



## PROGETTO BIBLIOGRAFIA

Giorgio Longobardo N86003571  
Claudio Simonelli N86003781  
Giuseppe Francione N86003734

Anno Accademico 2022/2023

Docente:  
Di Martino  
Cutugno  
Starace

# Indice

<b>1</b>	<b>Introduzione</b>	<b>5</b>
1.1	Descrizione richiesta del progetto . . . . .	5
1.2	Stato progetto originale . . . . .	5
1.2.1	Basi di dati . . . . .	6
1.2.2	Applicativo Java . . . . .	6
1.3	Migliorie del progetto originale e nuove funzionalità . . . . .	7
<b>2</b>	<b>Requisiti Software</b>	<b>9</b>
2.1	Modellazione casi d'uso richiesti . . . . .	9
2.2	Individuazione target degli utenti . . . . .	10
2.3	Casi d'uso significativi nel dettaglio . . . . .	10
2.3.1	Caso d'uso: Ricerca Riferimenti . . . . .	11
2.3.2	Caso d'uso: Creazione Riferimento . . . . .	13
2.3.3	Caso d'uso: Modifica propri Riferimenti . . . . .	15
2.3.4	Caso d'uso: Crea Categoria . . . . .	17
2.4	Prototipazione Visuale . . . . .	18
2.4.1	Interfaccia Grafica progetto originale . . . . .	18
2.4.2	Interfaccia Grafica di Alexandria . . . . .	20
2.5	Valutazione dell'usabilità . . . . .	21
2.6	Classi, oggetti e relazioni d'analisi . . . . .	22
2.7	Diagrammi di Sequenza . . . . .	22
2.7.1	Diagramma di Sequenza: Ricerca di un Autore . . . . .	23
2.7.2	Diagramma di Sequenza: Creazione Riferimento . . . . .	24
2.8	Prototipazione funzionale . . . . .	25
2.8.1	Statechart: Ricerca Riferimenti . . . . .	25
2.8.2	Statechart: Creazione Riferimento . . . . .	25
2.8.3	Statechart: Modifica dei propri Riferimenti . . . . .	25
2.8.4	Statechart: Creazione Categoria . . . . .	26
<b>3</b>	<b>Design del Sistema</b>	<b>27</b>
3.1	Analisi dell'architettura e motivazioni . . . . .	27
3.2	Descrizione e motivazioni delle scelte tecnologiche adottate . . . . .	27
3.2.1	PostgreSQL . . . . .	27
3.2.2	Spring Boot JPA . . . . .	28
3.2.3	Flutter . . . . .	28
3.3	Diagramma delle classi di design . . . . .	29
3.4	Diagrammi di sequenza di design . . . . .	30
3.4.1	Ricerca Riferimenti . . . . .	30
3.4.2	Creazione Riferimento . . . . .	31
3.4.3	Modifica Riferimento . . . . .	32
3.4.4	Creazione Categoria . . . . .	33
3.5	Codice sorgente e Dockerfile . . . . .	33
<b>4</b>	<b>Testing e valutazione sul campo dell'usabilità</b>	<b>34</b>
4.1	Codice xUnit . . . . .	34
4.1.1	Unit Testing: Creazione Categoria . . . . .	34
4.1.2	Unit Testing: Ricerca di un Riferimento . . . . .	35
4.1.3	Unit Testing: Calcolo hash . . . . .	36
4.1.4	Unit Testing: Registrazione . . . . .	36

4.1.5	Strategie adottate per la progettazione dei test . . . . .	37
4.2	Valutazione dell'usabilità sul campo . . . . .	37
<b>5</b>	<b>Conclusione</b>	<b>38</b>
5.1	Note finali . . . . .	38

*Questa pagina è stata lasciata intenzionalmente vuota.*

# Capitolo 1

## Introduzione

### 1.1 Descrizione richiesta del progetto

È richiesto di migliorare e potenziare un sistema informativo già esistente per la gestione di bibliografie. Il sistema deve essere capace di salvare e organizzare i riferimenti bibliografici degli utenti. In particolare, è possibile inserire, modificare, rimuovere riferimenti bibliografici di diverso tipo (e.g.: articoli scientifici su conferenza o rivista, libri, risorse on-line, dataset, etc.). Ciascun riferimento è caratterizzato da un titolo univoco, un elenco di autori, una data, un URL (obbligatorio solo per risorse on-line), un DOI (facoltativo, ma univoco ove presente), e una descrizione testuale in cui l'utente può indicare aspetti significativi. Inoltre, un riferimento può essere associato a un insieme di rimandi, ovvero di altri riferimenti presenti nel sistema che vengono menzionati nel testo.

Un utente, infine, può definire un insieme di categorie personalizzate e possibilmente gerarchiche, e associare ciascun riferimento a una o più categorie. Per organizzazione gerarchica delle categorie si intende la possibilità di specificare che una certa categoria (e.g.: “Informatica”) ha una o più sottocategorie (e.g.: “Basi di Dati” o “Testing”).

Non è possibile introdurre dipendenze cicliche, ovvero non è possibile che una categoria sia una sottocategoria (anche transitivamente) di sé stessa. L'appartenenza a una sottocategoria implica l'appartenenza a tutte le sue super-categorie.

Non è pertanto possibile associare esplicitamente a un riferimento una categoria e una sua super-categoria.

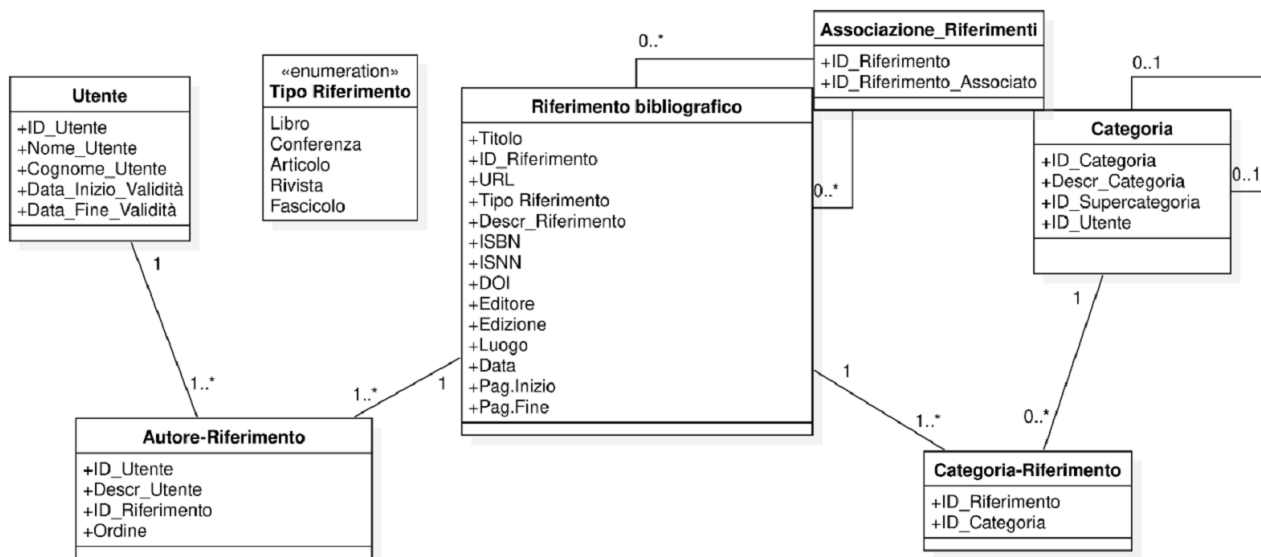
Il sistema permette infine di effettuare interrogazioni avanzate, con possibilità di filtraggio per una o più categorie, per data, per parole chiave e per autore. Inoltre, è possibile ordinare i riferimenti per numero di citazioni ricevute, ovvero per il numero di volte in cui il riferimento è presente nei rimandi di altri riferimenti.

Inoltre è richiesto lo sviluppo di nuove funzionalità da integrare nel sistema informativo già esistente.

### 1.2 Stato progetto originale

L'applicativo originale, seppur lasciato in ottimo stato, presenta alcuni punti deboli per quanto riguarda la progettazione software e usabilità. In particolare, verranno mostrati le varie funzionalità e i vari aspetti dell'applicativo che verranno modificati affinché possa rispettare gli standard odierni.

## 1.2.1 Basi di dati



La Basi di Dati originale è stata implementata nel modo seguente:

la tabella *Utente* descrive il possibile utente che accede alla piattaforma dei riferimenti bibliografici. Contiene un identificativo univoco, un nome e cognome e due date di inizio e fine validità rispettivamente.

La tabella *Autore-Riferimento* descrive il possibile ideatore o relatore in base a che tipologia di riferimento bibliografico si analizzi.

La tabella *Riferimento Bibliografico* descrive il possibile riferimento bibliografico e le sue caratteristiche in base alla tipologia.

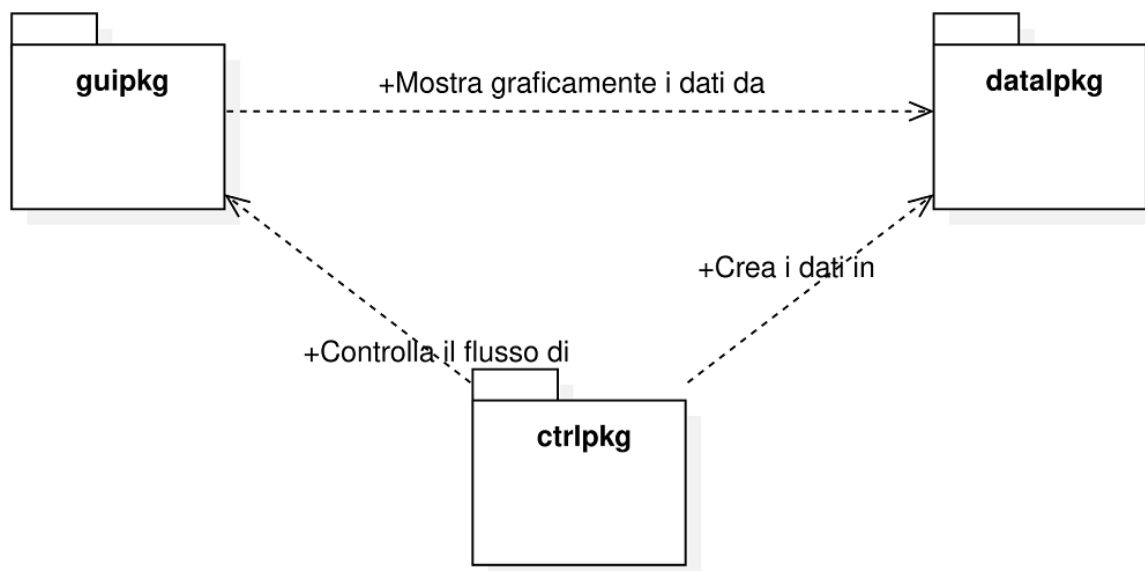
La tabella *Categoria* descrive una categoria e le sue possibili sottocategorie.

La tabella *Associazione-Riferimenti* è un descrittore di un riferimento che può essere associato a un insieme di rimandi.

La tabella *Categoria-Riferimento* è un descrittore di una categoria che è associata ad un riferimento.

## 1.2.2 Applicativo Java

L'approccio di design utilizzato è quello di un sistema Object Oriented sviluppato in Java che dipende strettamente da un database PostgreSQL. L'ambiente di sistema è un qualsiasi sistema operativo non-mobile (quindi desktop) fornito di una connessione al database.



Il sistema è costituito da 3 elementi principali, rappresentati in package:

*guipkg*, per la definizione delle interfacce grafiche e le loro interazioni;

*datalpkg*, per la definizione delle classi di dati che andranno trattati e mostrati;

*ctrlpkg*, per la definizione dei collegamenti e delle varie interazioni tra sistema e database esterno.

## 1.3 Migliorie del progetto originale e nuove funzionalità

La nuova versione del progetto prevede la modifica delle seguenti funzionalità:

- Nuovo sistema di accesso: il sistema non prevederà l'utilizzo dell'ID utente ma di una email e password apposita. Durante la registrazione verrà richiesto infatti di inserire le due informazioni che saranno poi salvate nel database. Inoltre, per preservare la sicurezza degli utenti, le password verranno criptate.
- Rimozione visibilità di ID accesso durante la registrazione: poiché l'utente non può più sapere il suo identificativo, non verrà mostrato il suo ID durante la registrazione.
- Potenziamento modalità di ricerca: la ricerca di riferimenti, citazioni e categorie verrà modificato e sarà più intuitivo ed efficiente.
- Apertura collegamenti: l'applicativo sarà capace di aprire gli URL inseriti per migliorare l'esperienza dell'utente, funzionalità mancante dell'applicativo originale.
- Impostazioni utente: verrà aggiunta la possibilità di modificare le proprie credenziali mediante un menù apposito.
- Miglioramento dell'interfaccia grafica: la GUI sarà totalmente ridisegnata per rispettare criteri di buona usabilità e con lo scopo di migliorare l'affordance iniziale, in tal modo da poter soddisfare più utenti possibili e di coprire tutte le possibili esigenze.

# Glossario

**Alexandria** Alexandria è un applicativo capace di gestire e creare i riferimenti bibliografici. 9

**attributo** Un attributo è un dettaglio di un riferimento e può essere della seguente natura: uno o più codici univoci, il titolo del riferimento, la data di pubblicazione, una descrizione del riferimento, il nome dell'autore, il nome della casa editrice, il numero di edizione ed eventuale titolo a cui si riferisce. 10

**autore** Un autore è una persona che ha prodotto un'opera da cui è poi tratto un riferimento presente nell'applicazione. 9

**casi d'uso** Un caso d'uso è una delle funzionalità principali del sistema. Senza di esse, l'applicativo non sarebbe completo e permette di svolgere una delle attività principali richieste. 9

**categoria** Una categoria è un insieme contenente più riferimenti che trattano la stessa collezione di argomenti. 9

**DBMS** Un DBMS (Database management system) è un sistema di gestione di una base di dati, in particolare per la creazione, manipolazione e ricerca dei dati. In Alexandria, si riferisce a PostgreSQL, un DBMS di tipo relazionale.. 27

**Front-End** Il front end è l'insieme di tutte le attività e operazioni che l'utente finale deve compiere o vedere.. 27

**HTTP** HTTP è un protocollo per la comunicazione delle informazioni fra client-server. In Alexandria, essa è utilizzata per la comunicazione fra l'applicativo e il DBMS. 27

**Layer** Un layer è uno strato dell'architettura del sistema. Può essere gerarchico e svolge una determinata attività o compito.. 27

**REST** Representational state transfer è uno stile architetturale per sistemi distribuiti. In questo caso, è la rappresentazione dell'Architettura di Alexandria.. 27

**riferimento** Un riferimento è un collegamento ad un'opera bibliografica esistente dotato di nome, data, descrizione, codici univoci e di un eventuale URL per la visualizzazione. 9

**sopra-categoria** Una sopra-categoria è un insieme contenente una o più categorie. La relazione che accomuna le categorie dell'insieme può essere varia e a scelta dell'utente. Infine, una sopra-categoria non può contenere se stessa o una categoria che ha già tale sopra-categoria come sopra-categoria. 10

**Spring Boot** Spring Boot è un'API rest per l'implementazione, appunto, di un applicativo basato su REST.. 28

**tipo di ricerca** Il tipo di ricerca determina in *quale modo* debba essere ricercato un determinato riferimento e può essere per titolo (mostra solo i riferimenti avente tale titolo), per autore (mostra tutti i riferimenti di quel determinato autore) e per DOI (mostra i riferimenti avente quel determinato DOI). 11

**tipo di riferimento** Un tipo di riferimento è il tipo di formato del riferimento pubblicato e può assumere diversi tipi, ovvero: un libro, un articolo, un fascicolo, una rivista, una conferenza. La conferenza può essere di formato visivo virtuale. 11

**Weak Equivalence Class Testing** La WECT è un processo di copertura che stabilisce che per ogni classe di equivalenza ci deve essere un Test case che usa un valore nominale da quella classe di equivalenza.. 37



## Capitolo 2

# Requisiti Software

### 2.1 Modellazione casi d'uso richiesti

All'interno della nostra applicazione rimodernizzata, da qui in avanti chiamata Alexandria, abbiamo individuato 6 casi d'uso: un caso d'uso relativo all'autenticazione, un caso d'uso relativo alla ricerca di un riferimento e di un autore, un caso d'uso relativo alla creazione dei riferimenti, un caso d'uso relativo alla creazione di una categoria, un caso d'uso relativo alla visualizzazione e creazione modifica dei propri riferimenti e infine caso d'uso relativo alle impostazioni utente.



Spiegazione dettagliata dei casi d'uso mostrati:

- Il caso d'uso *Accesso* permette l'autenticazione di un utente, ovvero permette ad un utente di accedere al sistema inserendo le proprie credenziali (scelte dall'utente stesso durante la fase di registrazione).
- Il caso d'uso *Registrazione* permette di registrare un nuovo utente al sistema, scegliendo un proprio username, una propria password e una email.
- Il caso d'uso *Ricerca Riferimenti* permette ad un utente di cercare un riferimento esistente nel sistema. Se inesistente, il sistema notifica l'utente dell'inesistenza del riferimento cercato, altrimenti permette di visualizzarlo.
- Il caso d'uso *Modifica Riferimenti* permette ad un utente di modificare un riferimento creato in precedenza, in particolare permette di modificare un attributo inserito in precedenza, potendo scegliere un nuovo valore.
- Il caso d'uso *Ricerca autori* permette all'utente di ricercare un autore in particolare e tutte le sue opere pubblicate presenti nel sistema.
- Il caso d'uso *Crea categoria* permette all'utente di creare una categoria e di poter scegliere un'eventuale sopra-categoria.
- Il caso d'uso *Crea Riferimenti* permette ad un utente di creare un riferimento e di poterne assegnare gli attributi.
- Infine, il caso d'uso *Modifica Impostazioni utente* permette ad un utente di modificare tutte le informazioni inserite durante la fase di registrazione.

Tutte queste funzionalità richiedono un attore esterno, ovvero il Server, il quale permette di registrare ogni modifica al sistema. Sostanzialmente, senza di esso l'applicativo non può funzionare correttamente.

## 2.2 Individuazione target degli utenti

Il target principale degli utenti sono coloro i quali intendono gestire e visualizzare i propri riferimenti bibliografici. Per tale motivo Alexandria permette la gestione e la visualizzazione affidabile dei riferimenti creati. In aggiunta, è possibile visualizzare i riferimenti degli altri utenti presenti nel sistema. Un altro possibile target di utenti sono gli autori stessi dei riferimenti bibliografici, poiché possono gestire facilmente le proprie opere pubblicate e visualizzarne gli attributi. Un altro target di utenti sono le case editrici che intendono gestire le varie edizioni dei propri riferimenti pubblicati. Considerando tutti i possibili utenti, il team si impegna di poter soddisfare tutte le esigenze degli utenti e di poter garantire un'eccellente usabilità e affidabilità del sistema.

## 2.3 Casi d'uso significativi nel dettaglio

Vengono qui riportati quattro casi d'uso significativi nel dettaglio utilizzando il template di Cockburn. La sezione *Descrizione*, presente in tutte le tabelle, indica lo scenario di successo. Le estensioni indicano uno scenario di fallimento o tipi di errore. Lo scenario sottovariante indica una variante del caso di successo.

### 2.3.1 Caso d'uso: Ricerca Riferimenti

Caso d'Uso 1	Ricerca di un riferimento		
Obiettivo	L'obiettivo principale è quello di ricercare uno o più riferimenti inserendo attributi specifici		
Precondizioni	L'utente deve essere stato correttamente registrato in precedenza.		
Condizioni di successo	L'utente trova e visualizza il riferimento desiderato		
Condizioni di fallimento	L'utente non trova il riferimento desiderato poiché inesistente		
Attore principale	Utente registrato		
Trigger	Utente preme su <i>Ricerca</i> nella Homepage.		
Descrizione	Step	Attore	Sistema
	1	Inserisce titolo del riferimento	
	2	Seleziona uno dei tipo di riferimento disponibili	
	3		Seleziona i tipi di riferimento scelti
	4	Cerca una categoria apposita	
	5		Mostra le categorie cercate dall'utente
	6	Seleziona il tipo di ricerca	
	7		Deselziona i tipi di ricerca non scelti dall'utente
	8	Preme su ricerca	
	9		Mostra frame <i>Risultati ricerca</i>
	10	Seleziona ordine ricerca	
	11		Ordina ricerca per il tipo selezionato dall'utente
	12	Seleziona un riferimento	
	13		Mostra frame <i>Visualizza Citazione</i>
Note	I nomi dei <i>frame</i> provengono dai MockUp realizzati su Figma.		

Extension A: L'utente torna indietro	Step	Attore	Sistema
	D.1 a D.7	Preme su <i>Indietro</i>	
	A.1		Mostra frame <i>Home-Page</i>
Extension B: L'utente non inserisce il testo	Step	Attore	Sistema
	D.1 a D.7	Preme su ricerca	
	B.1		Mostra tutti i riferimenti esistenti
Extension C: Non esistono i riferimenti ricercati	Step	Attore	Sistema
	D.8 + D.9	Preme su ricerca	
	C.1		Mostra frame <i>Ricerca Vuota</i>
	C.2	Preme su Crea	
	C.3		Mostra frame <i>Crea Modifica Citazione</i>
	C.2	Seleziona Indietro	
	C.4		Ritorna a frame <i>Ricerca</i>
Note	I nomi dei <i>frame</i> provengono dai MockUp realizzati su Figma.		

### 2.3.2 Caso d'uso: Creazione Riferimento

Caso d'Uso 2	Creazione di un Riferimento		
Obiettivo	L'obiettivo principale è quello di creare un nuovo riferimento visualizzabile per tutti gli utenti.		
Precondizioni	L'utente deve essere stato correttamente registrato in precedenza		
Condizioni di successo	Viene creato un nuovo riferimento nel sistema, visualizzabile per tutti gli utenti.		
Condizioni di fallimento	Il riferimento è già esistente nel sistema.		
Attore principale	Utente registrato		
Trigger	Utente preme su <i>Crea Riferimento</i> nella barra di controllo		
Descrizione	Step	Attore	Sistema
	1	Inserisce titolo citazione	
	2	Preme sull'icona <i>Data</i>	
	3		Mostra dialog <i>Crea Modifica Citazione Data</i>
	4	Sceglie una data e preme Ok	
	5		Ritorna al frame <i>Crea Modifica Citazione</i>
	6	Preme su Riferimento a	
	7		Mostra dialog <i>Crea Modifica Citazione Riferimento a</i>
	8	Cerca o sceglie un riiferimento e preme Ok	
	9		Ritorna al frame <i>Crea Modifica Citazione</i>
	10	Seleziona il tipo di riferimento	
	11		Deseleziona i tipi di riferimento non scelti dall'utente
	12	Preme su Descrizione	
	13		Mostra dialog <i>Crea Modifica Citazione Descrizione</i>
	14	Inserisce una descrizione, un DOI, una edizione e numero delle pagine e preme Ok	
	15		Ritorna al frame <i>Crea Modifica Citazione</i>
	16	Preme su Link	
	17		Mostra dialog <i>Crea Modifica Citazione Link</i>
	18	Inserisce URL e preme Salva	
	19		Ritorna al frame <i>Crea Modifica Citazione</i>
	20	Inserisce i valori per <i>Editore</i> , <i>Luogo</i> , <i>ISSN</i> e <i>ISBN</i> e preme su conferma	
	21		Mostra frame <i>Crea Modifica Citazione Successo</i>
	22	Preme su Visualizza	
	23		Mostra frame <i>Visualizza Citazione</i>
Note	I dialog sono descritti come <i>Frame</i> su Figma.		

Extension A: Riferimento già esistente	Step	Attore	Sistema
	D.20		Mostra dialog <i>Crea Modifica Citazione Errore</i>
	A.1	Attende pochi secondi	
	A.2		Ritorna sul frame <i>Crea Modifica Citazione</i>
Extension B: Riferimento connesso non esistente	Step	Attore	Sistema
	D.7	Inserisce un riferimento non esistente e preme Ok	
	B.1		Mostra snackbar <i>Errore Modifica Citazione Riferimento a</i>
Extension C: Riferimento nullo	Step	Attore	Sistema
	D.7	Lascia un campo vuoto e preme Ok	
	C.1		Mostra snackbar <i>Errore Modifica Citazione Riferimento a</i>
Extension E: URL non valido	Step	Attore	Sistema
	D.17	Inserisce un link non valido e preme Ok	
	E.1		Mostra snackbar <i>Errore Crea Modifica Citazione Link</i>
Extension F: Descrizione non valida	Step	Attore	Sistema
	D.13	Inserisce valori non validi per la descrizione, pagine, edizione o DOI e preme Ok	
	F.1		Mostra snackbar <i>Errore Crea Modifica Citazione Descrizione</i>
	F.2		Scomparsa della snackbar
Extension G: Campi tutti vuoti	Step	Attore	Sistema
	G.1	Non inserisce nessun valore e preme su Conferma	
	G.2		Mostra dialog <i>Crea Modifica Citazione Errore</i>
	G.3		Ritorna sul frame <i>Crea Modifica Citazione</i>
Note			

### 2.3.3 Caso d'uso: Modifica propri Riferimenti

Caso d'Uso 3	Modifica dei propri riferimenti creati		
Obiettivo	Poter modificare un proprio riferimento creato e poter visualizzare le modifiche correttamente.		
Precondizioni	Utente deve essere correttamente registrato		
Condizioni di successo	Modificare correttamente il riferimento		
Condizioni di fallimento	I nuovi valori inseriti non sono validi		
Attore principale	Utente registrato		
Trigger	L'utente preme su <i>Mie Citazioni</i>		
Descrizione	Step	Attore	Sistema
	1	Cerca titolo di una citazione	
	2		Mostra l'eventuale riferimento specificato
	3	Preme su Ordina per	
	4		Ordina i riferimenti per l'ordine specificato
	5	Seleziona un riferimento	
	6		Evidenzia il riferimento selezionato
	7	Preme su Modifica	
	8		Mostra dialog <i>Visualizza Propri Riferimenti Conferma Modifica</i>
	9	Preme su Modifica	
	10		Mostra frame <i>Crea Modifica Riferimento</i>
	11	Modifica il proprio riferimento e preme su Conferma	
	12		Mostra dialog <i>Crea Modifica Citazione Successo</i>
	13	Preme su visualizza	
	14		Mostra frame <i>Visualizza Citazione</i>
Note	I dettagli sulla modifica dei valori sono stati omessi.		
Note	Funzionamento analogo alla creazione di un riferimento. Si consulti la sezione 2.3.2.		

Extension A: L'utente seleziona non seleziona un riferimento	Step	Attore	Sistema
	D.7		Mostra dialog <i>Errore Visualizza Propri Riferimenti Selezione</i>
	A.1		Ritorna su <i>Visualizza Propri Riferimenti</i>
Extension B: L'utente cerca un riferimento inesistente	Step	Attore	Sistema
	D.7		Mostra dialog <i>Errore Visualizza Propri Riferimenti Selezione</i>
	B.1		Ritorna su <i>Visualizza Propri Riferimenti</i>
Extension C: L'utente modifica il riferimento inserendo valori non validi	Step	Attore	Sistema
	D.11		Mostra dialog <i>Crea Modifica Citazione Errore</i>
	C.1		Ritorna su <i>Crea Modifica Citazione</i>
Sottovariante: L'utente elimina un riferimento	Step	Attore	Sistema
	D.5	Clicca su Elimina	
	S.1		Mostra dialog <i>Visualizza Propri Riferimenti Conferma Eliminazione</i>
	S.2	Clicca su Elimina	
	S.3		Mostra dialog <i>Visualizza Propri Riferimenti Ok Eliminazione</i>
	S.4		Ritorna su <i>Visualizza Propri Riferimenti</i>
Note			



### 2.3.4 Caso d'uso: Crea Categoria

Caso d'Uso 4	Crea una categoria		
Obiettivo	L'obiettivo principale è quella di creare una nuova categoria. Se ha sottocategorie, non deve essere sottocategoria di se stessa, anche transitivamente.		
Precondizioni	Utente deve essere correttamente registrato		
Condizioni di successo	Creare una categoria. Se ha sottocategorie, non deve essere sottocategoria di se stessa		
Condizioni di fallimento	Creare una categoria che sia una sottocategoria di se stessa		
Attore principale	Utente registrato		
Trigger	L'Utente preme su <i>Crea Categoria</i>		
Descrizione	Step	Attore	Sistema
	1	Scriva il titolo della categoria	
	2	Cerca una sottocategoria	
	3	Seleziona una sottocategoria	
	4	Preme su Info	
	5		Mostra <i>Visualizza Categoria</i> della categoria selezionata
	6	Preme Indietro	
	7		Mostra <i>Crea Categoria</i>
	8	Preme su Salva	
	9		Mostra dialog <i>Crea Categoria Successo</i>
	10	Preme su Visualizza	
	11		Mostra frame <i>Visualizza Categoria</i>

Extension A: Inserisce titolo non valido	Step	Attore	Sistema
	D.7		Mostra <i>Errore Crea Categoria</i>
	A.1		Mostra <i>Crea Categoria</i>
Extension B: Cerca o seleziona una sottocategoria inesistente	Step	Attore	Sistema
	D.7		Mostra <i>Errore Crea Categoria</i>
	B.1		Mostra <i>Crea Categoria</i>
Extension C: Utente preme su indietro	Step	Attore	Sistema
	D.9	Preme su indietro	
	C.1		Mostra <i>Crea Categoria</i>
Note			

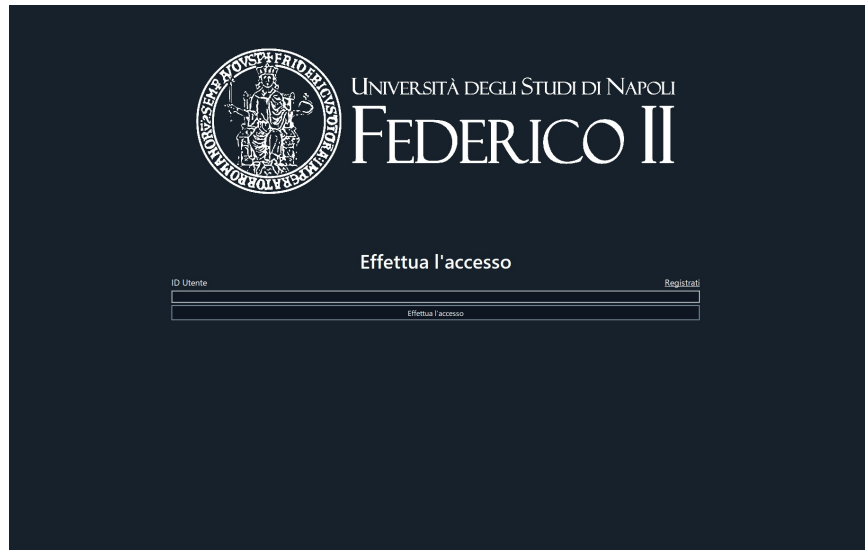
## 2.4 Prototipazione Visuale

L'interfaccia grafica della nuova applicazione prende ispirazione dall'applicativo originale, migliorandolo però dal punto di vista dell'usabilità.

### 2.4.1 Interfaccia Grafica progetto originale

Mostriamo ora alcune schermate dell'applicativo originale spiegando i punti che poi andranno modificati per rispettare un buon criterio di ingegnerizzazione.

Innanzitutto, la schermata di accesso iniziale:



The screenshot shows the login interface of the University of Naples Federico II. At the top left is the university's seal, and to its right is the text "UNIVERSITÀ DEGLI STUDI DI NAPOLI" and "FEDERICO II" in a large, serif font. Below this, the text "Effettua l'accesso" is centered. Underneath, there is a form with two input fields: "ID Utente" on the left and "Registra" on the right. A "Effettua l'accesso" button is located below the "ID Utente" field.

È possibile effettuare l'accesso mediante solamente l'ID utente. Come già accennato in precedenza, il nuovo sistema di accesso prevede l'utilizzo di credenziali univoche per ogni utente. Da qui, inserendo numeri casuali, è possibile effettuare l'accesso con utenti di cui non si dovrebbero disporre le credenziali. Inoltre, manca un eventuale aiuto se l'utente dovesse dimenticarsi delle proprie credenziali. Infine, l'utente non dovrebbe vedere il proprio ID essendo un'informazione prettamente riservata nel database.

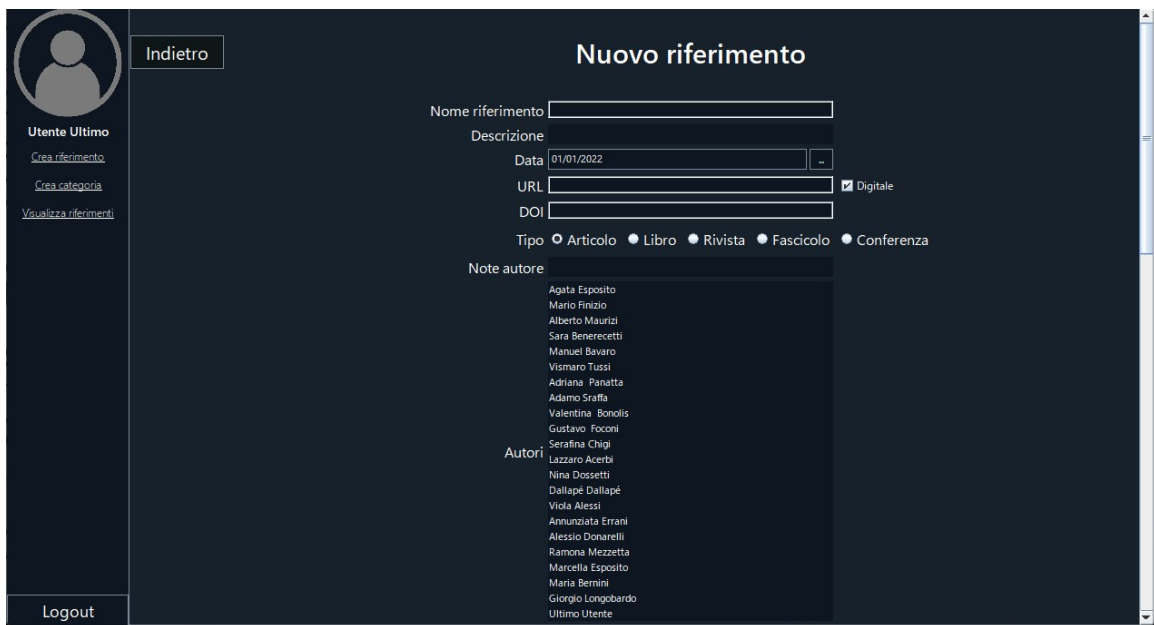


The screenshot shows the registration interface of the University of Naples Federico II. At the top left is the university's seal, and to its right is the text "UNIVERSITÀ DEGLI STUDI DI NAPOLI" and "FEDERICO II" in a large, serif font. Below this, the text "Registrati" is centered. Underneath, there is a form with three input fields: "ID Utente" (containing the number "22"), "Nome", and "Cognome". A "Crea utente" button is located below the "Cognome" field. In the top left corner, there is a button labeled "Indietro".

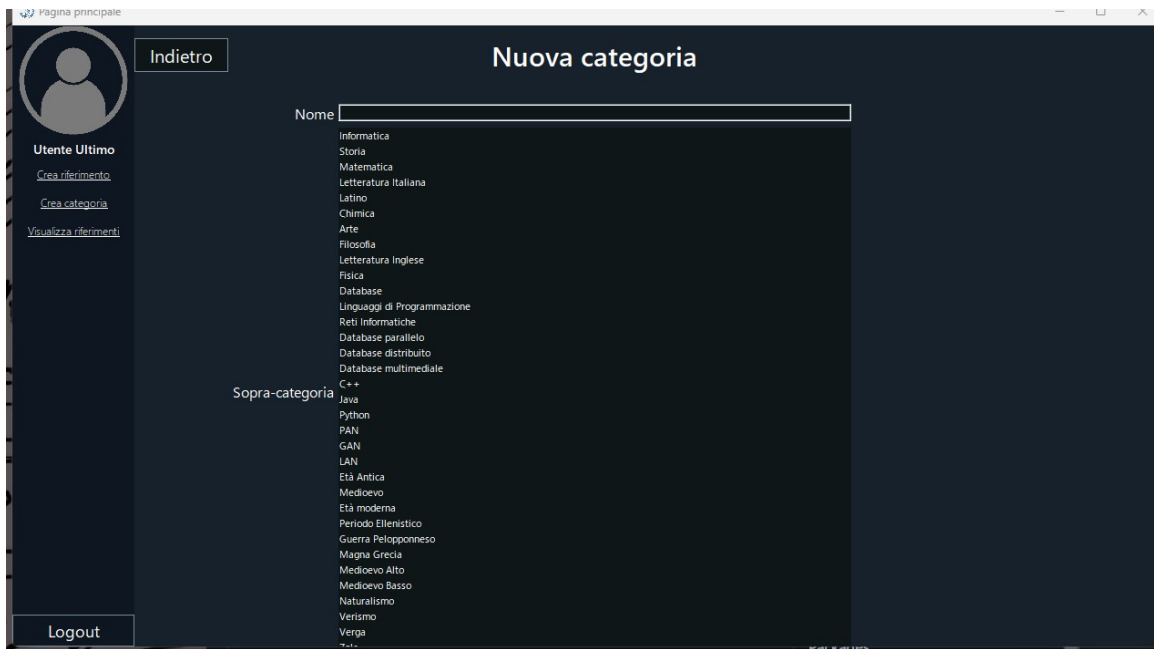
In fase di registrazione, il futuro nuovo utente deve inserire solamente il proprio cognome e nome, mentre l'ID sarà solamente visualizzato. Se l'utente non dovesse vedere il proprio numero, o non capire come effettuare l'accesso successivamente, perderebbe le proprie credenziali inaspettatamente. Dunque, la schermata di registrazione del nuovo applicativo chiederà all'utente, oltre al proprio nome e cognome, una email e una password, da usare successivamente. Inoltre, una volta completata la registrazione, il vecchio sistema non effettua in automatico il login, funzionalità abbastanza comoda che viene introdotta in Alexandria.



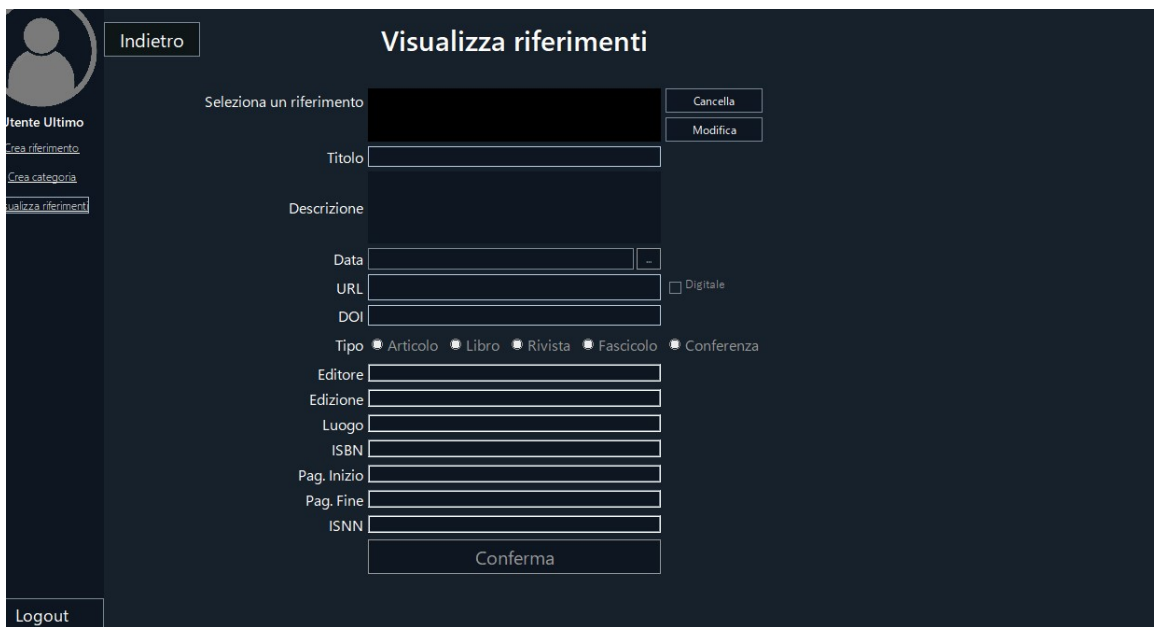
La HomePage del vecchio applicativo ci è sembrato piuttosto ambiguo su alcuni aspetti: che cosa si intende per *Ultimi 5 riferimenti*? Gli ultimi 5 riferimenti creati dall'utente inseriti, gli ultimi 5 riferimenti visualizzati dall'utente, oppure ancora gli ultimi 5 riferimenti inseriti da altri utenti? Discorso analogo per quanto riguarda le citazioni. Abbiamo quindi interpretato che questi fossero delle cronologie delle citazioni e riferimenti inserite dall'utente stesso. Inoltre, si cercherà di migliorare l'affordance generale della schermata, facendo capire a intuito dove inserire il testo per cercare e come cercare.



La schermata originale della creazione di un nuovo riferimento ci è risultata piuttosto confusoria e alcune sezioni troppo grandi (come la lista degli autori) o troppo piccole (come quella della descrizione). Il nuovo applicativo cercherà di raggruppare tutte queste informazioni in spazi più piccoli e più intuitivi, mostrando solamente le informazioni necessarie per l'inserimento.



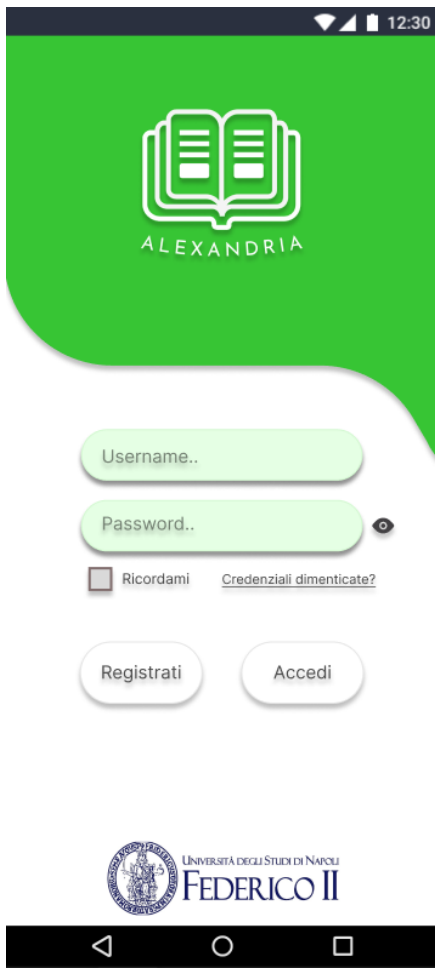
Discorso analogo vale per la creazione di una categoria, ci è sembrato eccessivo mostrare tutte le sopra-categorie possibili. Essendo molte risulta poi difficoltoso per un utente ricercare quella desiderata.



La schermata di visualizzazione dei riferimenti è ambigua nel suo nome: non solo è possibile visualizzare tutti i propri riferimenti creati, ma anche modificarli ed eliminarli. Verrà quindi descritto il suo scopo.

## 2.4.2 Interfaccia Grafica di Alexandria

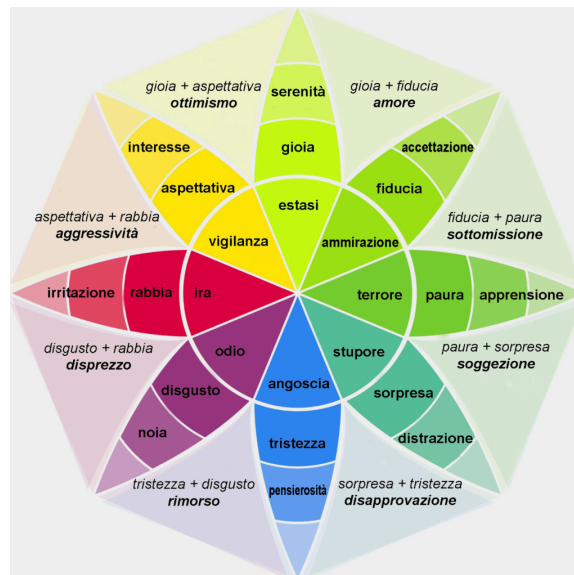
Queste sono alcune delle nuove schermate prototipizzate:



La schermata di accesso e la schermata di homepage. Se si volesse consultare la prototipizzazione completa basta consultare questo link cliccabile, dove è annesso il flow dell'applicazione e tutti i frame completi, ognuno realizzato con Figma.

## 2.5 Valutazione dell'usabilità

Come già ribadito in precedenza, la nuova interfaccia grafica di Alexandria è stata pensata cercando di rispettare tutti i principi di buona usabilità. Uno dei cambiamenti principali è stato il tema principale dell'applicazione. Attraverso la ruota di Plutchik abbiamo deciso di assegnare alla nostra nuova applicazione un nuovo colore: il verde chiaro e le sue variazioni.

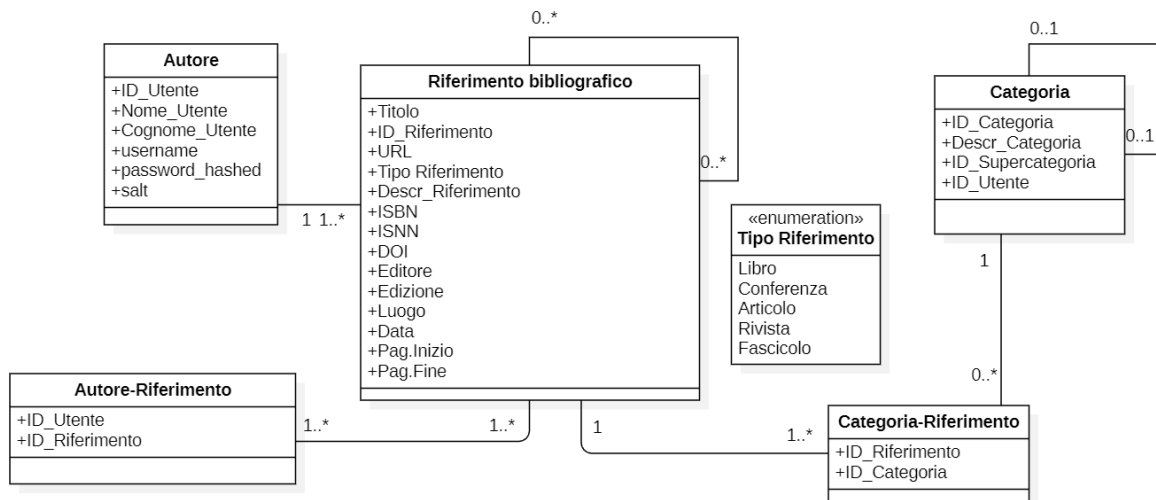


Essendo un'applicazione che deve gestire i propri riferimenti, è importante che tale applicazione esprima *Fiducia* nei confronti dell'utente, e consultando la ruota delle emozioni il verde ci è sembrato il colore più adatto alla scelta del tema principale dell'applicazione.

Abbiamo inoltre deciso che a schermo non verranno mostrate più informazioni del necessario se non richiesto. Le informazioni presenti sull'interfaccia grafica sono minimali e riassuntive, per non caricare troppo la memoria a breve termine dell'utente. Inoltre, le icone sono state pensate per rispettare l'uso comune di esse e abbiamo tentato di rendere ogni icona autoesplicativa. Considerato che l'applicativo deve girare su dispositivi mobili, l'interfaccia è stata pensata per apparire su schermi più piccoli, e che quindi l'utente ha una percezione minore rispetto a un monitor: per ovviare a questo problema abbiamo tentato di rendere i pulsanti che creino contrasto con lo sfondo, abbiamo ridotto la quantità di testo per non occupare troppo spazio su schermo, abbiamo preferito uno stile che non sia troppo sfarzoso e ingombrante. Infine, in caso di più pulsanti, essi sono posti distanziati (nei limiti, ovviamente, dei vincoli dell'usabilità e visivi) per evitare che si preme accidentalmente un bottone anziché di un altro, penalizzando l'esperienza dell'utente.

## 2.6 Classi, oggetti e relazioni d'analisi

Alla luce delle nuove funzionalità da inserire, vengono apportate queste nuove modifiche al Database:

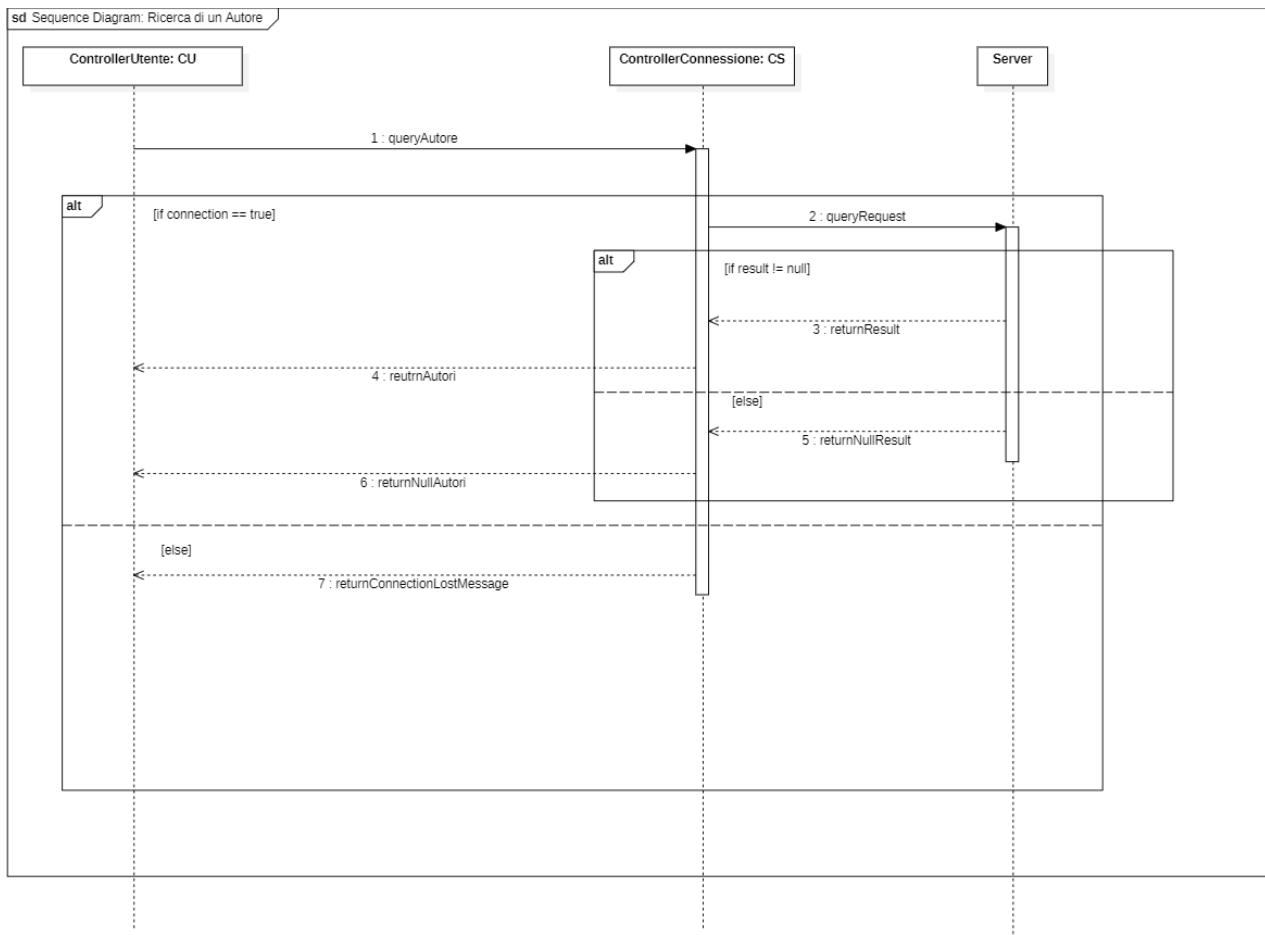


La tabella utente ha nuovi attributi: *password hashed*, che contiene la password hashata, *salt*, che servirà per comporre l'hash nel Database e *username*, che serve agli utenti per effettuare il login. Inoltre vengono rimossi gli attributi di *Data Inizio Validità* e *Data Fine Validità* poiché non venivano utilizzati all'interno dell'applicativo e non viene richiesto il suo scopo. Inoltre, alla tabella di associazione *Autore-Riferimento* vengono rimossi gli attributi *Ordine* e *Descrizione Utente*, il primo perché ininfluente ai fini dell'implementazione del Database, il secondo perché ridondante. Per il resto, l'intero sistema ci è sembrato pressoché pertinente allo sviluppo della nuova app reingegnerizzata.

## 2.7 Diagrammi di Sequenza

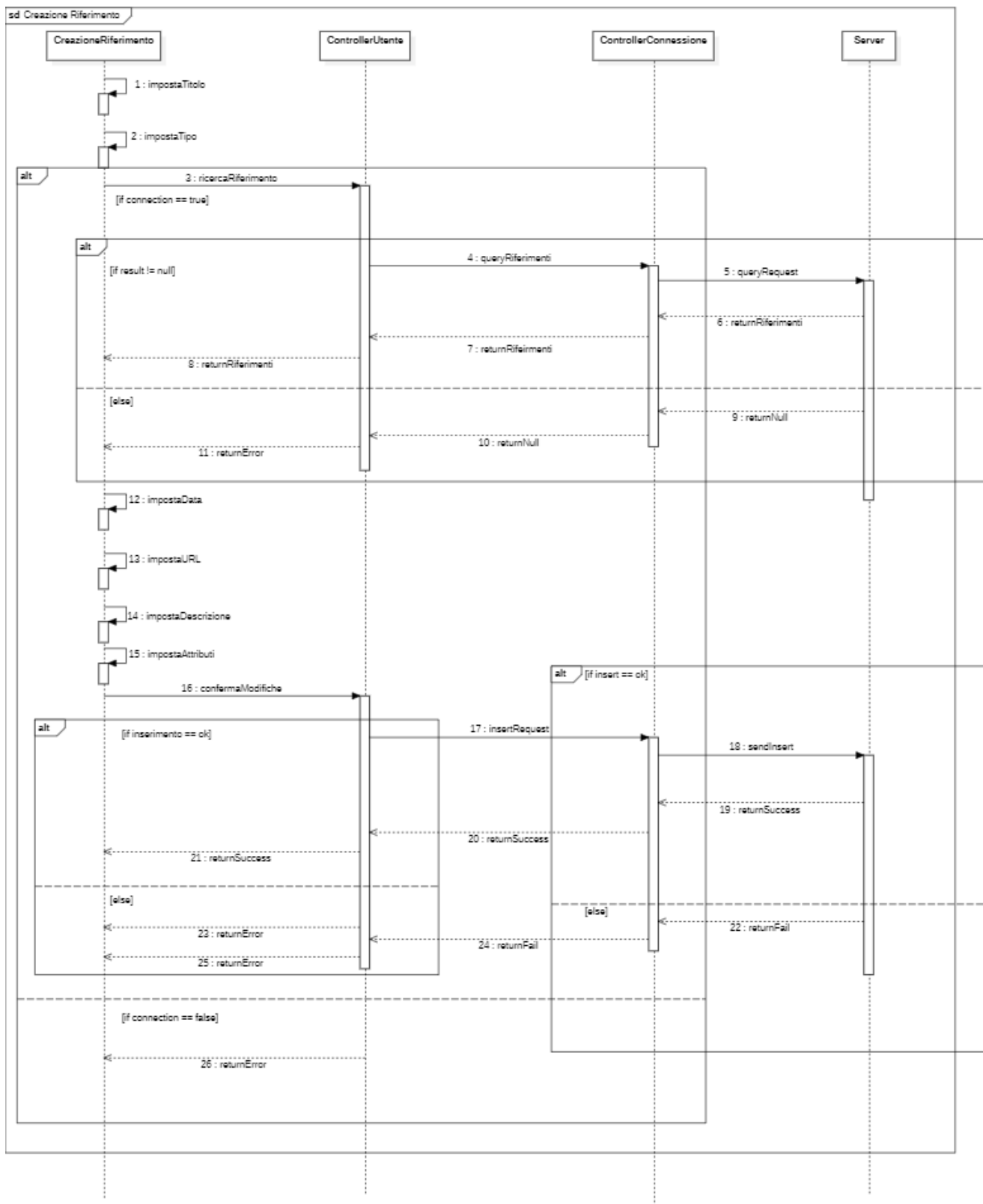
Di seguito vengono riportati i Sequence Diagram di design di due funzionalità: la ricerca di un autore e la creazione di un riferimento.

### 2.7.1 Diagramma di Sequenza: Ricerca di un Autore



Il sequence Diagram qui presente rappresenta l'istante in cui un utente preme su *Ricerca* una volta inseriti tutti i parametri preferiti, supponendo non abbia riscontrato problemi prima di premere invio.

## 2.7.2 Diagramma di Sequenza: Creazione Riferimento



Il sequence diagram di design qui presente rappresenta l'istante in cui un utente naviga nella finestra dedicata alla creazione di un nuovo riferimento. In particolare, sono stati inseriti tutti i casi di errore e successo.

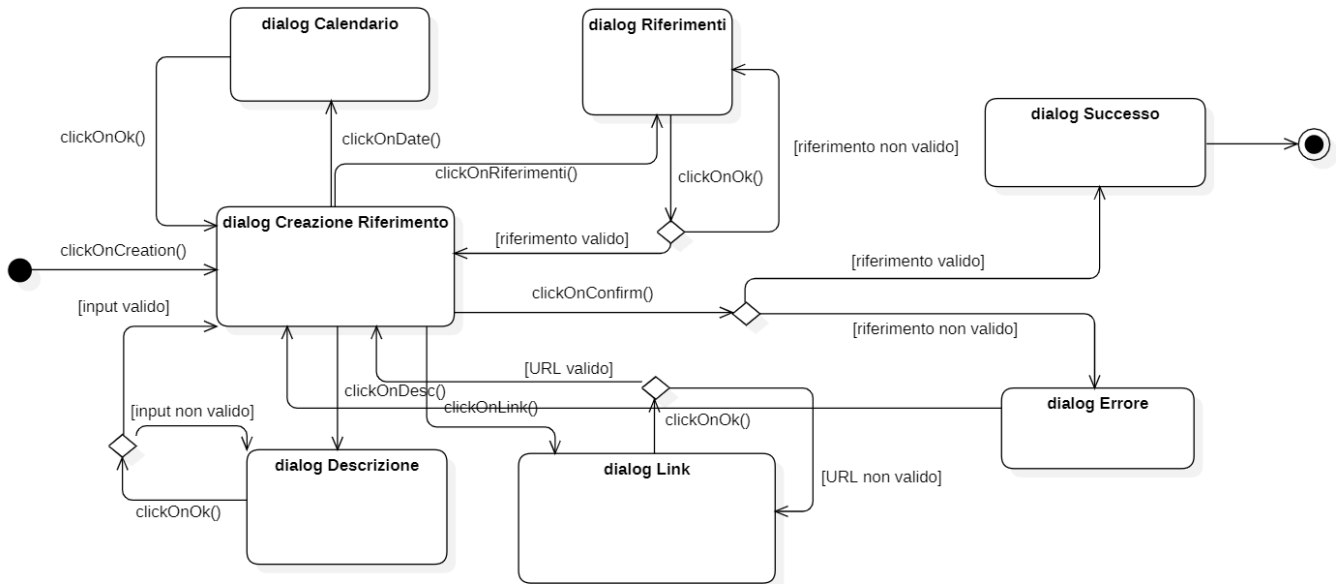


## 2.8 Prototipazione funzionale

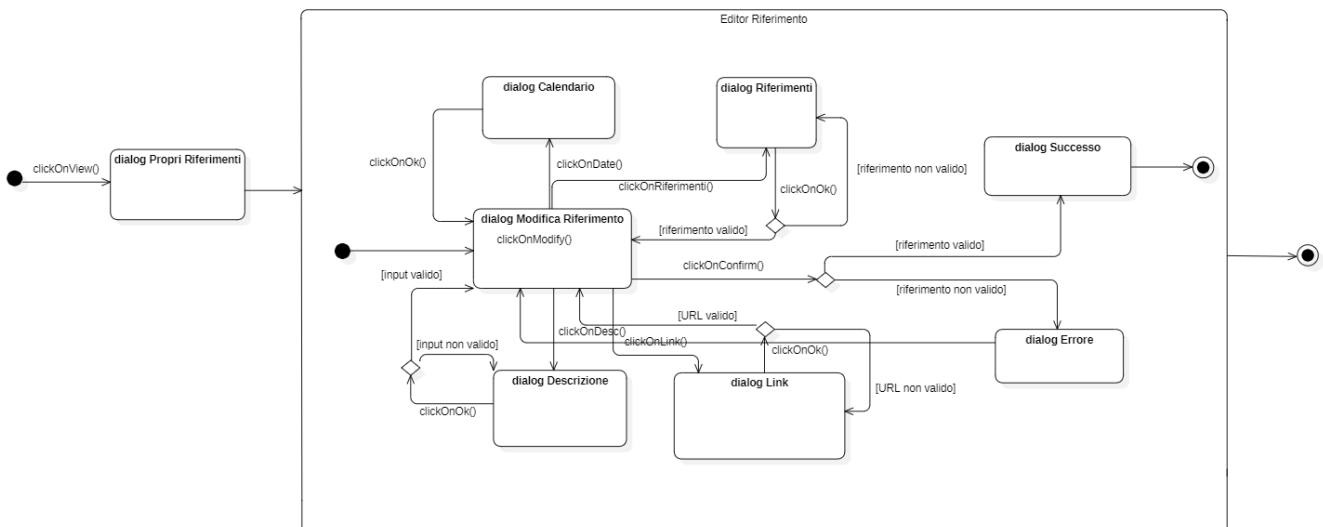
### 2.8.1 Statechart: Ricerca Riferimenti



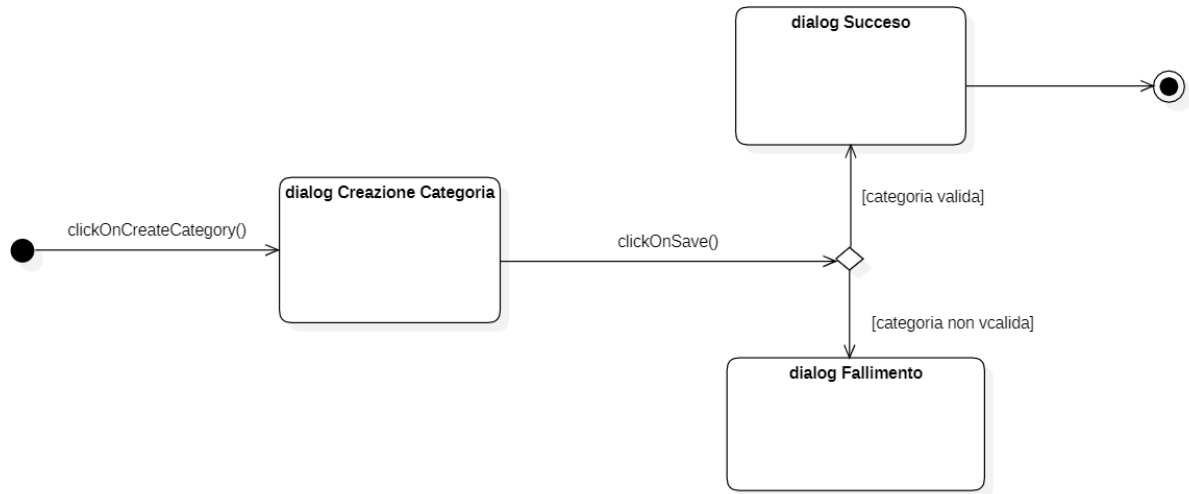
### 2.8.2 Statechart: Creazione Riferimento



### 2.8.3 Statechart: Modifica dei propri Riferimenti



## 2.8.4 Statechart: Creazione Categoria



# Capitolo 3

## Design del Sistema

### 3.1 Analisi dell'architettura e motivazioni

L'architettura del software precedente era strutturata in maniera tale che fosse monolitica: era composta da tre strati gerarchici, dove il più basso, la View, era il Layer deputato al Front-End, il layer centrale, il Model, era deputato alla gestione dei modelli richiesti dal sistema e infine il layer Database, che gestiva la manipolazione dei dati ed eventuale loro creazione, rimozione o modifica.

In Alexandria, nella fase di progettazione dell'architettura del sistema, il compito più arduo è stato quello di mantenere il Database e riutilizzare il codice sorgente ma ristrutturare l'architettura per renderla più flessibile e non monolitica, in particolare di adottare l'architettura REST. Il sistema infatti prevede più sottoinsiemi di layer che collettivamente compongono l'applicativo: il front end è composto da tre layer, uno per la comunicazione HTTP con il server, uno per l'interfaccia grafica e un'altra per la modellazione delle classi richieste. Il layer per la comunicazione a sua volta è composto da un layer aggiuntivo per la corretta gestione dei dati, utilizzato solamente dal front end nella trasmissione dei dati o per la richiesta di lettura. L'intero sottosistema del front end è inglobato nell'architettura REST che comprende anche il lato server. Quest'ultimo infatti rispecchia la rappresentazione di una architettura REST basata su HTTP, dove lo stato dell'applicazione e le funzionalità sono divisi in risorse, ogni risorsa è unica e indirizzabile usando sintassi universale per uso nei link ipertestuali. Tutte le risorse sono condivise come interfaccia uniforme per il trasferimento di stato tra front end e risorse, questo consiste in un insieme vincolato di operazioni ben definite, un insieme vincolato di contenuti, opzionalmente supportato da codice a richiesta. Abbiamo deciso di implementare un'architettura REST per la versatilità di richiedere e trasmettere le risorse al server, per la facilità del mapping del Database nelle entità e soprattutto perché perfetto per le nostre esigenze: il database durante la fase di progettazione già era implementato, ma non era completamente utilizzabile per via dell'architettura monolitica del precedente sistema. L'API Rest si è presentata come un'ottima scelta per poter implementare un nuovo sistema senza dover realizzare dal nulla lo stesso database.

### 3.2 Descrizione e motivazioni delle scelte tecnologiche adottate

Le tecnologie usate durante lo sviluppo del progetto sono essenzialmente tre: PostgreSQL per il Database Management System, Spring Boot per l'API Rest da implementare per la comunicazione con il DBMS e Flutter per lo sviluppo del front-end. Di seguito descrizione e motivazioni delle nostre scelte.

#### 3.2.1 PostgreSQL

PostgreSQL è un DBMS di tipo relazionale in cui è possibile inserire dati, modificarli, cercarli ed eliminarli. E' inoltre fornito di funzionalità aggiuntive come la creazione di funzioni tramite PLPGSQL o SQL Dinamico.

In Alexandria, il back-end è gestito in parte da PostgreSQL. Il motivo principale di questa scelta è la già nota esistenza di un Database apposito e le modifiche pensate durante la fase di progettazione non erano tali da dover reimplementare interamente il Database. Non avrebbe avuto senso dover reimplementare lo stesso database con le stesse funzioni per un nuovo DBMS, perdendo così tempo per la progettazione dell'applicativo, anche perché il Database risultava perfettamente funzionante durante la fase di testing del vecchio applicativo.

Il DBMS ci ha permesso di implementare funzioni apposite per determinate query, soprattutto per le relazioni ricorsive, tipi enumerativi e cast dedicati, funzionalità che magari non erano presenti in altri DBMS.

### 3.2.2 Spring Boot JPA

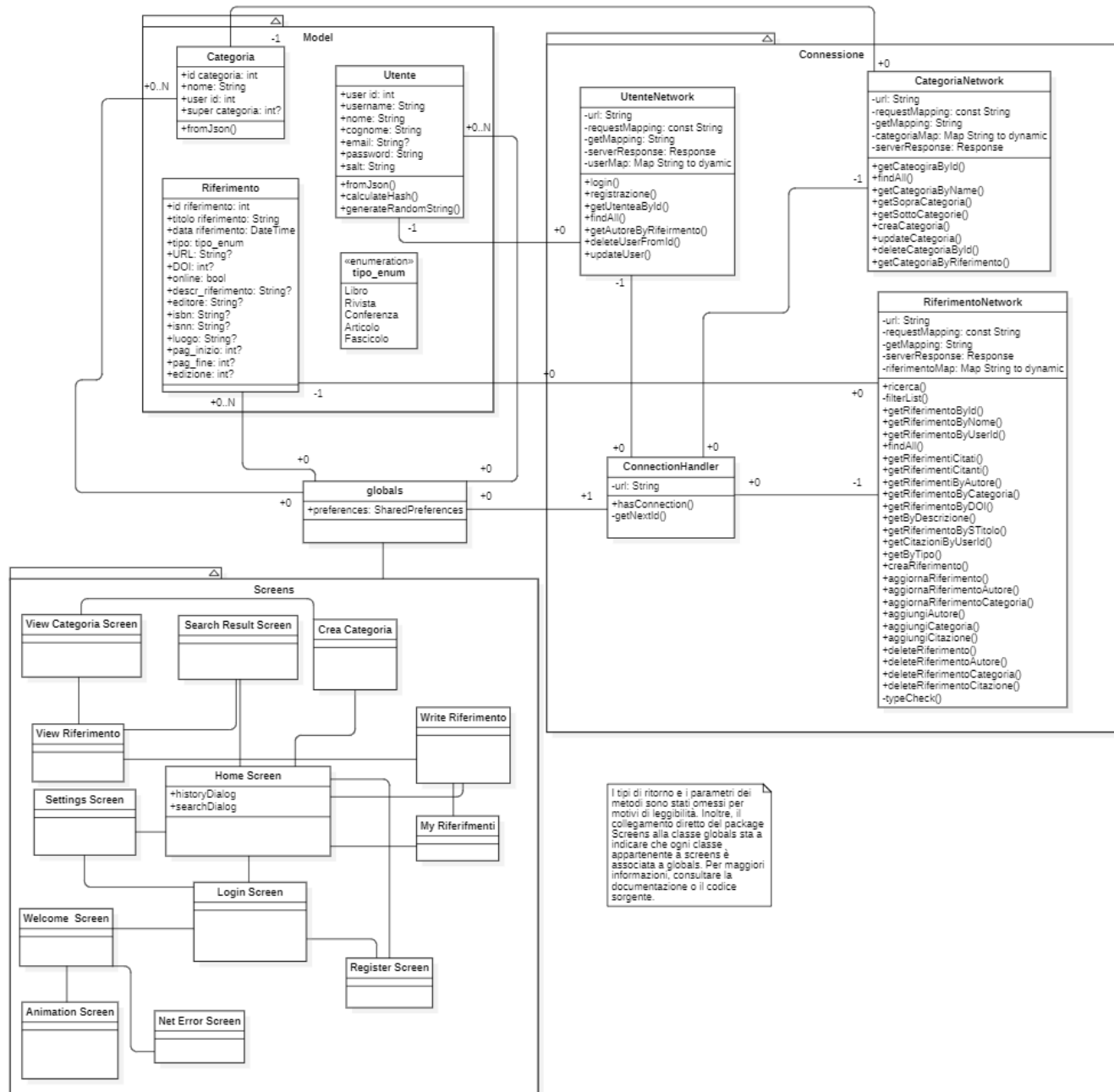
Spring Boot è un API Rest atto a implementare un applicativo capace di comunicare mediante architettura REST. In Alexandria, Spring Boot è stato uno strumento fondamentale per l'implementazione e la gestione delle risorse dal client al DBMS. Prima dell'implementazione effettiva, abbiamo avuto dinanzi una vaste alternative di framework per l'implementazione di tale architettura. La scelta finale è però ricaduta su Spring Boot principalmente per due motivi: dato che l'applicativo originale era stato scritto in Java, ci è risultato più conveniente implementare Spring Boot su un eseguibile Java essendo possibile poter recuperare il codice sorgente originale e modificarlo all'occorrenza. Inoltre, Spring Boot offre la possibilità di effettuare query native, oltre a quelle di supporto: in tal modo abbiamo potuto riportare uno a uno le query già implementate nel vecchio sistema, senza doverle reimplementare. L'unico sforzo effettivo è stato quello di adattare le funzioni di query per ogni caso.

### 3.2.3 Flutter

Flutter è un framework open source per la creazione delle interfacce native per diversi sistemi operativi, tra cui Android. Una delle motivazioni principali della scelta dell'utilizzo di Flutter è la sua enorme semplicità nella creazione e modifica delle interfacce grafiche. Inoltre, è fornito nativamente il supporto per la maggior parte dei sistemi operativi in uso, così, se si dovesse esportare Alexandria su nuove piattaforme, basterà solamente compilare il codice sorgente e distribuire l'applicativo. In caso di modifiche alle schermate in base alle esigenze del dispositivo che lo richiede, basterà modificare i metodi appositi, senza dover ricorrere ad artifici particolari. Infine, essendo un linguaggio particolarmente simile a Java, è stato semplice dover esportare il codice originale in Flutter, modificando però opportunamente il codice in caso di incompatibilità.

### 3.3 Diagramma delle classi di design

Alla luce di quanto è stato detto, viene riportato il seguente diagramma delle classi di Design del front-end:

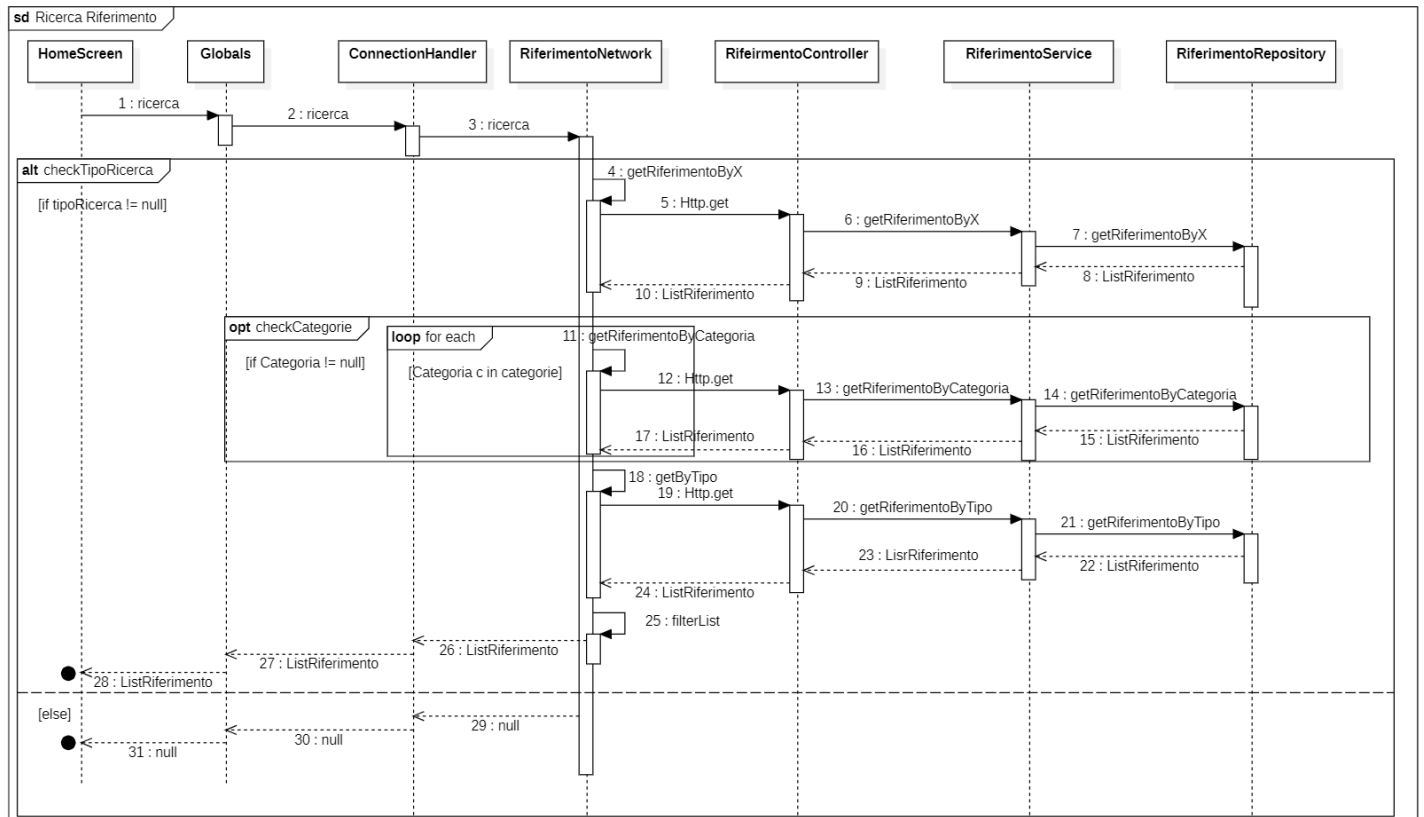


L'applicativo consta in tre package principali, uno per gli Screen (GUI), uno per la connettività e infine uno per rappresentare i Model. Inoltre sono presenti classi globali come appunto *globals* che fungono da intermediario fra i package o per l'avvio dell'applicativo. In questo diagramma sono stati rappresentati in dettaglio le relazioni fra le classi e gli attributi, ma sono stati omessi i tipi di ritorno e i parametri dei metodi in maniera tale da rendere l'intero diagramma più leggibile. Se si volesse consultare in dettaglio ogni metodo utilizzato, si consiglia di leggere la sezione apposita. Inoltre, sono stati omessi dal diagramma anche le classi che implementano widget o componenti particolari per l'interfaccia grafica: essendo solamente classi di supporto, abbiamo considerato ridondante la loro presenza nel Class Diagram soprastante, anche perché le classi contenute nel package Screen sono autoesplicative e fanno intendere il loro funzionamento anche senza specificare le componenti che sfruttano per la loro corretta esecuzione. Infine, sono stati omessi gli attributi degli screen e i loro metodi essendo la maggior parte attributi nativi e solamente tecnici che non contribuiscono alla comprensione della visione complessiva del sistema.

## 3.4 Diagrammi di sequenza di design

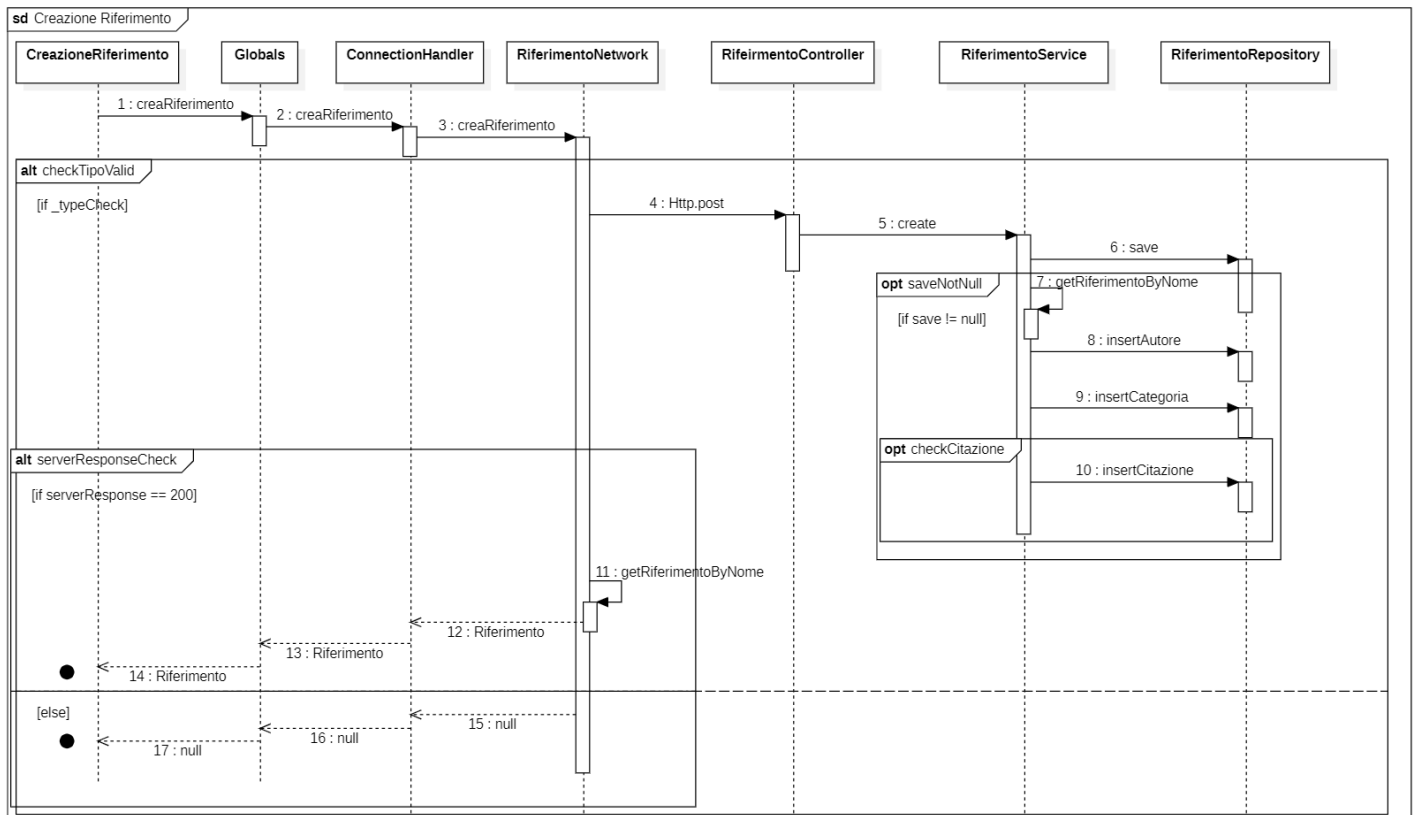
Di seguito vengono riportati i Sequence Diagram per i casi d'uso identificato nella sezione 2.3.

### 3.4.1 Ricerca Riferimenti



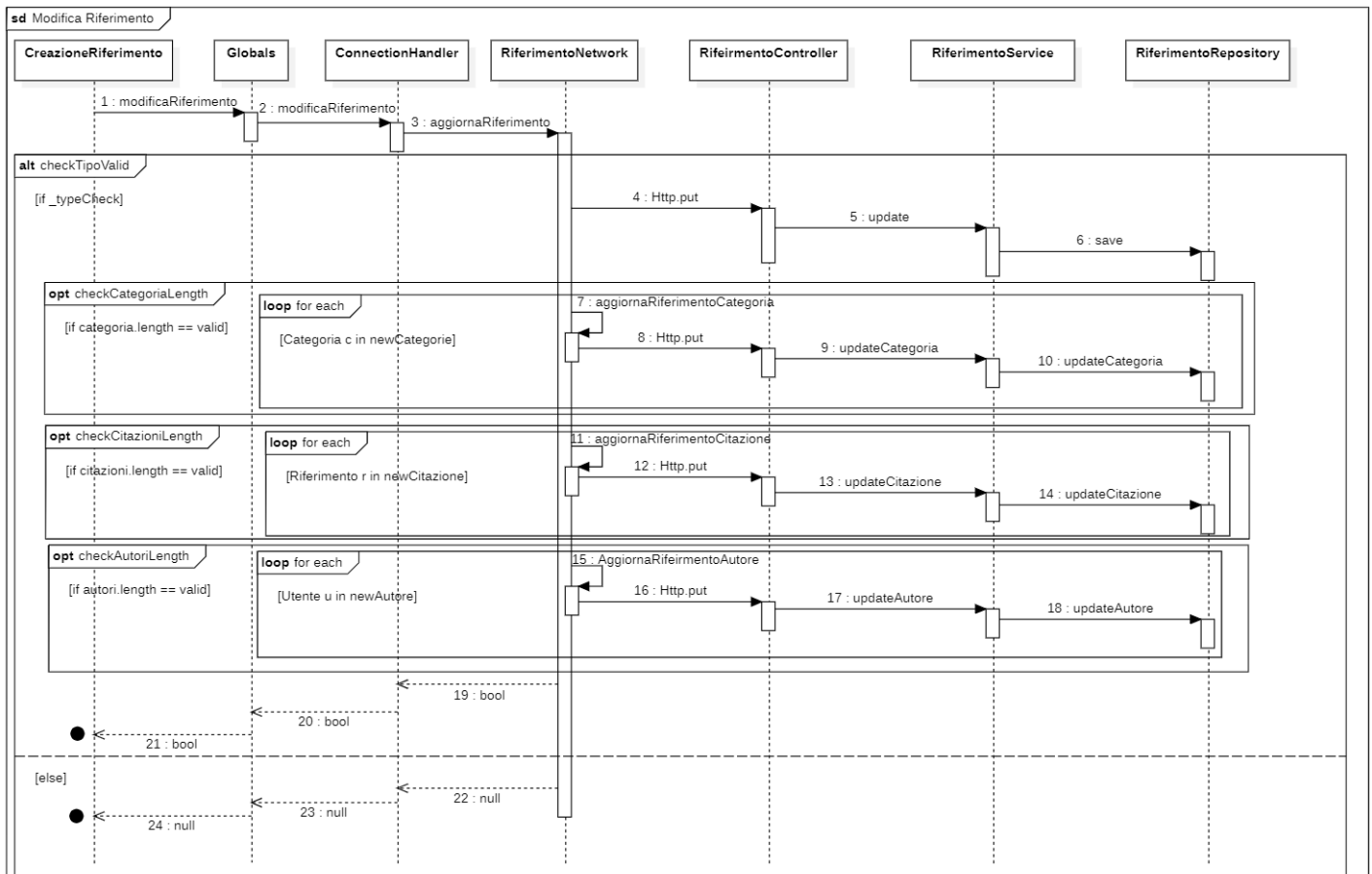
Il seguente Sequence Diagram mostra le entità coinvolte durante la fase di ricerca. Da notare che sono state incluse entrambi i sottosistemi front-end e back-end. Nota: il metodo *getRiferimentoByX* sta a indicare diversi metodi che hanno lo stesso nome ma che si differenziano per X, ovvero il tipo di ricerca che viene effettuata (per Titolo, per Autore o per DOI).

### 3.4.2 Creazione Riferimento



Il seguente Sequence Diagram mostra le classi coinvolte durante la fase di creazione di un riferimento. Il sequence assume un comportamento diverso in base al metodo `typeCheck`: se l'utente inserisce valori non validi per un determinato tipo, allora l'operazione di `post` non viene effettuata.

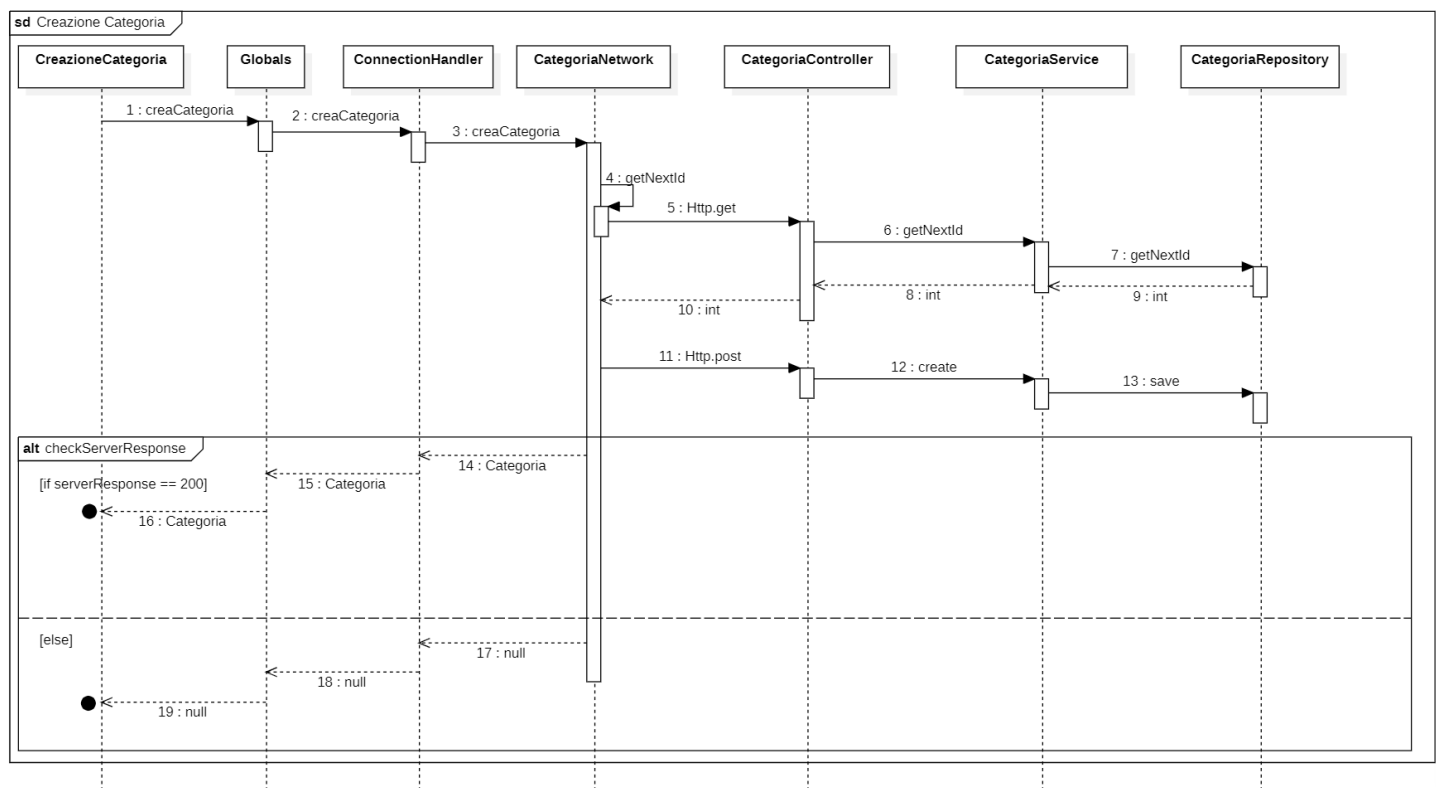
### 3.4.3 Modifica Riferimento



Il seguente Sequence Diagram mostra le classi coinvolte e i metodi coinvolti durante la fase di modifica di un riferimento. Anche in questo caso, si può notare un comportamento differente in base al check basato sui valori inseriti dall'utente. Inoltre, viene restituito un AND fra gli i vari valori modificati: se un valore, per esempio Autore, viene modificato, si effettua un controllo effettivo sulla modifica e viene restituito un AND con il riferimento modificato che restituirà il risultato finale.



### 3.4.4 Creazione Categoria



Il seguente Sequence Diagram mostra le classi coinvolte durante la fase di creazione di una categoria. Non è un'operazione complessa data la natura semplice della categoria stessa, quindi non vengono effettuati particolari check sugli attributi.

## 3.5 Codice sorgente e Dockerfile

E' possibile visionare e consultare l'intero codice sorgente, dockerfile e file vari presso la seguente repository In particolare, è possibile consultare il codice sorgente dell'applicativo del front-end in questa subdirectory, il codice sorgente del back-end in questa subdirectory e dockerfile in questa subdirectory. Il testo è cliccabile.

Abbiamo cercato di rendere l'applicativo più modulare possibile in maniera tale che le modifiche da apportare in futuro, se necessario, saranno più facili da effettuare e per poter integrare nuove funzionalità con semplicità nel codice. Inoltre sono presenti nella nostra API funzioni non utilizzate per offrire supporto in futuro, se necessario.

## Capitolo 4

# Testing e valutazione sul campo dell'usabilità

### 4.1 Codice xUnit

Di seguito viene riportato il codice dartUnit per 4 metodi non banali che abbiano almeno due parametri.

#### 4.1.1 Unit Testing: Creazione Categoria

```
1 import 'package:test/test.dart';
2
3 void main() {
4   group('Test categorie', () {
5
6     test('Creazione categoria', () async {
7
8       expect(await networkHelper.createCategoria('', 1, null), null); //CE1 CE4 CE5
9       expect(await networkHelper.createCategoria('WECT 1', -100, null), null); //CE2 CE3 CE5
10      expect(await networkHelper.createCategoria('WECT 2', 10, -5), null); //CE2 CE4 CE6
11      expect(await networkHelper.createCategoria('WECT 3', 10, null) != null, true); //CE2 CE4 CE5
12      expect(await networkHelper.createCategoria('WECT 4', 10, 20) != null, true); //CE2 CE4 CE7
13
14    });
15
16    test('Test sopra-categoria', () async {
17
18      /*Codice Omesso*/
19
20    });
21
22    test('Test eliminazione categoria', () async {
23
24      /*Codice Omesso*/
25
26    });
27  });
28 }
```

## 4.1.2 Unit Testing: Ricerca di un Riferimento

```
1 import 'package:test/test.dart';
2
3 void main() {
4   group('Test riferimenti', () {
5
6     test('Test Ricerca Riferimento', () async {
7
8       List<Categoria> categoriaPiena = [(await networkHelper.getCategoriaById(1))!];
9       List<tipo_enum> tipoPiena = [tipo_enum.Libro];
10
11       expect(await networkHelper.ricerca('', 10, 'autore', categoriaPiena, tipoPiena), []); //CE2 CE5 CE7 CE9
12
13       expect(await networkHelper.ricerca(null, 10, 'autore', categoriaPiena, []), []); //CE1 CE5 CE7 CE8
14
15       expect(await networkHelper.ricerca(null, 10, 'autore', categoriaPiena, tipoPiena), []); //CE1 CE5 CE7 CE9
16
17       expect(await networkHelper.ricerca('Ricerca bug', null, null, [], tipoPiena) != [], true); //CE3 CE4 CE6 CE9
18
19       for (var i = 0; i < 6; i++) {
20         tipoPiena.add(tipo_enum.Libro);
21       }
22
23       expect(await networkHelper.ricerca(null, 10, 'autore', categoriaPiena, tipoPiena), []); //CE1 CE5 CE7 CE10
24
25     });
26
27     test('Test Creazione riferimento', () async {
28       /*Codice Omesso*/
29     });
30
31     test('Test eliminazione riferimento', () async {
32       /*Codice Omesso*/
33     });
34     test('Test Aggiungere Citazione', () async {
35       /*Codice Omesso*/
36     });
37
38     test('Test aggiorna riferimento autore', () async {
39       /*Codice Omesso*/
40     });
41   });
42 }
```

### 4.1.3 Unit Testing: Calcolo hash

```
1 import 'package:test/test.dart';
2
3 void main() {
4   group('Test utenti', () {
5
6     test('Test calcolo hash', () async {
7       Utente u = Utente(-1, "Test", "nome", "cognome", "email", "password");
8       expect(u.calculateHash("", "salt"), ""); //CE1 CE4
9       expect(u.calculateHash("password", "", "")); //CE2 CE3
10      expect(u.calculateHash("unhashedPassword", "salt") != null, true); //CE2 CE4
11    });
12  });
13
14 }
```

### 4.1.4 Unit Testing: Registrazione

```
1 import 'package:test/test.dart';
2
3 void main() {
4   group('Test Utenti', () {
5
6     test('Test Registrazione', () async {
7
8       expect(await networkHelper.registrazione("", "password", "nome", "cognome", "email"), null); //CE1
9       CE4 CE6 CE8 CE10
10      expect(await networkHelper.registrazione("username", "", "nome", "cognome", "email"), null); //CE2
11      CE3 CE6 CE8 CE10
12      expect(await networkHelper.registrazione("username", "password", "", "cognome", "email"), null); //
13      CE2 CE4 CE5 CE8 CE10
14      expect(await networkHelper.registrazione("username", "password", "nome", "", "email"), null); //CE2
15      CE4 CE5 CE7 CE10
16      expect(await networkHelper.registrazione("username", "password", "nome", "cognome", ""), null); //
17      CE2 CE4 CE5 CE8 CE9
18      expect(await networkHelper.registrazione("username", "password", "nome", "cognome", "email") != null
19        , true); //CE2 CE4 CE6 CE8 CE10
20    });
21  });
22
23 }
```

### 4.1.5 Strategie adottate per la progettazione dei test

Tali unità di test sono state implementate mediante un approccio Black Box, anche se il codice sorgente era ovviamente disponibile. Questo perché abbiamo preferito testare il comportamento esterno dell'applicativo, senza doverci focalizzare sulle sfaccettature del codice.

Abbiamo deciso di presentare quattro unità di test tra i tanti metodi che sarebbero stati maggiormente utilizzati in Alexandria: un metodo per la creazione di una categoria, uno per la ricerca di un riferimento, uno per la creazione di un riferimento e infine uno per l'aggiunta di una citazione. In particolare, sono stati individuate tali classi d'equivalenza:

<b>creazioneCategoria</b>			
String nome	CE1: {""} n. v.	CE2: {"abc"} val.	
int user_id	CE3: {minInt, -1} n. v.	CE4: {0, maxInt} val.	
int? superCategoria	CE5: {null} val.	CE6: {minInt, -1} n. v.	CE7: {0, maxInt} val.

<b>ricercaRiferimento</b>			
String? titolo	CE1: {null} val.	CE2: {""} n. v.	CE3: {"abc"} val.
int? doi	CE4: {null} val.	CE5: {minInt, maxInt} val.	
List<Categoria>c	CE6: {[}] val.	CE7: {[1, .., n]} val.	
List<tipo_enum>t	CE8: {[}] n. v.	CE9: {[1, .., 5]} val.	CE10: {[6, .., n]} n. v.

<b>registrazioneUtente</b>		
String username	CE1: {""} n. v.	CE2: {"abc"} val.
String unhashedPassword	CE3: {""} n. v.	CE4: {"abc"} val.
String nome	CE5: {""} n. v.	CE6: {"abc"} val.
String cognome	CE7: {""} n. v.	CE8: {"abc"} val.
String email	CE9: {""} n. v.	CE10: {"abc"} val.

<b>calculateHash</b>		
String unhashedPassword	CE1: {""} n. v.	CE2: {"abc"} val.
String salt	CE3: {""} n. v.	CE4: {"abc"} val.

Il criterio di copertura utilizzato è la *Weak Equivalence Class Testing*. Le caratteristiche individuate per ogni classe d'equivalenza rispecchiano i casi limite dei possibili valori che un parametro della funzione possa assumere.

## 4.2 Valutazione dell'usabilità sul campo

## Capitolo 5

# Conclusione

### 5.1 Note finali