

Programmazione Mobile

Lorenzo Gaetani 1099077

September 8, 2023

1 Android

1.1 Descrizione

L'obiettivo del progetto consiste nella realizzazione di un'applicazione per la gestione della biblioteca dell'università. Coloro che possono accedere all'applicazione sono gli utenti e l'amministratore. L'applicazione non gestisce la procedura di registrazione in quanto, gli utenti che possono accedervi, sono gli studenti iscritti all'università. Gli utenti possono consultare la lista dei libri disponibili e selezionare il libro che desiderano prendere in prestito. Ogni utente può prendere in prestito al massimo 3 libri alla volta. Ogni prestito ha una durata di 21 giorni. Gli utenti possono scrivere recensioni e valutare i libri, anche senza averli presi in prestito. Inoltre, gli utenti posso prenotare un posto nella biblioteca. La prenotazione di un posto vale per tutta la giornata, non ci sono fasce orarie. Per restituire il libro preso in prestito, l'utente deve recarsi nella biblioteca e restituirlo; sarà compito dell'amministratore registrare l'avvenuta restituzione.

1.2 Requisiti

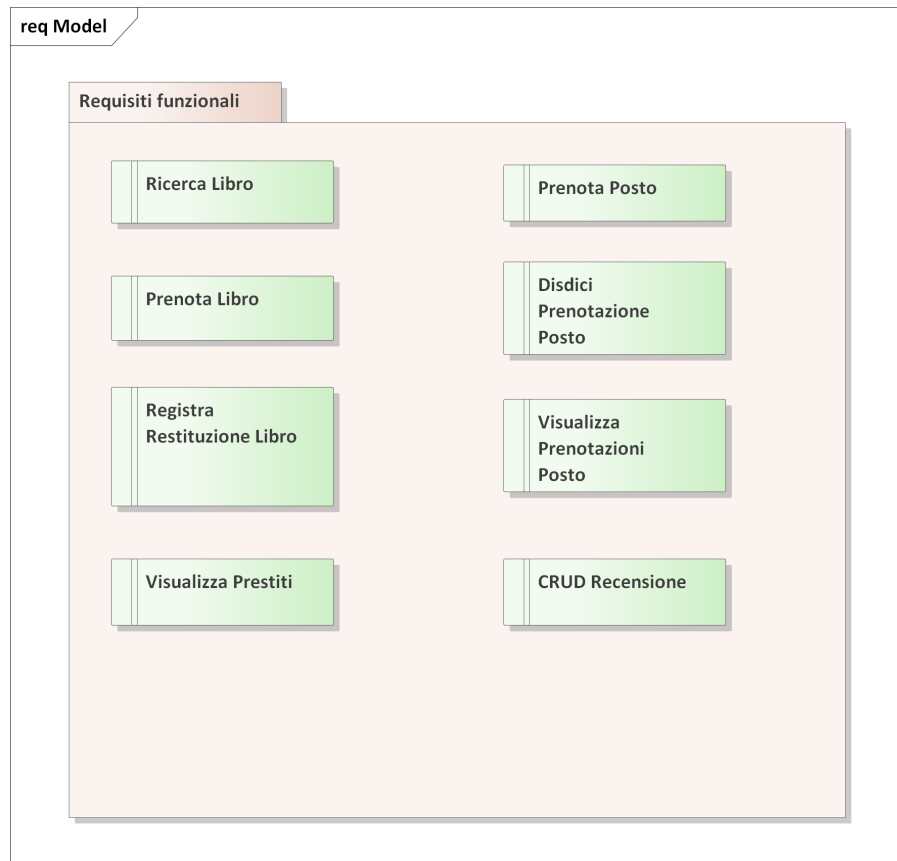


Figure 1: Requisiti

1.3 Casi d'uso

1.3.1 Attori

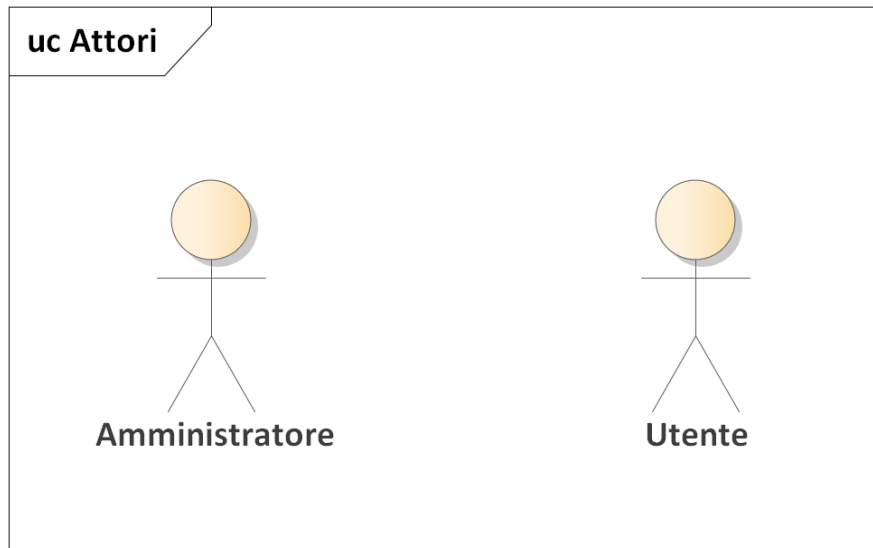


Figure 2: Attori

1.3.2 Diagrammi dei casi d'uso

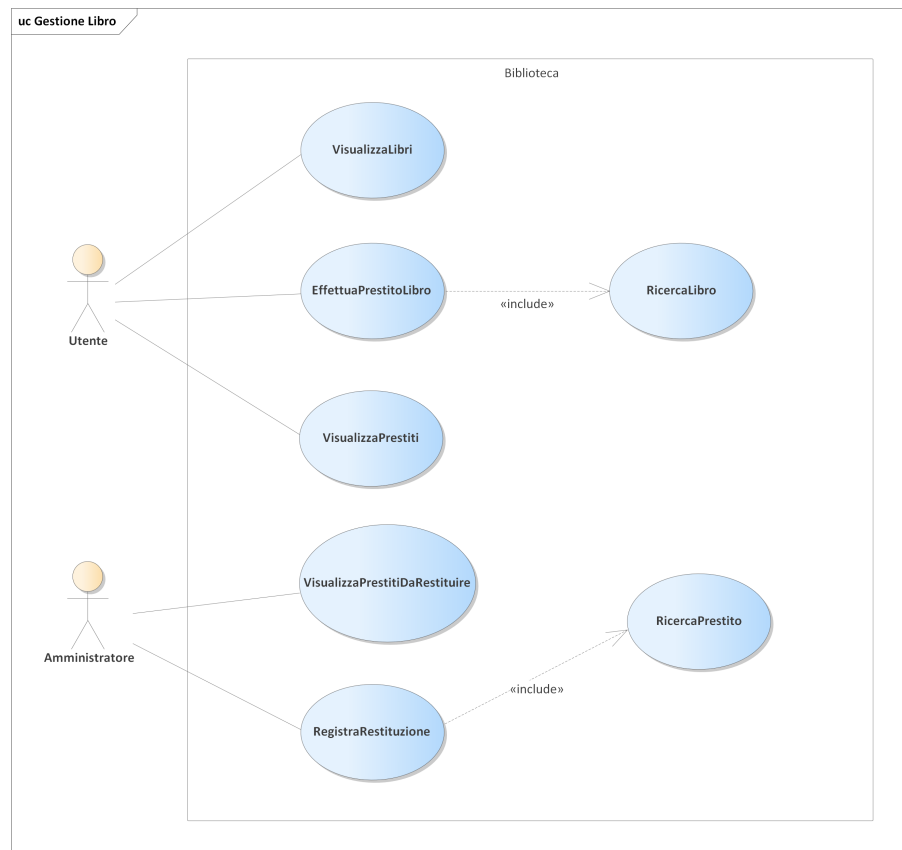


Figure 3: GestioneLibro

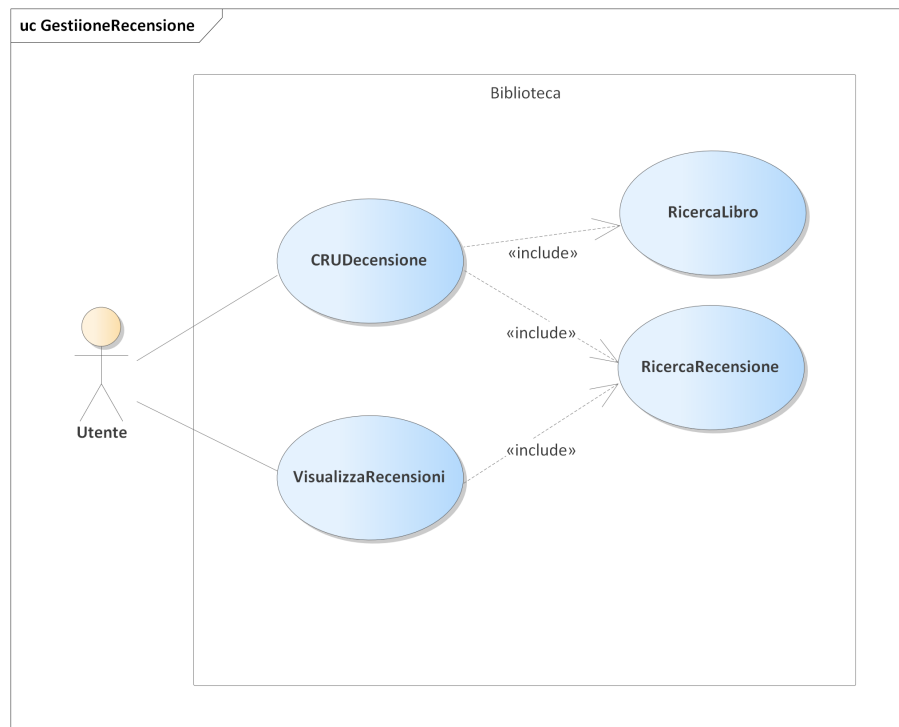


Figure 4: GestioneRecensione

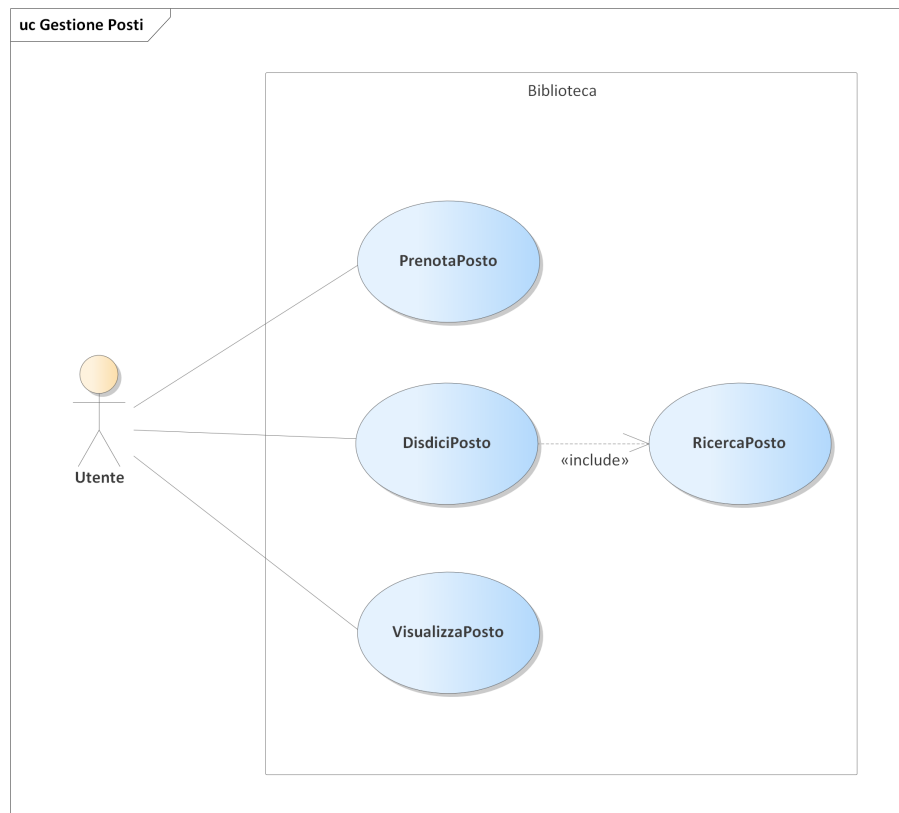


Figure 5: GestionePosti

1.4 Descrizione dei casi d'uso

Titolo: CUDRecensione

Descrizione: il caso d'uso permette di inserire, modificare e eliminare una recensione.

Attori primari: Utente

Precondizioni: per modificare o eliminare una recensione, l'utente deve averla creata

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole inserire o modificare o eliminare una recensione.
 2. if l'attore primario vuole inserire una recensione:
 - (a) include(RicercaLibro)
 - (b) l'attore primario inserisce i dati relativi alla recensione
 - (c) il sistema memorizza la recensione.
 3. if l'attore primario vuole modificare una recensione:
 - (a) include(RicercaRecensione)
 - (b) l'attore primario modifica i dati relativi alla recensione
 - (c) il sistema memorizza le modifiche effettuate.
 4. if l'attore primario vuole eliminare una recensione:
 - (a) include(RicercaRecensione)
 - (b) il sistema elimina la recensione.
-

Titolo: VisualizzaRecensioni

Descrizione: il caso d'uso permette di visualizzare tutte le recensioni.

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare tutte le recensioni
 2. for each recensioni:
 - (a) il sistema visualizza a schermo i dati relativi alla recensione.
-

Titolo: RicercaRecensioni

Descrizione: il caso d'uso permette di ricercare una recensione.

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole ricercare una recensione
2. l'attore primario inserisce i dati chiave per la ricerca
3. il sistema ritorna i dati relativi alla recensione

Titolo: EffettuaPresitoLibro

Descrizione: il caso d'uso permette all'utente di prendere in prestito un libro

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole prendere in prestito un libro
2. include (RicercaLibro)
3. if l'utente può prendere in prestito il libro:
 - (a) il sistema memorizza il prestito
 - (b) il sistema decrementa la quantità del libro
4. else il sistema mostra un messaggio di errore.

Titolo: RegistraRestituzione

Descrizione: il caso d'uso permette di registrare la restituzione di un libro

Attori primari: Amministratore

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole registrare l'avvenuta restituzione di un libro
2. include (RicercaPrestito)
3. il sistema memorizza l'avvenuta restituzione del libro
4. il sistema incrementa la quantità del libro.

Titolo: RicercaLibro

Descrizione: il caso d'uso permette di ricercare un libro.

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole ricercare un libro
2. l'attore primario inserisce i dati chiave per la ricerca

3. il sistema ritorna i dati relativi al libro.

Titolo: VisualizzaLibri

Descrizione: il caso d'uso permette di visualizzare i libri.

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare i libri item for each libri:
 - (a) il sistema visualizza a schermo i dati relativi ai libri.

Titolo: VisualizzaPrestiti

Descrizione: il caso d'uso permette di visualizzare i prestiti dell'utente.

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare la lista dei prestiti
2. for each prestiti dell'utente:
 - (a) il sistema visualizza a schermo i dati relativi al prestito.

Titolo: VisualizzaPrestitiDaRestituire

Descrizione: il caso d'uso permette di visualizzare i prestiti ancora da restituire.

Attori primari: Amministratore

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare la lista dei prestiti da restituire
2. for each prestiti dell'utente:
 - (a) il sistema visualizza a schermo i dati relativi al prestito.

Titolo: RicercaPrestito

Descrizione: il caso d'uso permette di ricercare un prestito.

Attori primari: Amministratore

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole ricercare un
2. l'attore primario inserisce i dati chiave per la ricerca

3. il sistema ritorna i dati relativi al libro.

Titolo: PrenotaPosto

Descrizione: il caso d'uso permette di prenotare un posto nella biblioteca

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole prenotare un posto
2. if l'utente può prenotare il posto:
 - (a) il sistema memorizza la prenotazione
3. else il sistema mostra un messaggio di errore.

Titolo: DisdiciPosto

Descrizione: il caso d'uso permette di disdire la prenotazione di un posto nella biblioteca

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole disdire la prenotazione di un posto
2. include (RicercaPosto)
3. il sistema memorizza la disdetta.

Titolo: VisualizzaPosto

Descrizione: il caso d'uso permette di visualizzare la lista delle prenotazioni dei posti

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare le prenotazioni dei posti
2. foreach prenotazioni non scadute fatte dall'utente:
 - (a) visualizza dati prenotazione.

Titolo: RicercaPosto

Descrizione: il caso d'uso permette di ricercare

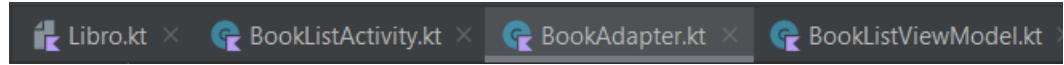
Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole ricercare la prenotazione di un posto
 2. l'attore primario inserisce i valori chiave per la ricerca
 3. il sistema ritorna i dati relativi alla prenotazione del posto
-

1.5 Architettura

Per lo sviluppo della applicazione, il pattern utilizzato è il Model-View-ViewModel (MVVM). Inoltre, il pattern LiveData, è usato per la gestione dei dati.



1.6 Database

I dati sono memorizzati in Cloud Firestore.

Gli utenti di prova sono [(email = S003@s.it, password = utente),(email = S002@s.it, password = utente2)]

Amministratore (email = S001@s.it, password = adminn)

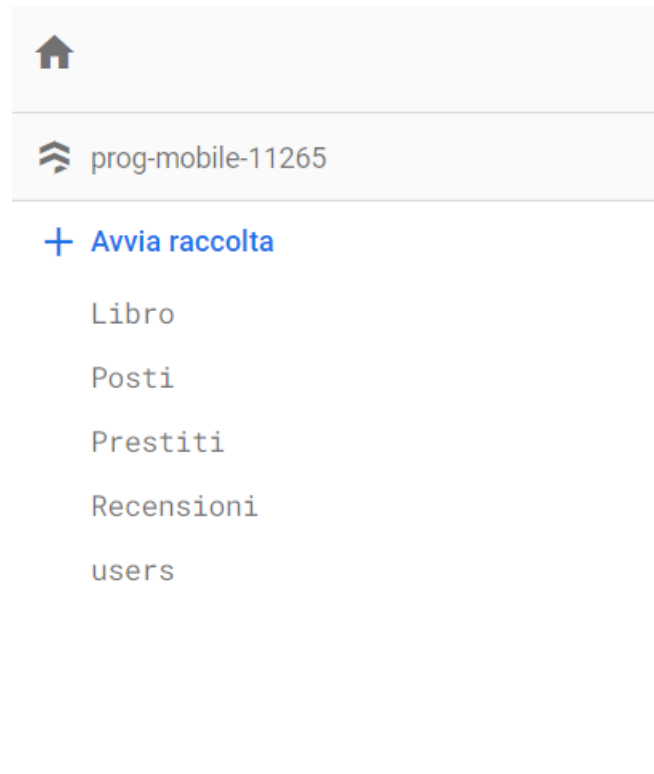


Figure 6: Cloud Firestore

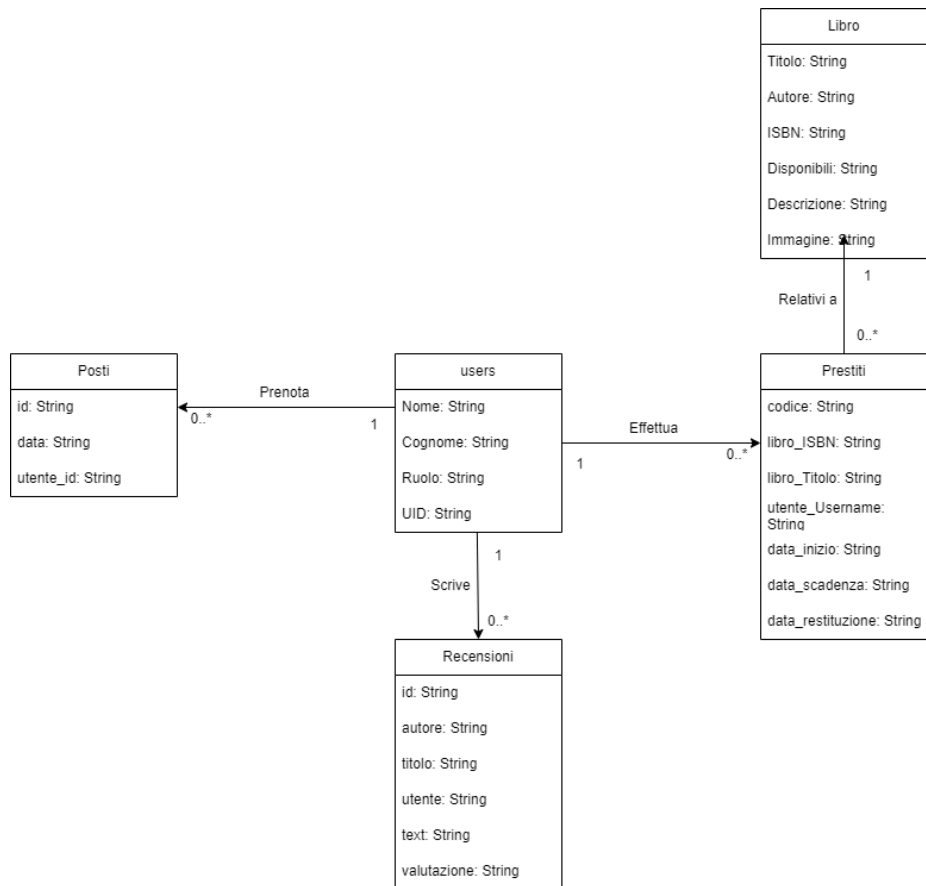


Figure 7: Tabelle del database

1.7 Sviluppo

La prima schermata dell'applicazione è quella per effettuare il login. La procedura di autenticazione è stata gestita tramite Firebase Authentication. A seconda del ruolo, l'utente è indirizzato o nella home dell'admin o in quella dell'utente. La struttura delle varie schermate è simile: una toolbar in alto con un navigation drawer, per navigare tra le schermate.

Per quanto riguarda la home dell'utente, la prima schermata è quella del catalogo dei libri disponibili. Nel caso in cui l'utente abbia dei prestiti scaduti, comparirà un popup, ogni volta che va nel catalogo, per avvertirlo. I libri sono visualizzati come una lista di card all'interno di una recycler view. In alto, si trova una search view che filtra i libri del catalogo. I parametri per la ricerca sono titolo, autore, isbn e descrizione.

Cliccando su una card, l'utente può vedere ulteriori dettagli relativi al libro. Inoltre, può prenderlo in prestito, se c'è più di una copia disponibile, il libro

non è già stato preso in prestito dall'utente e l'utente non ha superato il numero massimo di prestiti (3). Il prestito inizia nel momento in cui l'utente conferma di volere prendere in prestito il libro e la durata è di 21 giorni. In questo caso, è stato usato un `ViewModelProvider.Factory` personalizzato per creare un'istanza del `viewmodel` e passargli dei dati. Oltre al bottone per prendere in prestito, c'è un'altro per scrivere una recensione relativa a quel libro.

Nella sezione 'lista prestiti', l'utente può vedere una lista dei suoi prestiti.

Nella sezione 'recensioni', l'utente può vedere tutte le recensioni dei libri e filtrarle per titolo del libro. In basso, c'è una check box per visualizzare solo le recensioni scritte dall'utente. Cliccando sulla card della recensione, si può far comparire e scomparire la recensione. Facendo un click prolungato sulla card, compariranno le icone per eliminare o modificare una recensione (se scritta dall'utente).

Nella sezione 'prenota un posto', l'utente può visualizzare le prenotazioni dei posti effettuate e non scadute. La prenotazione di un posto vale per tutto il giorno. In basso, c'è il bottone per prenotare un posto nella biblioteca. Per effettuare la prenotazione, l'utente non deve avere già una prenotazione per lo stesso giorno e ci devono essere abbastanza posti disponibili per quel giorno (3).

Per quanto riguarda, invece, la home dell'admin, la struttura è simile a quella dell'utente, ma l'unica funzionalità riguarda la registrazione della restituzione di un libro.

Alcuni problemi sono stati riscontrati nella fase di testing poichè, dato che Firebase dipende da un contesto Android reale, che non è disponibile durante un test unitario, sono stati usati dei mock per le sue funzionalità e i test non sono molto significativi.


Un'altro problema riscontrato, è stata l'attesa per effettuare il login nell'applicazione. Per cercare di risolverlo è stata usata una coroutine che non interrompesse il thread principale.

```
CoroutineScope(Dispatchers.Main).launch { this: CoroutineScope
    try {
        val authResult = withContext(Dispatchers.IO) { this: CoroutineScope
            FirebaseAuth.getInstance().signInWithEmailAndPassword(username, password).await()
        }

        val id = FirebaseAuth.getInstance().currentUser?.uid.toString()
        UserManager.checkUserRole(id, role: "utente") { isUtente ->
            val intent = if (isUtente) {
                Intent( packageContext: this@MainActivity, HomeUtente::class.java)
            } else {
                Intent( packageContext: this@MainActivity, HomeAdmin::class.java)
            }
            startActivity(intent)
        }
    }
} catch (e: Exception) {
```

1.8 Mockup

Login



Login

Catalogo



Titolo Pagina



Card Libro



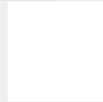
Titolo:
Autore:

Card Libro



Titolo:
Autore:

Card Libro



Titolo:
Autore:

Home utente

Navigation drawer



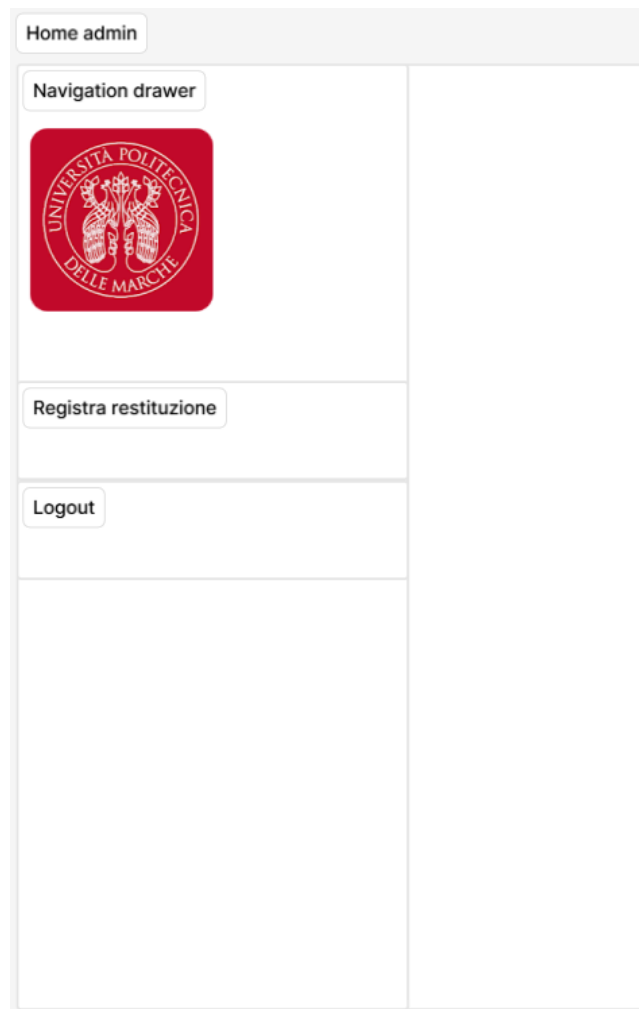
Catalogo

Prestiti

Scrivi una recensione

Prenota posto

Logout



1.9 Test

```
gray1 class AuthRepositoryTest {
gray2
gray3     @Mock
gray4     private lateinit var mockFirebaseAuth:
        FirebaseAuth
gray5
gray6     @Mock
gray7     private lateinit var mockAuthResultTask: Task<
        AuthResult>
gray8
```

```

gray9     private lateinit var authRepository:
           AuthRepository

gray10
gray11     @Before
gray12     fun setUp() {
gray13         MockitoAnnotations.initMocks(this)
gray14         authRepository = AuthRepository(
           mockFirebaseAuth)
gray15     }
gray16
gray17     @Test
gray18     fun testLoginSuccess() {
gray19         val email = "test@example.com"
gray20         val password = "testPassword"
gray21
gray22         `when`(mockFirebaseAuth.
           signInWithEmailAndPassword(anyString(),
           anyString()))
           .thenReturn(mockAuthResultTask)
gray23         `when`(mockAuthResultTask.isSuccessful).
gray24             thenReturn(true)
gray25
gray26         assertTrue(authRepository.login(email,
           password))
gray27     }
gray28
gray29     @Test
gray30     fun testLoginFailure() {
gray31         val email = "test@example.com"
gray32         val password = "wrongPassword"
gray33
gray34         `when`(mockFirebaseAuth.
           signInWithEmailAndPassword(anyString(),
           anyString()))
           .thenReturn(mockAuthResultTask)
gray35         `when`(mockAuthResultTask.isSuccessful).
gray36             thenReturn(false)
gray37
gray38         assertFalse(authRepository.login(email,
           password))
gray39     }
gray40 }

```

```

gray1 class PostoTest {
gray2
gray3     @Mock
gray4     private lateinit var mockFirestore:
        FirebaseFirestore
gray5
gray6     @Mock
gray7     private lateinit var mockCollectionReference:
        CollectionReference
gray8
gray9     @Mock
gray10    private lateinit var mockDocumentReference:
        DocumentReference
gray11
gray12    @Mock
gray13    private lateinit var mockTask: Task<
        DocumentReference>
gray14
gray15    private lateinit var postoRepository:
        PostoRepository
gray16
gray17    @Before
gray18    fun setUp() {
gray19        MockitoAnnotations.initMocks(this)
gray20        postoRepository = PostoRepository(
            mockFirestore)
gray21    }
gray22
gray23    @Test
gray24    fun testNewPosto() {
gray25        val data = "2023-08-31"
gray26        val utenteId = "user123"
gray27        val postId = "random123"
gray28
gray29        'when'(mockFirestore.collection(anyString())).
            thenReturn(mockCollectionReference)
gray30        'when'(mockCollectionReference.add(any())).
            thenReturn(mockTask)
gray31        'when'(mockTask.addOnSuccessListener(any())).
            thenAnswer {
gray32            (it.arguments[0] as? OnSuccessListener<
                DocumentReference>)?.onSuccess(
                    mockDocumentReference)
gray33            mockTask
gray34        }

```

```
gray35      'when '(mockDocumentReference.id).thenReturn(  
gray36          postId)  
gray37  
gray38      postoRepository.newPosto(data, utenteId)  
gray39  
gray40      verify(mockFirestore).collection("Posti")  
gray41      verify(mockCollectionReference).add(any())  
gray42  }
```

2 Flutter

2.1 Descrizione

L'obiettivo del progetto consiste nella realizzazione di un'applicazione per la gestione della biblioteca dell'università. Coloro che possono accedere all'applicazione sono gli utenti e l'amministratore. L'applicazione non gestisce la procedura di registrazione in quanto, gli utenti che possono accedervi, sono gli studenti iscritti all'università. Gli utenti possono consultare la lista dei libri disponibili e selezionare il libro che desiderano prendere in prestito. Ogni utente può prendere in prestito al massimo 3 libri alla volta. Ogni prestito ha una durata di 21 giorni. Inoltre, gli utenti posso prenotare un posto nella biblioteca. La prenotazione di un posto vale per tutta la giornata, non ci sono fasce orarie. Per restituire il libro preso in prestito, l'utente deve recarsi nella biblioteca e restituirlo; sarà compito dell'amministratore registrare l'avvenuta restituzione.

2.2 Requisiti

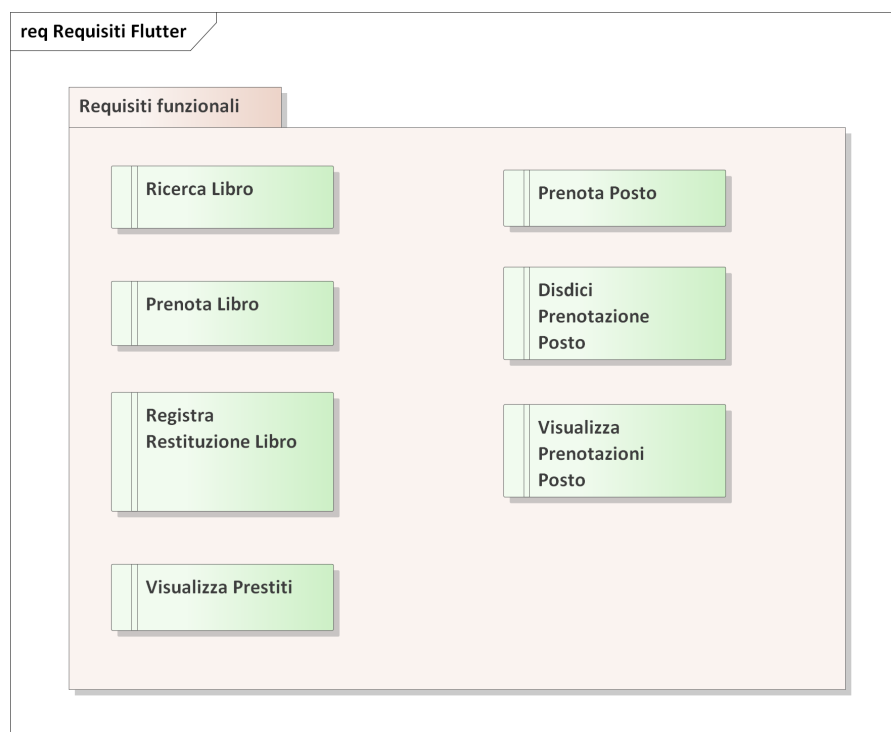


Figure 8: Requisiti

2.3 Casi d'uso

2.4 Attori

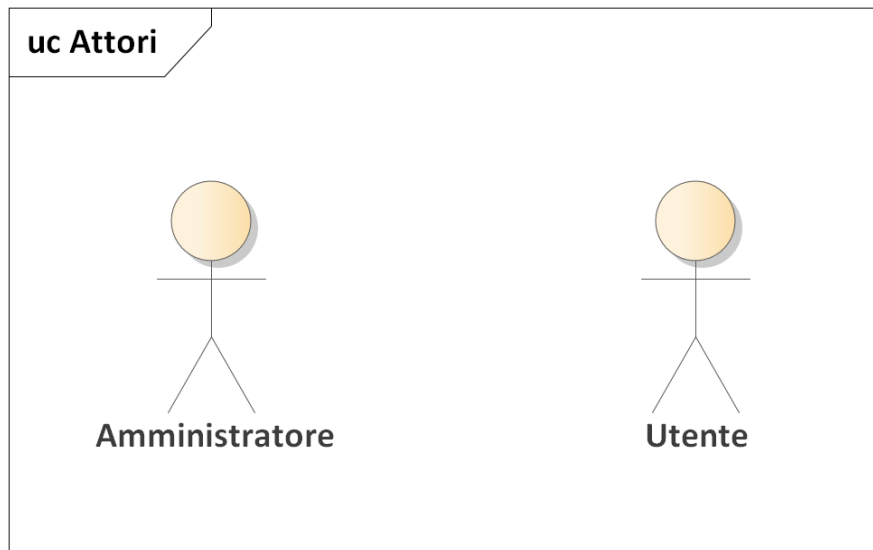


Figure 9: Attori

2.5 Diagrammi dei casi d'uso

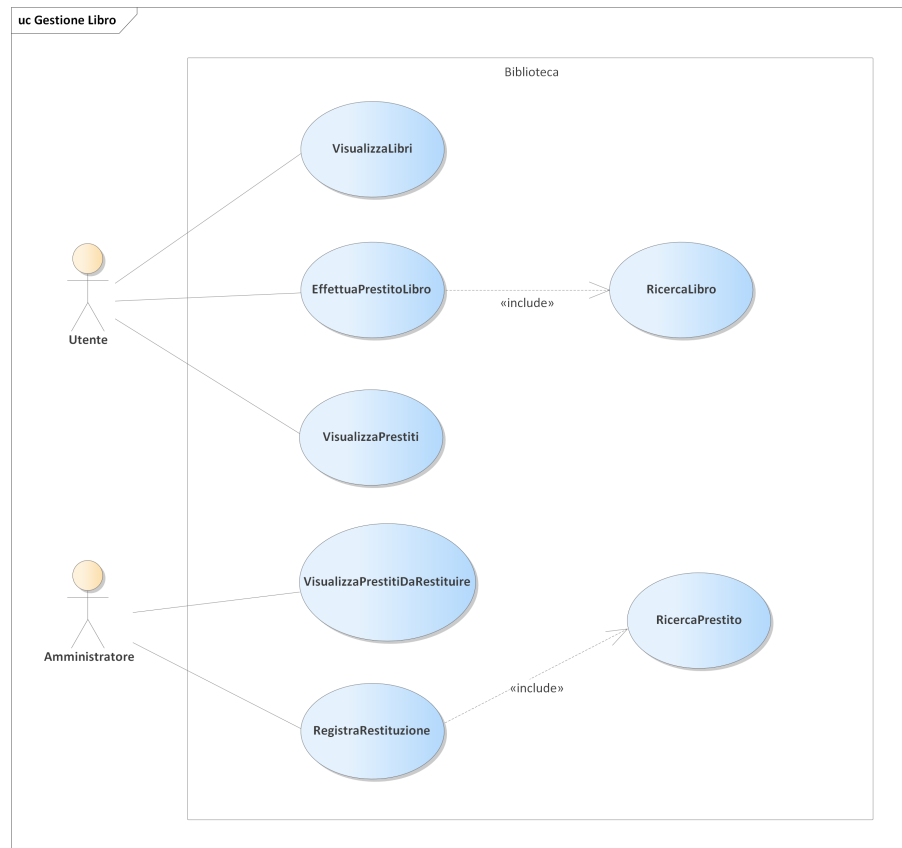


Figure 10: GestioneLibro

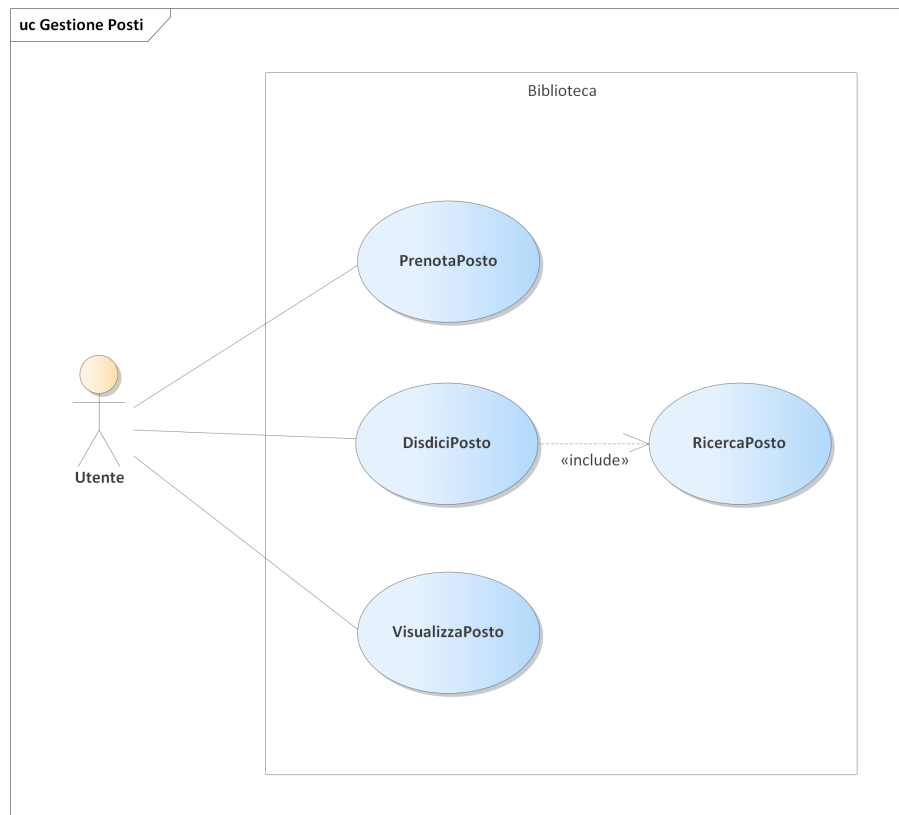


Figure 11: GestionePosti

2.6 Descrizione dei casi d'uso

Titolo: EffettuaPresitoLibro

Descrizione: il caso d'uso permette all'utente di prendere in prestito un libro

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole prendere in prestito un libro
 2. include (RicercaLibro)
 3. if l'utente può prendere in prestito il libro:
 - (a) il sistema memorizza il prestito
 - (b) il sistema decrementa la quantità del libro
 4. else il sistema mostra un messaggio di errore.
-

Titolo: RegistraRestituzione

Descrizione: il caso d'uso permette di registrare la restituzione di un libro

Attori primari: Amministratore

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole registrare l'avvenuta restituzione di un libro
 2. include (RicercaPrestito)
 3. il sistema memorizza l'avvenuta restituzione del libro
 4. il sistema incrementa la quantità del libro.
-

Titolo: RicercaLibro

Descrizione: il caso d'uso permette di ricercare un libro.

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole ricercare un libro
2. l'attore primario inserisce i dati chiave per la ricerca
3. il sistema ritorna i dati relativi al libro.

Titolo: VisualizzaLibri
Descrizione: il caso d'uso permette di visualizzare i libri.
Attori primari: Utente
Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare i libri item for each libri:
 - (a) il sistema visualizza a schermo i dati relativi ai libri.

Titolo: VisualizzaPrestiti
Descrizione: il caso d'uso permette di visualizzare i prestiti dell'utente.
Attori primari: Utente
Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare la lista dei prestiti
2. for each prestiti dell'utente:
 - (a) il sistema visualizza a schermo i dati relativi al prestito.

Titolo: VisualizzaPrestitiDaRestituire
Descrizione: il caso d'uso permette di visualizzare i prestiti ancora da restituire.
Attori primari: Amministratore
Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare la lista dei prestiti da restituire
2. for each prestiti dell'utente:
 - (a) il sistema visualizza a schermo i dati relativi al prestito.

Titolo: RicercaPrestito
Descrizione: il caso d'uso permette di ricercare un prestito.
Attori primari: Amministratore
Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole ricercare un
2. l'attore primario inserisce i dati chiave per la ricerca
3. il sistema ritorna i dati relativi al libro.

Titolo: PrenotaPosto

Descrizione: il caso d'uso permette di prenotare un posto nella biblioteca

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole prenotare un posto
2. if l'utente può prenotare il posto:
 - (a) il sistema memorizza la prenotazione
3. else il sistema mostra un messaggio di errore.

Titolo: DisdiciPosto

Descrizione: il caso d'uso permette di disdire la prenotazione di un posto nella biblioteca

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole disdire la prenotazione di un posto
2. include (RicercaPosto)
3. il sistema memorizza la disdetta.

Titolo: VisualizzaPosto

Descrizione: il caso d'uso permette di visualizzare la lista delle prenotazioni dei posti

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole visualizzare le prenotazioni dei posti
2. foreach prenotazioni non scadute fatte dall'utente:
 - (a) visualizza dati prenotazione.

Titolo: RicercaPosto

Descrizione: il caso d'uso permette di ricercare

Attori primari: Utente

Sequenza eventi principali:

1. il caso d'uso inizia quando l'attore primario vuole ricercare la prenotazione di un posto
 2. l'attore primario inserisce i valori chiave per la ricerca
 3. il sistema ritorna i dati relativi alla prenotazione del posto
-

2.7 Architettura

Per lo sviluppo dell'applicazione non è stato usato nessun pattern architetturale.

2.8 Database

Il database è lo stesso dell'app in Android.

2.9 Sviluppo

La prima schermata dell'applicazione è quella per effettuare il login. La procedura di autenticazione è stata gestita tramite Firebase Authentication. A seconda del ruolo, l'utente è indirizzato o nella home dell'admin o in quella dell'utente. La struttura delle varie schermate è simile: una toolbar in alto con un navigation drawer, per navigare tra le schermate.

Per quanto riguarda la home dell'utente, la prima schermata è una schermata di benvenuto. Tramite il navigation drawer l'utente può navigare tra le varie schermate. Nella sezione 'catalogo' si viene portati alla lista dei libri disponibili. Nel caso in cui l'utente abbia dei prestiti scaduti, comparirà un popup per avvertirlo ogni volta che ritorna nel catalogo. I libri sono visualizzati come una lista di card all'interno di una recycler view. In alto, si trova una search view che filtra i libri del catalogo. I parametri per la ricerca sono titolo, autore, isbn e descrizione.

Cliccando su una card, l'utente può vedere ulteriori dettagli relativi al libro. Inoltre, può prenderlo in prestito, se c'è più di una copia disponibile, il libro non è già stato preso in prestito dall'utente e l'utente non ha superato il numero massimo di prestiti. Il prestito inizia nel momento in cui l'utente conferma di volere prendere in prestito il libro e la durata è di 21 giorni.

Nella sezione 'lista prestiti', l'utente può vedere solo i suoi prestiti, ordinati per data.

Nella sezione 'prenota un posto', l'utente può visualizzare le prenotazioni dei posti effettuate e non scadute. La prenotazione di un posto vale per tutto il giorno. In basso, c'è il bottone per prenotare un posto nella biblioteca. Per effettuare la prenotazione, l'utente non deve avere già una prenotazione per lo stesso giorno e ci devono essere abbastanza posti disponibili per quel giorno. In ogni card è presente anche l'icona per disdire la prenotazione.

Per quanto riguarda, invece, la home dell'admin, la struttura è simile a quella dell'utente, ma l'unica funzionalità riguarda la registrazione della restituzione di un libro.

Per la navigazione è stato usato il metodo 'generateRoute', che genera le rotte in base al nome della rotta.

```

12 class Routes{
13   static Route<dynamic> generateRoute(RouteSettings settings){
14     final args = settings.arguments;
15
16     switch(settings.name){
17       case '/login':
18         return MaterialPageRoute(builder: (context) => AuthPage());
19       case '/logout':
20         return MaterialPageRoute(builder: (context) => AuthPage());
21       case '/catalogo':
22         return MaterialPageRoute(builder: (context) => Catalogo());
23       case '/home_utente':
24         return MaterialPageRoute(builder: (context) => HomeUtente());
25       case '/home_admin':
26         return MaterialPageRoute(builder: (context) => HomeAdmin());
27       case '/registra_restituzioni':
28         return MaterialPageRoute(builder: (context) => RegistraRestituzioni());
29       case '/lista_prestiti':
30         return MaterialPageRoute(builder: (context) => ListaPrestiti());
31       case '/lista_posti':
32         return MaterialPageRoute(builder: (context) => ListaPosti());
33       case '/detail_book':
34         if (args is DetailBook) return MaterialPageRoute(builder: (context) => DetailBook(titolo: args.titolo,
35         autore: args.autore, isbn: args.isbn, disponibili: args.disponibili,
36         descrizione: args.descrizione, immagine: args.immagine)); // DetailBook, MaterialPageRoute
37         else return MaterialPageRoute(builder: (context) => Error());
38       case 'errore':
39         return MaterialPageRoute(builder: (context) => Error());
40       default:
41         return MaterialPageRoute(builder: (context) => Error());
42     }

```


2.10 Bug

Quando si apre il dialog per far selezionare la data all'utente, per prenotare un posto, cliccando su annulla, viene comunque presa la data iniziale (quella di oggi).

Se l'utente ha dei prestiti scaduti, quando l'admin registra la restituzione di uno dei suoi prestiti, vede anche lui il popup del prestito in ritardo. Questo è dovuto al fatto che ogni volta che modifico i dati della collezione viene inviata una notifica, quindi anche quando l'admin registra la restituzione, poichè lo stream è lo stesso.

2.11 Mockup

Login



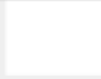
Login

Catalogo

Catalogo



Card Libro



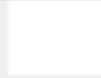
Titolo:
Autore:

Card Libro



Titolo:
Autore:

Card Libro



Titolo:
Autore:

Home



Biblioteca

benvenuto!