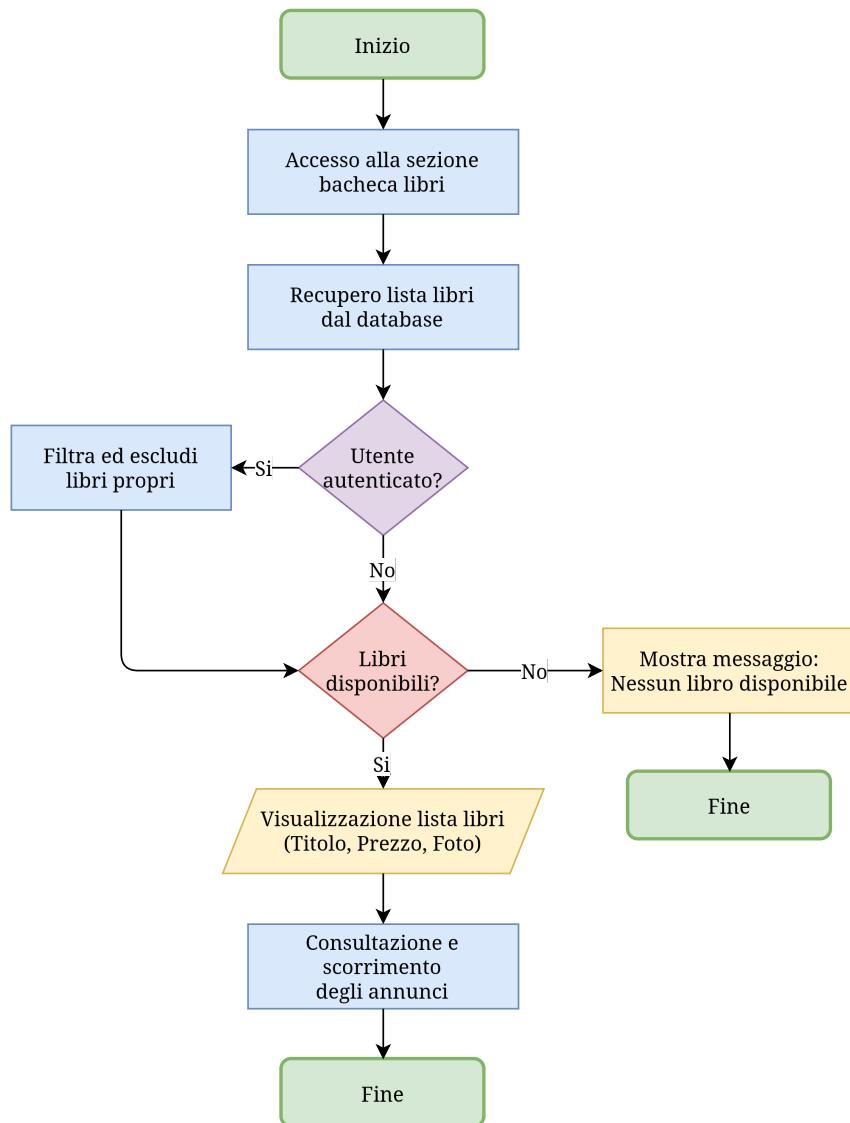


Figura 7.12: Activity Diagram - UC03 Visualizzazione

7.3.4 Criteri di Accettazione

- La lista deve essere caricata in modo asincrono senza ricaricare la pagina.
- L’utente non deve visualizzare i propri annunci nella lista.
- Ogni annuncio deve mostrare chiaramente il prezzo.

7.3.5 Piano di Test Manuale

ID	Azione	Risultato Atteso	Valida
T01	Accesso da Visitatore	Visualizzazione di tutti i libri nel DB	✓
T02	Accesso da Utente Loggato	I propri libri sono nascosti	✓
T03	Database vuoto	Messaggio “Nessun libro disponibile”	✓

Tabella 7.3: Piano di Test - UC03

7.3.6 Design Tecnico

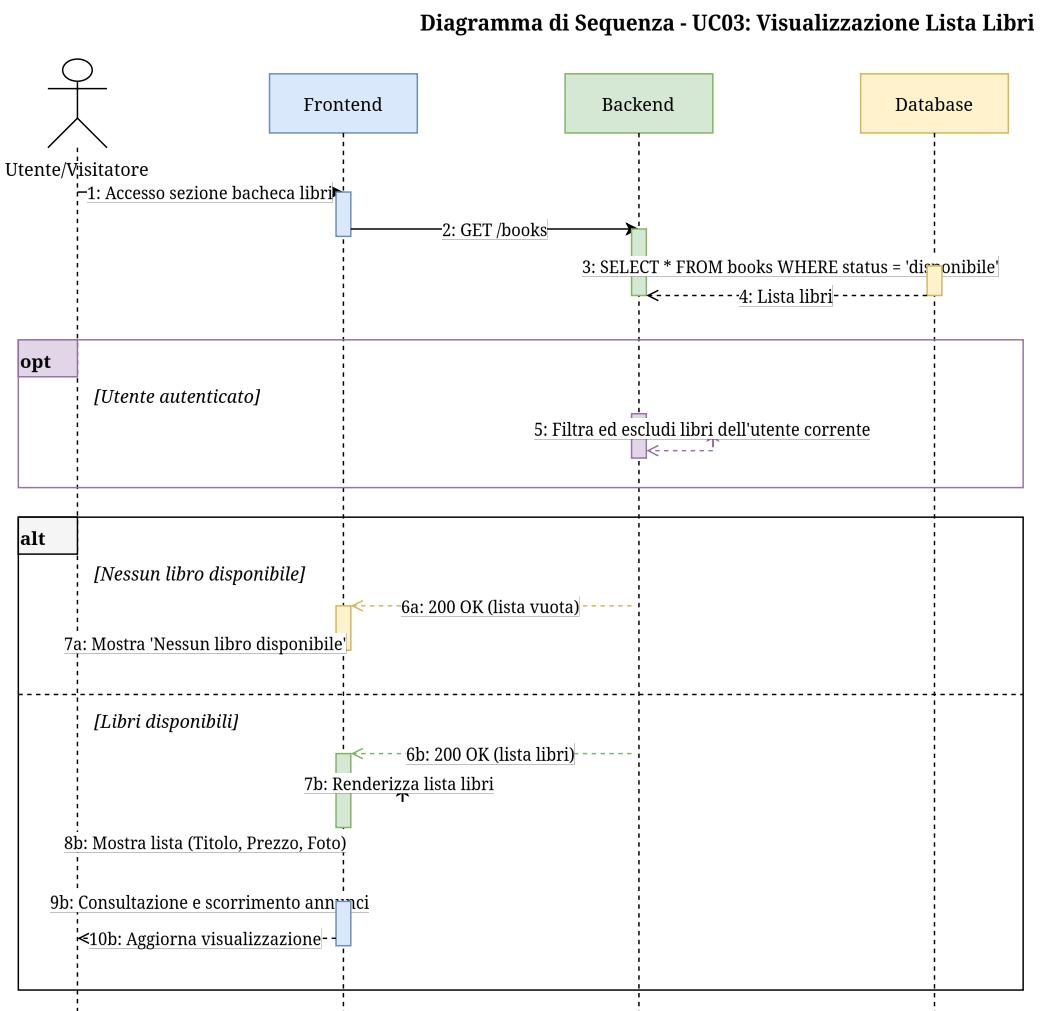
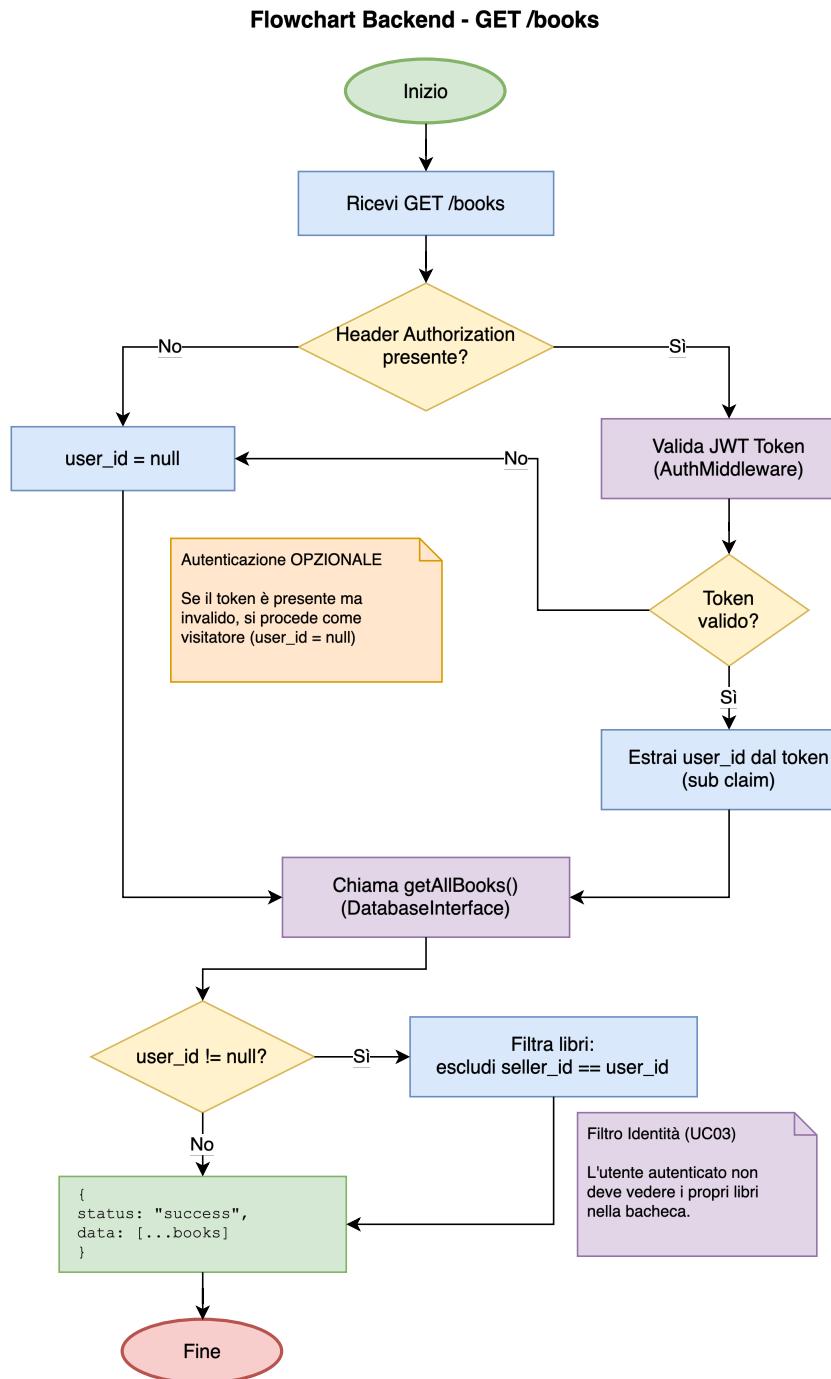
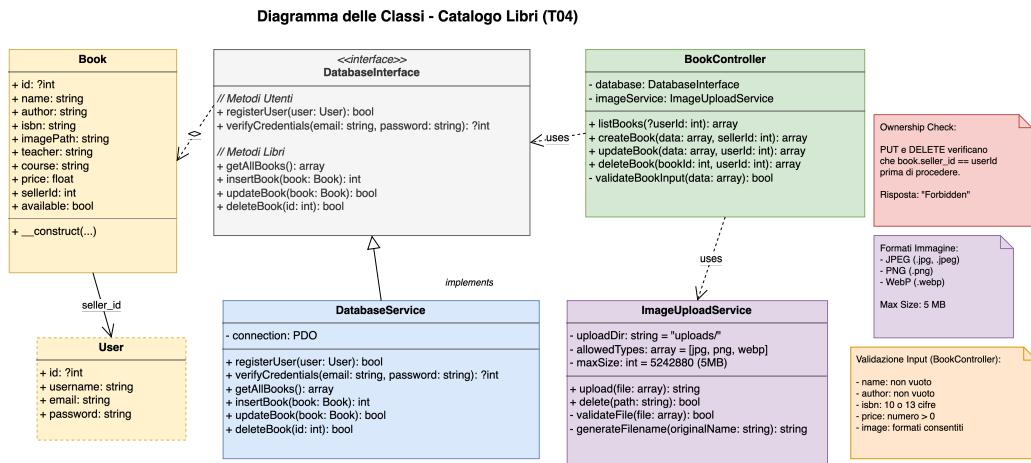


Figura 7.13: Sequence Diagram - UC03

**Figura 7.14:** Backend Flowchart - UC03

**Figura 7.15:** Class Diagram - Catalogo Libri

7.4 UC04 - Pubblicazione Annuncio

7.4.1 Panoramica

Descrizione: Consente a un Utente Autenticato di inserire un nuovo annuncio di vendita per un libro, specificando dettagli accademici e caricando una fotografia.

Attori	Utente Autenticato (Venditore)
Pre-condizioni	L'utente è loggato e si trova nel form di creazione annuncio
Post-condizioni	L'annuncio è salvato nel DB e l'immagine è memorizzata sul server

**Figura 7.16:** Use Case Diagram - UC04 Pubblicazione

7.4.2 Flussi di Eventi

Flusso Principale

- Il **Venditore** clicca su "Pubblica Annuncio".
- Il sistema mostra il modulo di inserimento: Titolo, Autore, ISBN, Prezzo, Corso, Docente e Caricamento Foto.
- Il **Venditore** compila i campi e seleziona un'immagine dal dispositivo.
- Il sistema (Frontend) valida la presenza dei campi obbligatori e il formato numerico del prezzo.

5. Il sistema (Backend) riceve i dati, salva l'immagine nel filesystem e crea il record nel database.
6. Il sistema reindirizza l'utente alla propria area personale o alla bacheca.

Flussi Alternativi

A1: Dati mancanti o Formato errato

1. Il sistema evidenzia i campi non validi (es. ISBN non numerico o prezzo negativo).
2. Il pulsante di invio viene disabilitato finché i dati non sono corretti.

A2: Errore Caricamento Immagine

1. Il file caricato non è un'immagine o supera la dimensione massima.
2. Il sistema mostra l'errore: “Formato file non supportato o file troppo grande”.

7.4.3 Activity Diagram

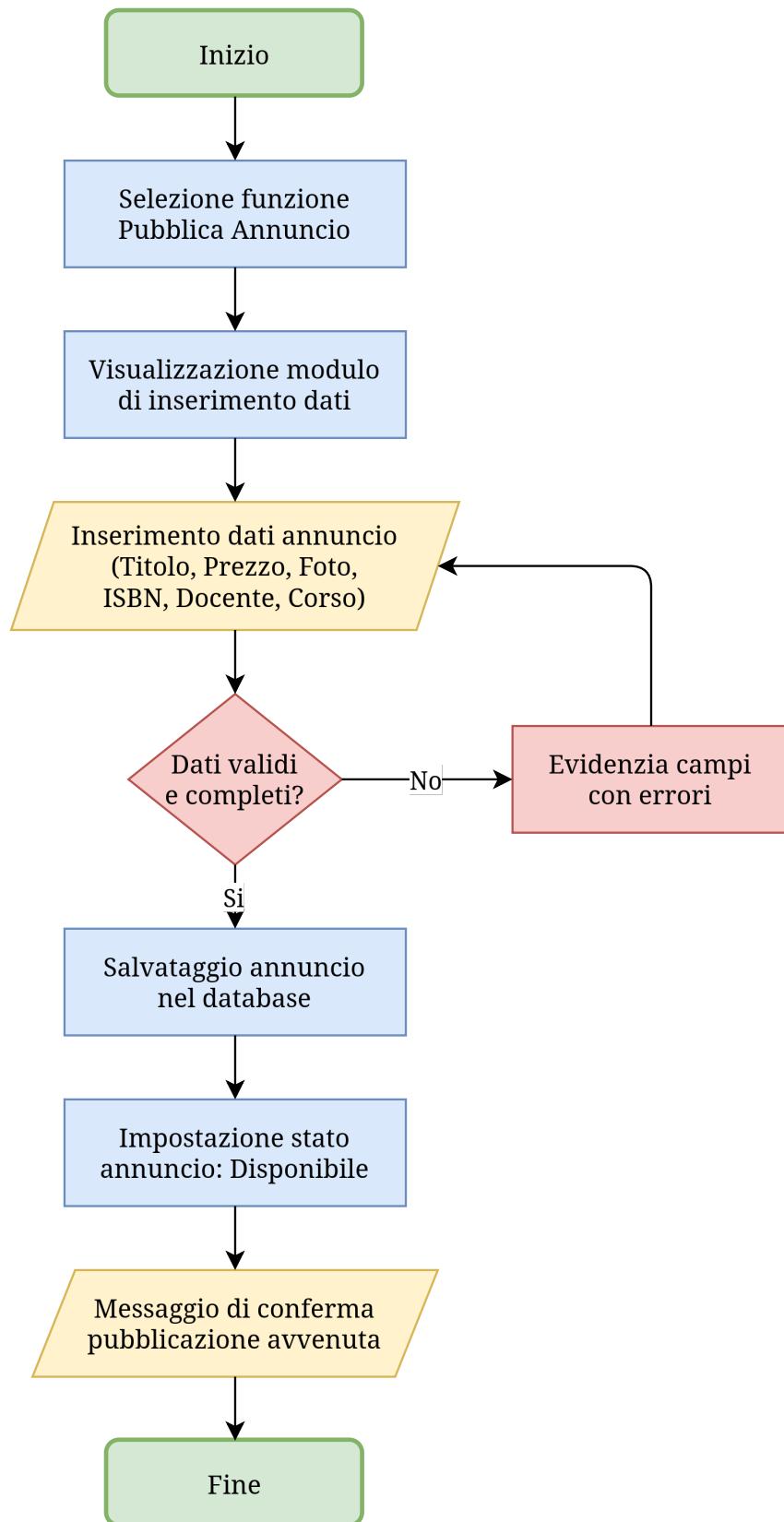


Figura 7.17: Activity Diagram - UC04 Pubblicazione

7.4.4 Criteri di Accettazione

- Il campo ISBN deve accettare solo numeri (10 o 13 cifre).
- Il prezzo deve essere obbligatoriamente un numero positivo.
- L'utente deve poter caricare una foto.
- L'annuncio deve essere collegato all'ID dell'utente che lo ha creato.

7.4.5 Piano di Test Manuale

ID	Azione	Risultato Atteso	Valida
T01	Tentativo di invio con campi vuoti	Errore “Campi obbligatori mancanti”	✓
T02	Inserimento prezzo negativo	Errore di validazione sul campo prezzo	✓
T03	Caricamento file non immagine	Blocco del caricamento e avviso	✓
T04	Invio corretto dei dati	Successo e comparsa in bacheca	✓

Tabella 7.4: Piano di Test - UC04

7.4.6 Design Tecnico

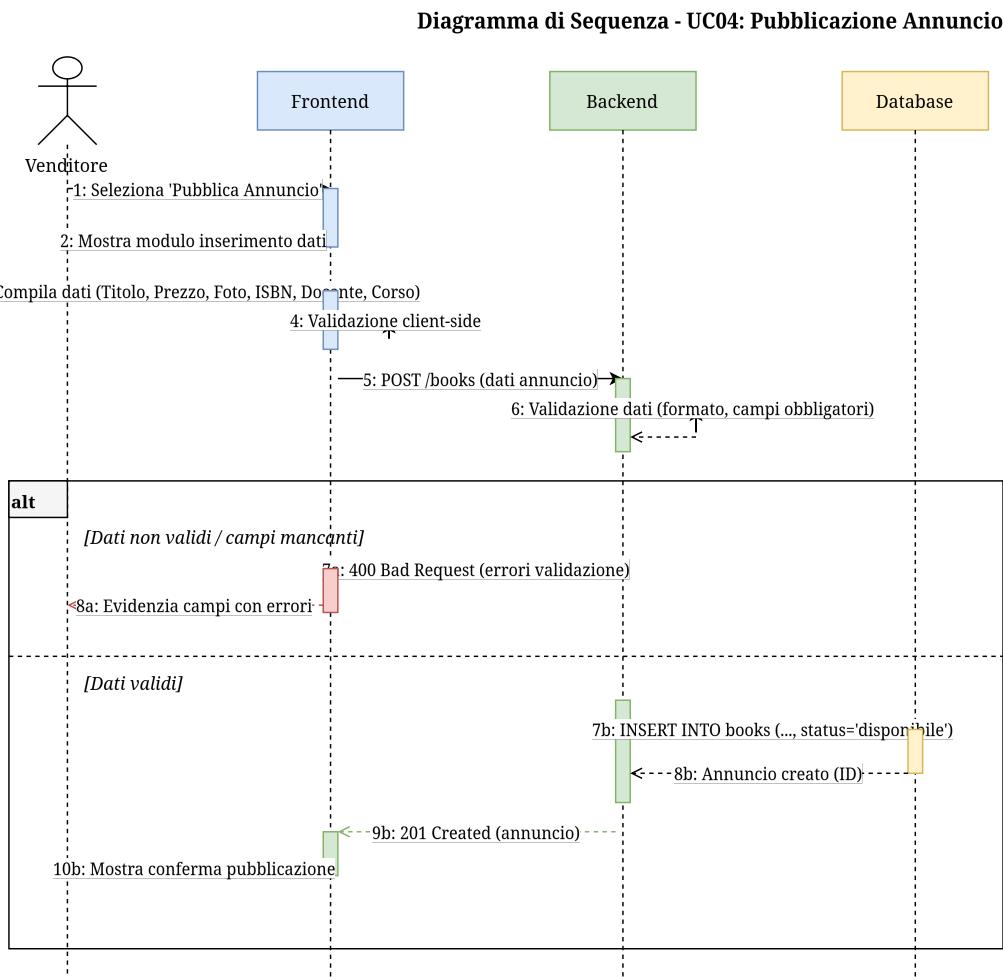
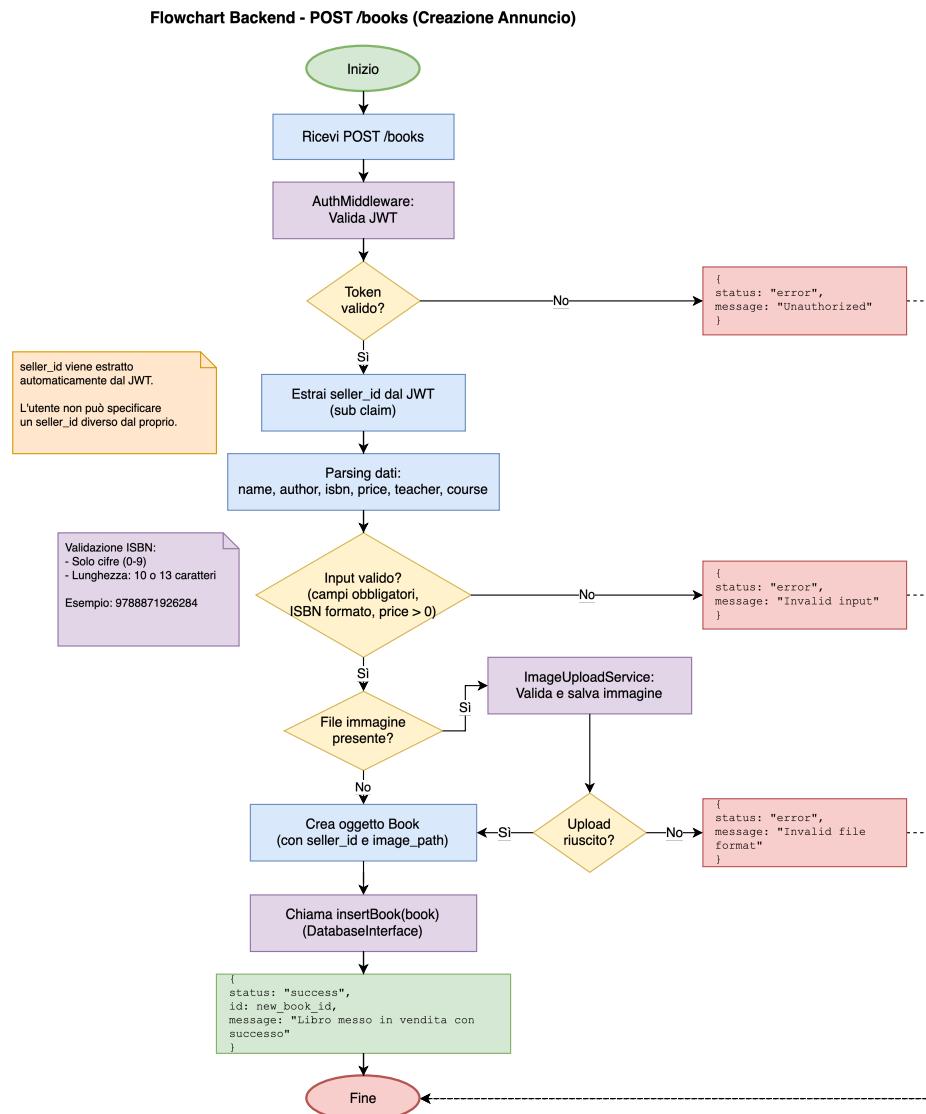


Figura 7.18: Sequence Diagram - UC04

**Figura 7.19:** Backend Flowchart - UC04

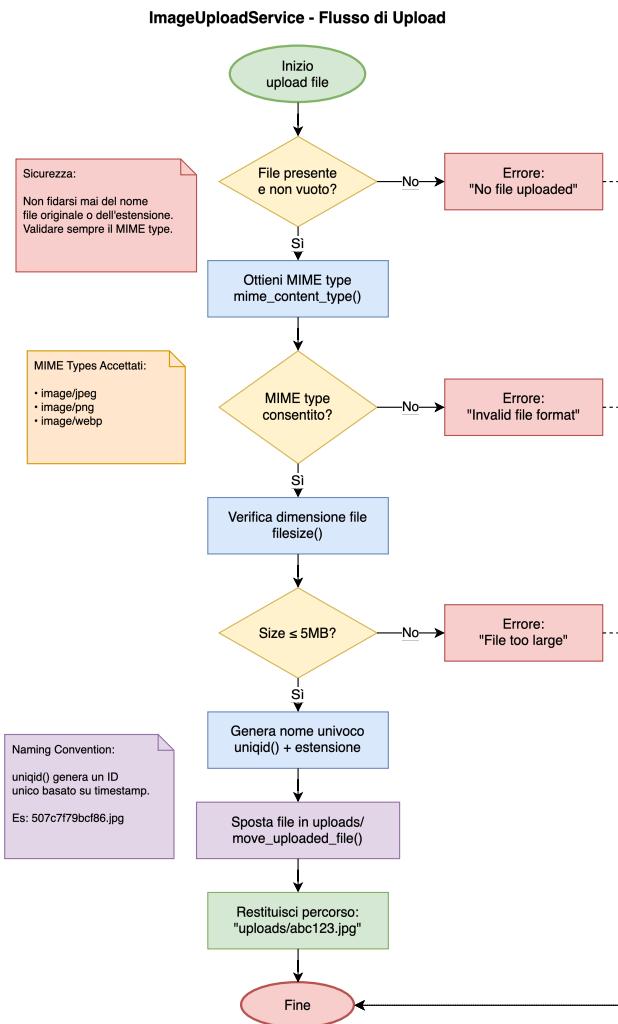


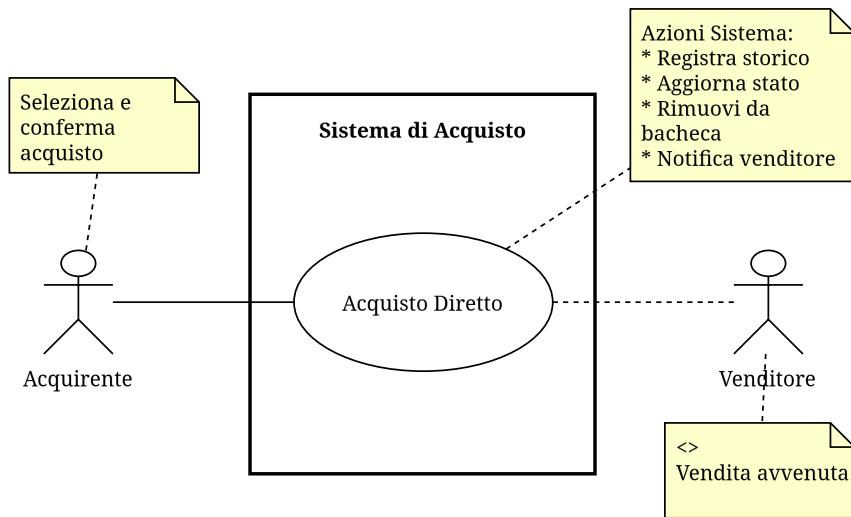
Figura 7.20: Flusso Upload Immagine

7.5 UC05 - Acquisto Diretto

7.5.1 Panoramica

Descrizione: Consente a un Utente Autenticato di acquisire un libro. L'operazione è puramente logica (non c'è transazione monetaria reale): il libro sparisce dalla bacheca e viene spostato negli storici dei due utenti coinvolti.

Attori	Utente Autenticato (Acquirente)
Pre-condizioni	L'acquirente è loggato e il libro ha stato “Disponibile”
Post-condizioni	Libro impostato come “Venduto”, rimosso dalla bacheca e aggiunto agli storici

**Figura 7.21:** Use Case Diagram - UC05 Acquisto

7.5.2 Flussi di Eventi

Flusso Principale

1. L'**Acquirente** visualizza i dettagli di un libro nella bachecca.
2. L'**Acquirente** clicca sul pulsante “Acquista”.
3. Il sistema chiede conferma dell’operazione.
4. L'**Acquirente** conferma.
5. Il sistema (Backend) verifica che il libro sia ancora nello stato “Disponibile”.
6. Il sistema crea un record nella tabella **transactions**.
7. Il sistema aggiorna lo stato del libro in **status = 'sold'**.
8. Il sistema mostra un messaggio di successo e fornisce i contatti del venditore.

Flussi Alternativi

A1: Libro già venduto

1. Il sistema rileva che il libro è appena stato acquistato da un altro utente.
2. Il sistema mostra l’errore: “Spiacenti, il libro è stato appena venduto”.
3. L’utente viene reindirizzato alla bachecca aggiornata.

A2: Tentativo di auto-acquisto

1. Il sistema rileva che l’acquirente è anche il venditore del libro.
2. Il pulsante “Acquista” è disabilitato o restituisce un errore di logica.

7.5.3 Activity Diagram

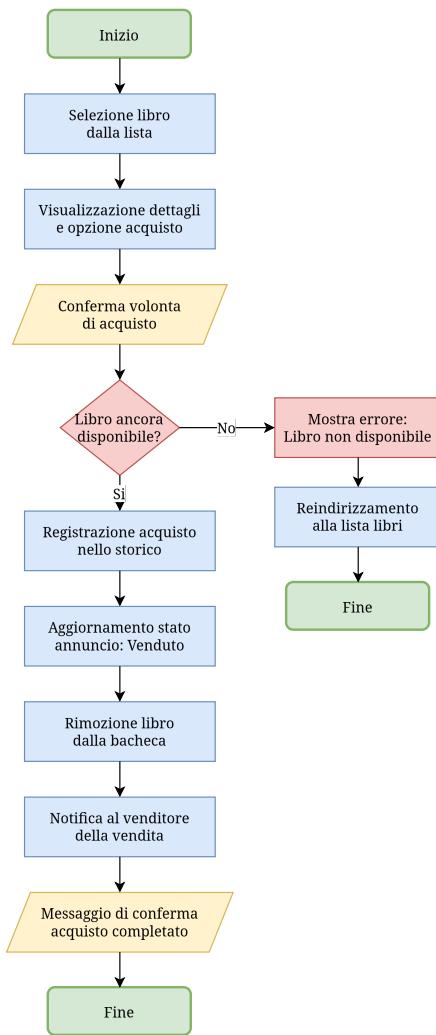


Figura 7.22: Activity Diagram - UC05 Acquisto

7.5.4 Criteri di Accettazione

- Un libro venduto non deve comparire nei risultati di ricerca.
- L'acquisto deve generare una voce nello storico “Miei Acquisti” dell'acquirente.
- L'acquisto deve generare una voce nello storico “Libri Venduti” del venditore.
- L'operazione deve essere atomica (se fallisce il salvataggio dello storico, il libro non deve risultare venduto).

7.5.5 Piano di Test Manuale

ID	Azione	Risultato Atteso
T01	Cliccare “Acquista” e poi “Annulla”	Nessuna modifica, libro resta disponibile
T02	Confermare l’acquisto	Successo, libro sparito dalla bacheca
T03	Verificare “Storico Acquisti”	Il libro appena comprato appare in cima
T04	Accesso diretto a libro già venduto	“Libro non disponibile”

Tabella 7.5: Piano di Test - UC05

7.5.6 Design Tecnico

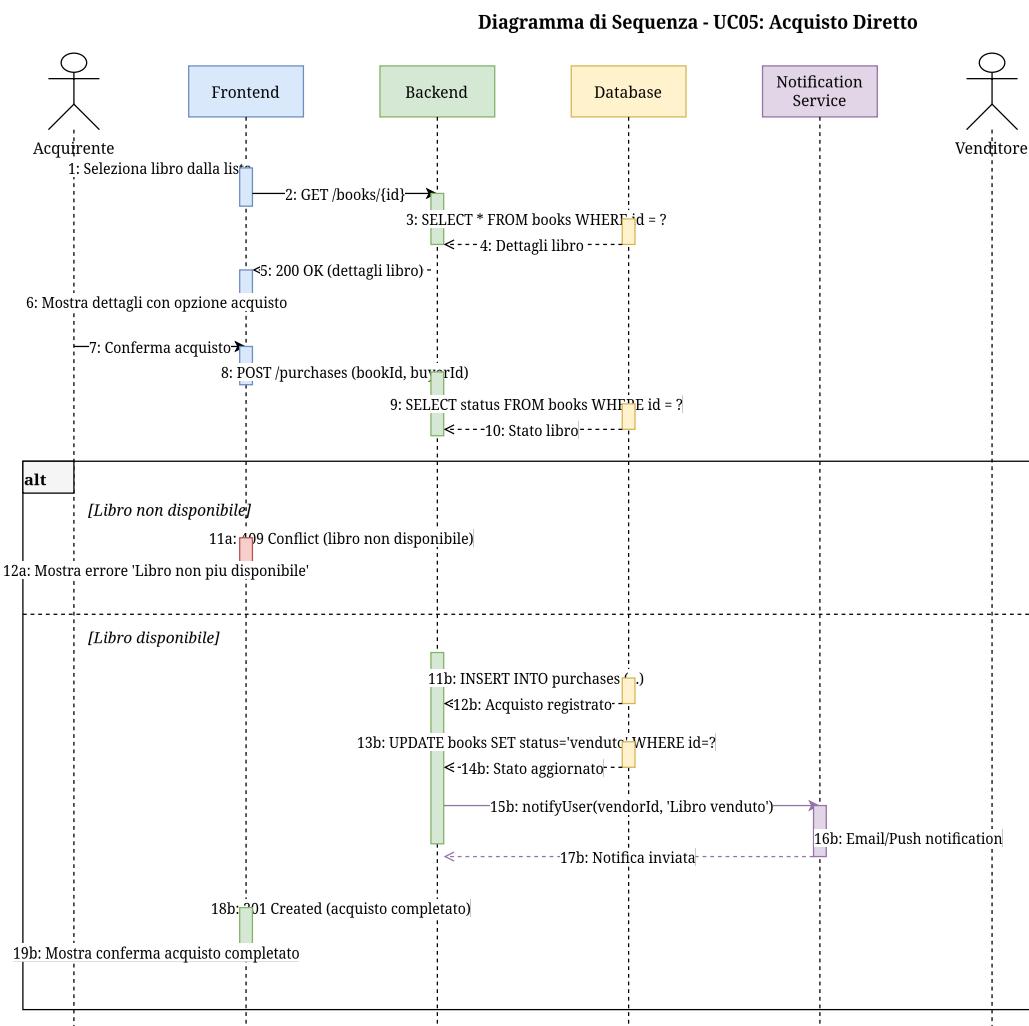


Figura 7.23: Sequence Diagram - UC05

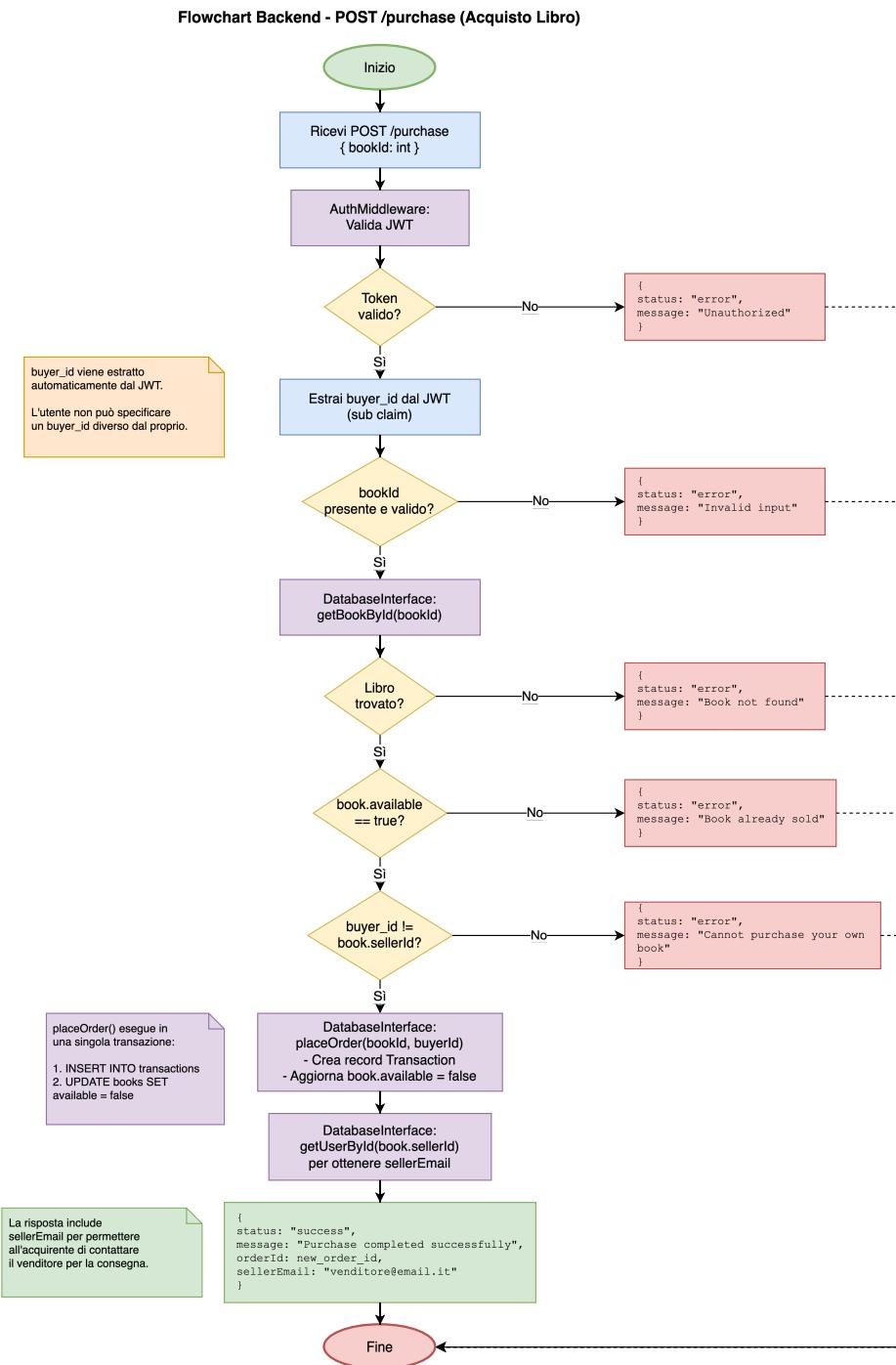


Figura 7.24: Backend Flowchart - UC05

Flusso di validazione:

1. **Autenticazione JWT:** Verifica che il token sia valido e non scaduto.
2. **Validazione Input:** Controllo presenza e formato del bookId.
3. **Esistenza Libro:** Verifica che il libro esista nel database.
4. **Disponibilità:** Controllo che book.available == true.
5. **Anti Auto-Acquisto:** Verifica che buyer_id != seller_id.

Operazione Atomica: La funzione placeOrder() esegue in una singola transazione database:

- Creazione record nella tabella `transactions`.
- Aggiornamento del campo `available = false` sul libro.

7.6 UC06 - Ricerca Avanzata

7.6.1 Panoramica

Descrizione: Consente a un Utente o Visitatore di cercare libri nella bacheca filtrando per ISBN, corso o docente. La ricerca viene eseguita lato Frontend sui dati già caricati.

Attori	Visitatore, Utente Autenticato
Pre-condizioni	La bacheca dei libri è stata caricata
Post-condizioni	Vengono visualizzati solo i libri che corrispondono ai criteri

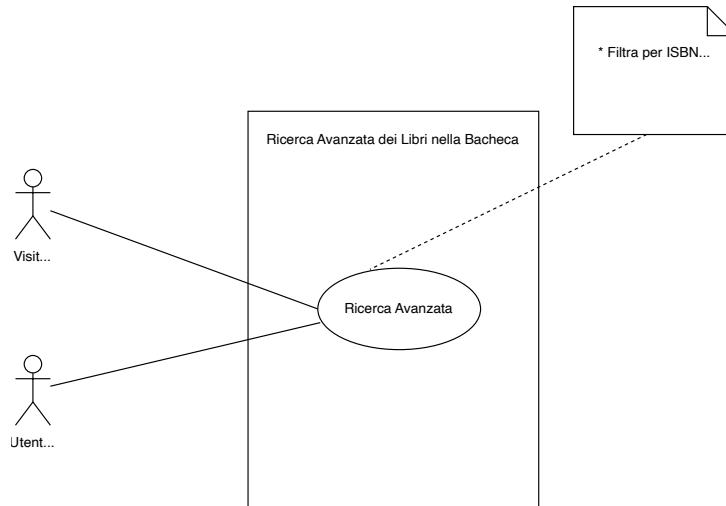


Figura 7.25: Use Case Diagram - UC06 Ricerca

7.6.2 Flussi di Eventi

Flusso Principale

1. L'Utente/Visitatore visualizza la bacheca con tutti i libri disponibili.
2. L'Utente/Visitatore inserisce uno o più criteri di ricerca (ISBN, corso, docente).
3. Il sistema (Frontend) filtra i risultati in tempo reale.
4. Il sistema mostra solo i libri che corrispondono ai criteri inseriti.

Flussi Alternativi

A1: Nessun risultato trovato

1. Nessun libro corrisponde ai criteri di ricerca.
2. Il sistema mostra un messaggio: "Nessun libro trovato per i criteri selezionati".

A2: Ricerca parziale

1. L'utente inserisce solo parte di un criterio (es. alcune cifre dell'ISBN).

2. Il sistema mostra tutti i libri che contengono la stringa parziale.

A3: Reset ricerca

1. L'utente svuota i campi di ricerca.
2. Il sistema mostra nuovamente tutti i libri disponibili.

7.6.3 Criteri di Accettazione

- La ricerca per ISBN deve restituire corrispondenze esatte o parziali.
- La ricerca per corso deve essere case-insensitive.
- La ricerca per docente deve essere case-insensitive.
- È possibile combinare più criteri di ricerca contemporaneamente.
- La ricerca deve funzionare sia per Visitatori che per Utenti autenticati.
- Svuotando i campi di ricerca devono ricomparire tutti i libri.

7.6.4 Piano di Test Manuale

ID	Azione	Risultato Atteso	Valida
T01	Inserire un ISBN completo	Solo il libro con quell'ISBN	✓
T02	Inserire un ISBN parziale	Libri il cui ISBN contiene quella sequenza	✓
T03	Inserire un nome corso esistente	Tutti i libri associati a quel corso	✓
T04	Inserire un nome docente	Tutti i libri associati a quel docente	✓
T05	Criteri senza corrispondenze	Messaggio “Nessun libro trovato”	✓
T06	Combinare ISBN + Corso	Solo libri che soddisfano entrambi	✓
T07	Svuotare tutti i campi	Tornano visibili tutti i libri	✓

Tabella 7.6: Piano di Test - UC06

7.6.5 Design Tecnico

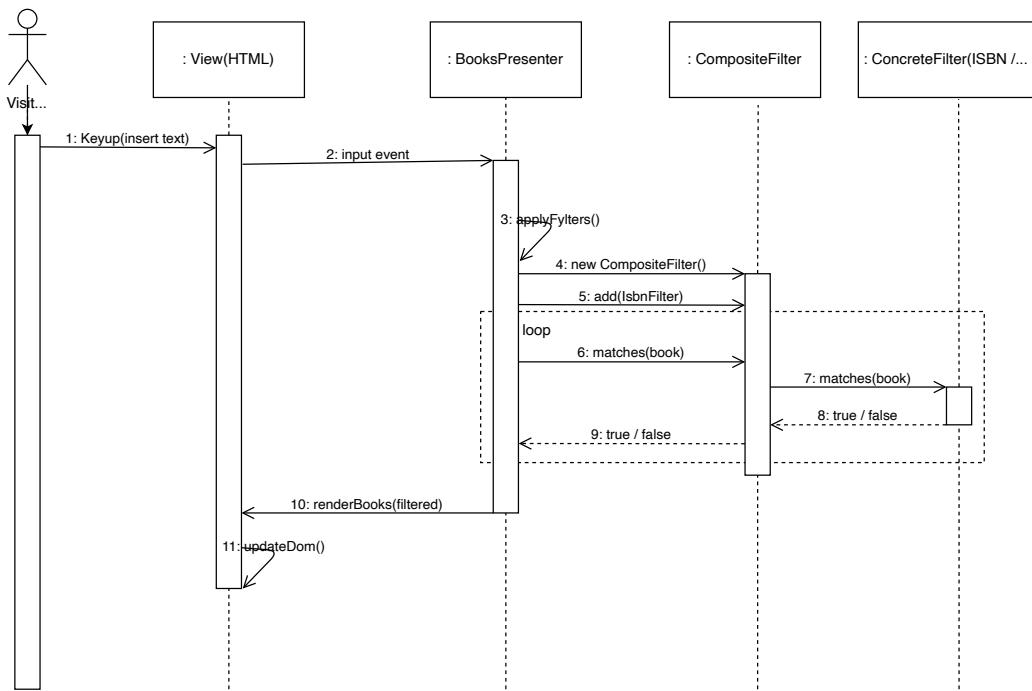


Figura 7.26: Sequence Diagram - UC06

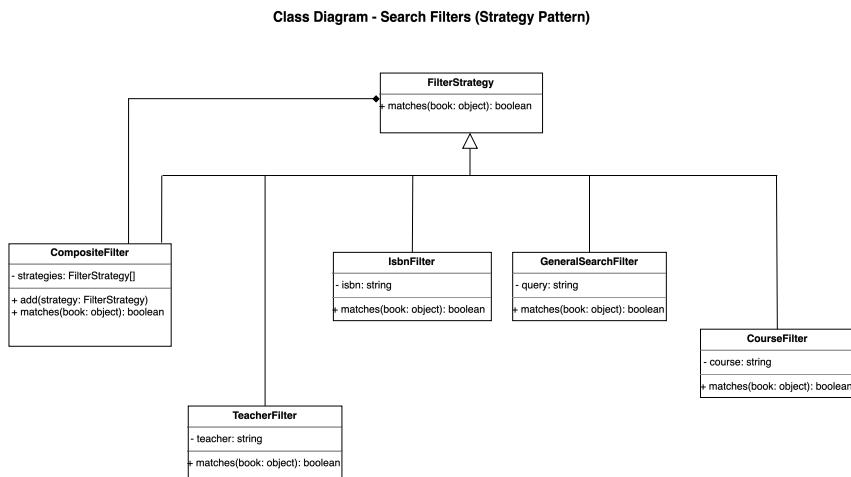


Figura 7.27: Class Diagram - Strategy Pattern per i Filtri

7.7 UC07 - Messaggistica tra Utenti

7.7.1 Panoramica

Descrizione: Consente a un Utente Autenticato di inviare e ricevere messaggi diretti da altri utenti, per accordarsi su prezzo, pagamento e consegna dei libri.

Attori	Utente Autenticato (Mittente), Utente Autenticato (Destinatario)
Pre-condizioni	Entrambi gli utenti sono registrati. Il mittente è autenticato
Post-condizioni	Il messaggio è salvato nel DB e visibile al destinatario

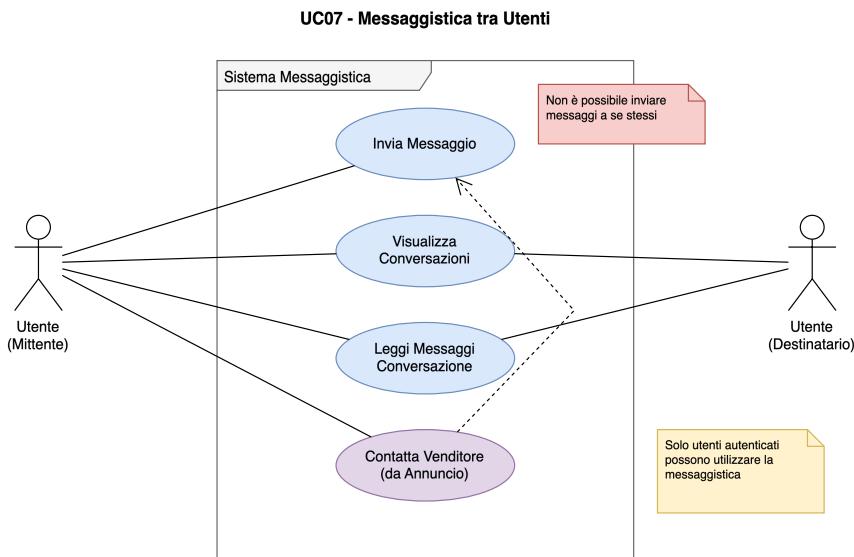


Figura 7.28: Use Case Diagram - UC07 Messaggistica

7.7.2 Flussi di Eventi

Flusso Principale

1. L'**Utente** accede alla sezione messaggi o clicca “Contatta venditore” da un annuncio.
2. Il sistema mostra la lista delle conversazioni esistenti o apre una nuova conversazione.
3. L'**Utente** scrive un messaggio nel campo di testo.
4. L'**Utente** clicca su “Invia”.
5. Il sistema (Backend) salva il messaggio nel database con mittente, destinatario e timestamp.
6. Il sistema aggiorna la conversazione mostrando il nuovo messaggio.

Flussi Alternativi

A1: Messaggio vuoto

1. L'utente tenta di inviare un messaggio senza contenuto.
2. Il sistema disabilita il pulsante “Invia” o mostra un errore.

A2: Destinatario non trovato

1. Il sistema non trova l'utente destinatario (es. account eliminato).
2. Il sistema mostra l'errore: “Impossibile inviare il messaggio. Utente non trovato”.

A3: Visualizzazione nuovi messaggi

1. L'utente accede alla sezione messaggi.
2. Il sistema evidenzia le conversazioni con messaggi non letti.

7.7.3 Activity Diagram

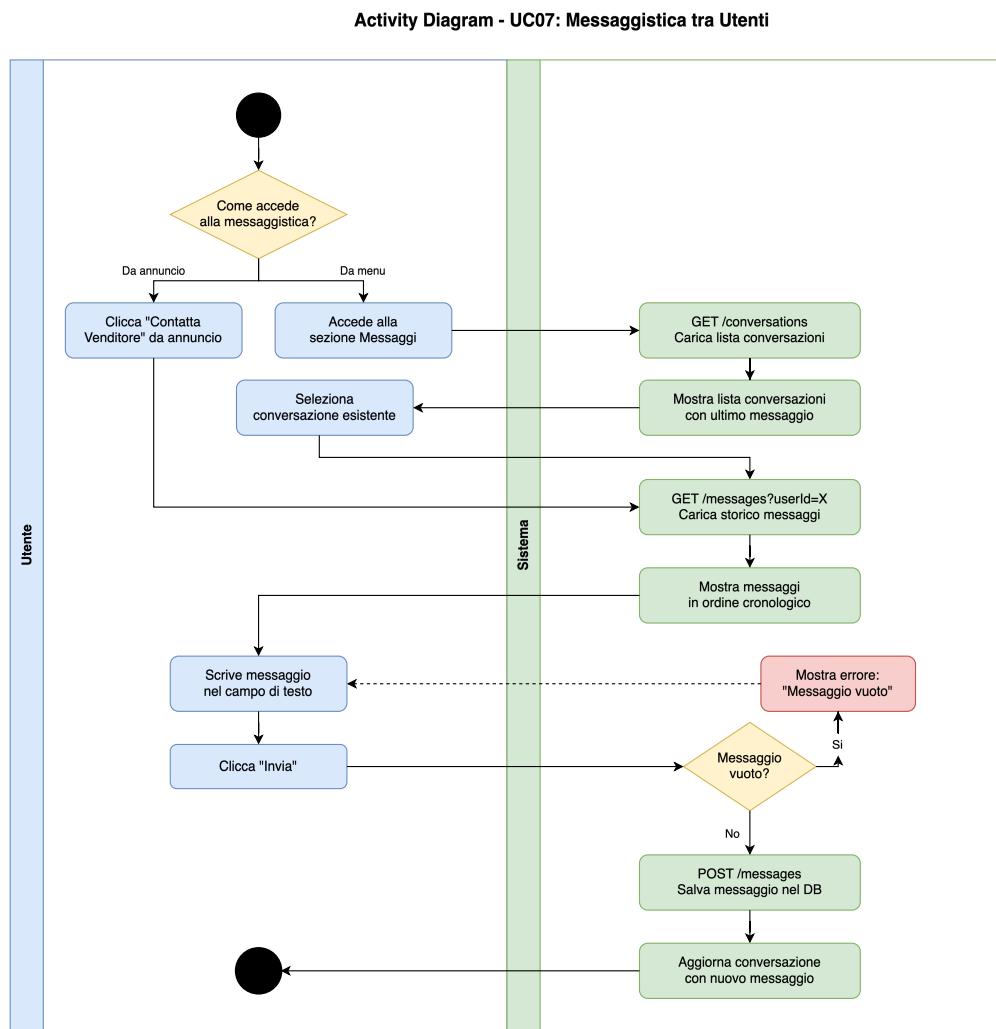


Figura 7.29: Activity Diagram - UC07 Messaggistica

7.7.4 Criteri di Accettazione

- Solo gli utenti autenticati possono inviare e ricevere messaggi.
- Ogni messaggio deve essere associato a un mittente e un destinatario.
- I messaggi devono essere ordinati cronologicamente nella conversazione.
- L'utente deve poter visualizzare lo storico dei messaggi con un altro utente.
- I messaggi non letti devono essere evidenziati o contrassegnati.
- Non è possibile inviare messaggi a se stessi.

7.7.5 Piano di Test Manuale

ID	Azione	Risultato Atteso
T01	Inviare un messaggio a un altro utente	Il messaggio appare nella conversazione di entrambi
T02	Tentativo di invio messaggio vuoto	Pulsante “Invia” disabilitato o errore
T03	Accedere con messaggi non letti	I messaggi non letti sono evidenziati
T04	Visualizzare lo storico di una conversazione	Tutti i messaggi in ordine cronologico
T05	Tentativo da utente non autenticato	Redirect alla pagina di login
T06	Contattare venditore da un annuncio	Si apre la conversazione con il venditore

Tabella 7.7: Piano di Test - UC07

7.7.6 Design Tecnico

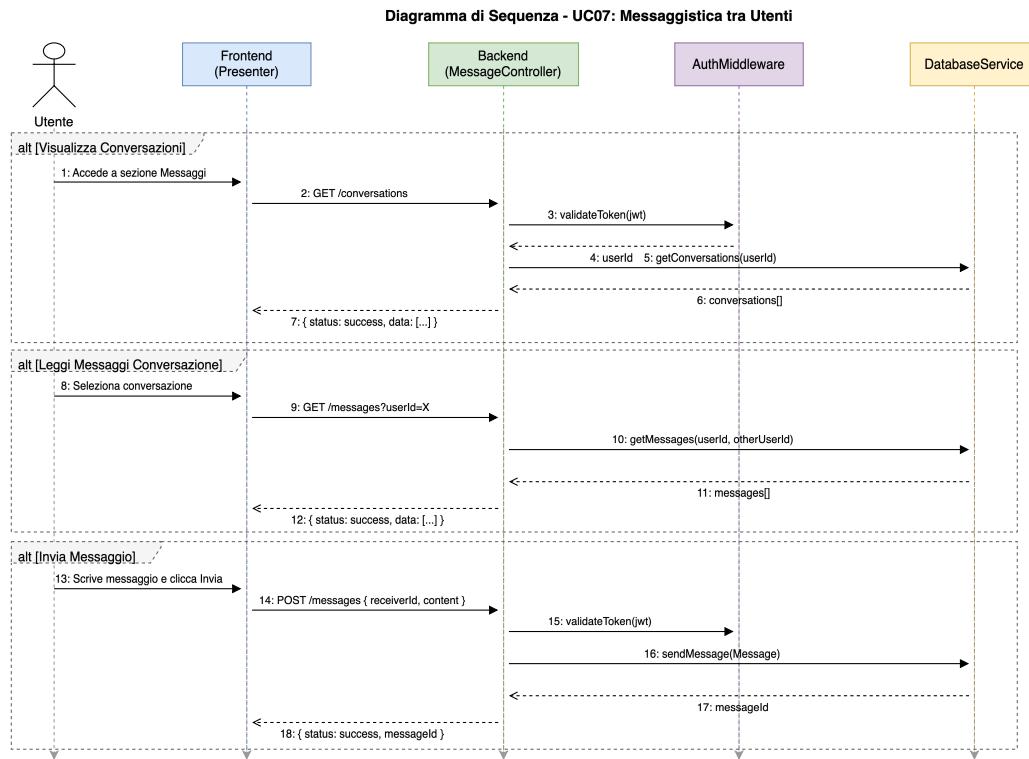


Figura 7.30: Sequence Diagram - UC07

UC07 - Backend Flowcharts: Messaggistica

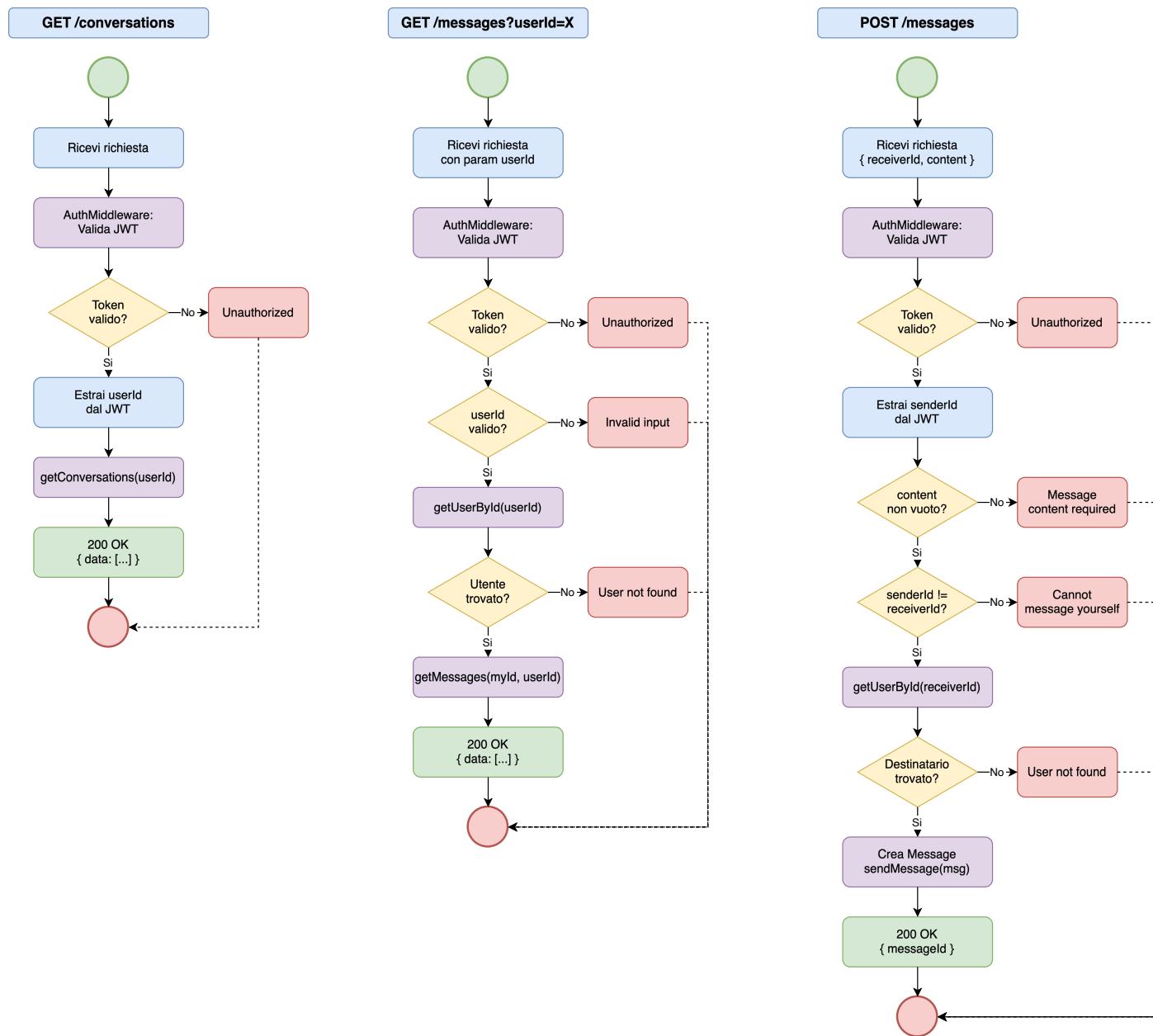


Figura 7.31: Backend Flowchart - UC07

Validazioni comuni a tutti gli endpoint:

1. **Autenticazione JWT:** Tutti gli endpoint richiedono un token JWT valido.
2. **Estrazione senderId:** Il mittente viene sempre estratto dal token, mai dal body della richiesta.

Validazioni specifiche per POST /messages:

1. **Contenuto non vuoto:** Il messaggio deve avere contenuto.
2. **Anti auto-invio:** Non è possibile inviare messaggi a se stessi (`SenderId != receiverId`).
3. **Destinatario esistente:** Verifica che l'utente destinatario esista nel sistema.