



UNIVERSITÀ
DI TRENTO

Dipartimento di Ingegneria e Scienza
dell'Informazione

Progetto:



Titolo del documento:

Documento di sviluppo

Document Info:

Doc. Name	<i>D4-XBooks_Documento_Sviluppo</i>	Doc. Number	<i>D4</i>
Description	<i>Il documento include la descrizione del backend, con test e swagger, e frontend</i>		

1. Scopo del documento	2
2. UserFlow	3
3. Implementazione e documentazione delle API	4
3.1 Struttura del BackEnd	4
3.2 Dipendenze del progetto	5
3.3 Gestione dati nel database	6
3.4 Specifica delle risorse	6
3.4.1 Estrazione delle risorse	7
3.4.2 Modello delle risorse	10
4. API	17
5. Documentazione delle API	33
6. Implementazione del FrontEnd	35
7. GitHub repository e deployment	41
8. Testing delle API	42

1. Scopo del documento

Il presente documento riporta lo sviluppo del progetto XBooks. In particolare si parlerà di:

- Specifica delle risorse
- Implementazione e documentazione delle API
- Testing delle API
- Implementazione del frontend
- Deployment del progetto

Segue una breve descrizione del progetto:

Il sito permette la **registrazione** e **accesso** alla web app degli utenti. L'utente, dopo aver fatto l'accesso, potrà **cercare** il libro che sta leggendo, **aggiungerlo** a una delle proprie librerie e **modificare** il numero delle pagine lette. L'utente potrà anche **aggiungere** o **eliminare** una o più librerie, in base alle proprie necessità, dall'apposita sezione delle librerie personali. Accedendo all'interno di una delle proprie librerie, potrà **vedere** la lista dei libri presenti nella suddetta e, premendo su l'immagine di uno di questi, vedere a che punto si trova del libro. In questa

pagina potrà anche **togliere** il libro dalla libreria. Oltre tutto l'utente ha la possibilità di **modificare** il proprio username e la propria password e **eliminare** il proprio account.

2. UserFlow

Di seguito verranno riportati gli User Flows dell'utente in base alle funzioni che è in grado di eseguire durante l'utilizzo della web app.

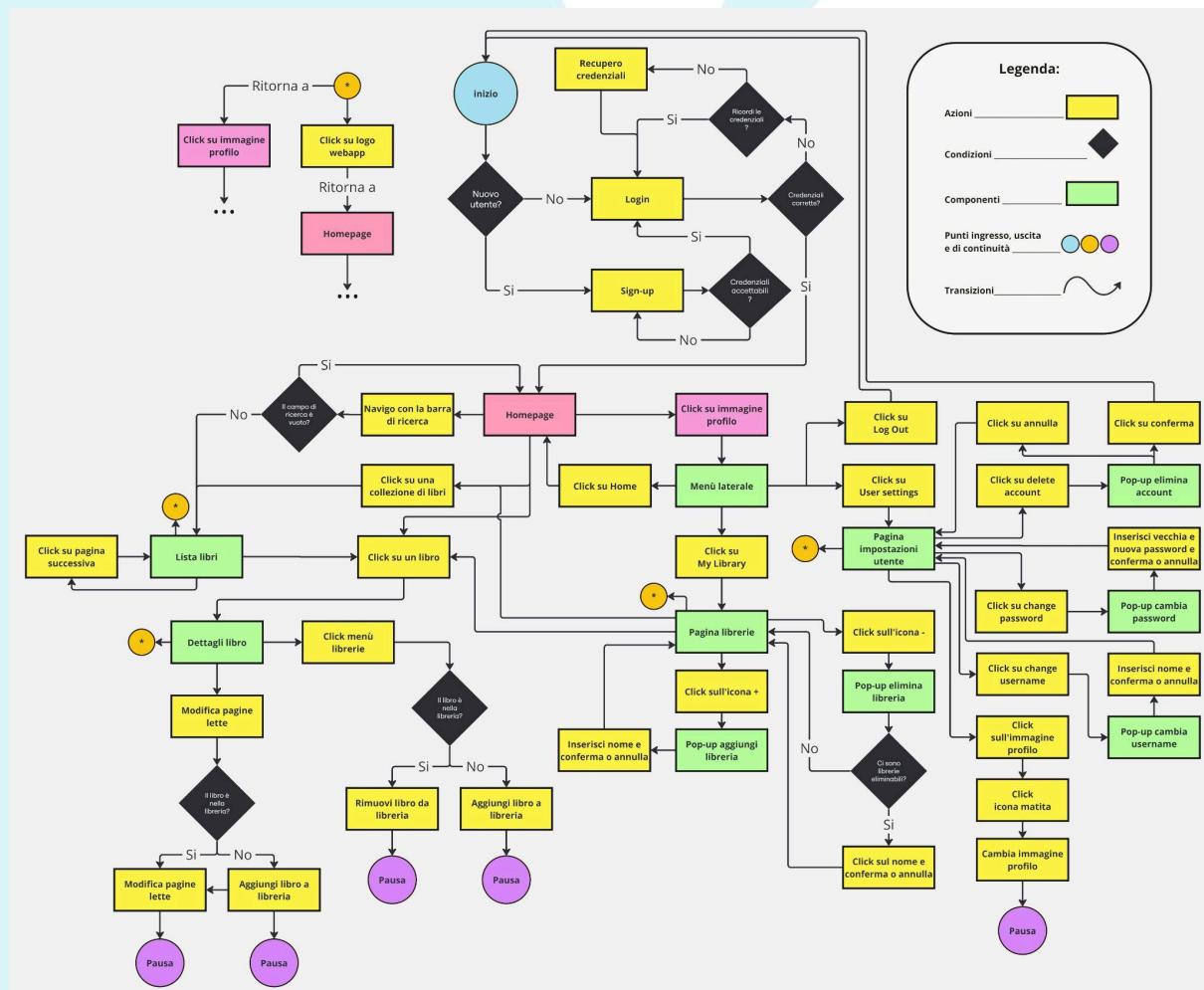


figura 1

3. Implementazione e documentazione delle API

In questo capitolo si parlerà dell'implementazione e documentazione delle API.

3.1 Struttura del BackEnd

Prima di iniziare a descrivere le API è necessario mostrare la struttura del backend che si trova nella figura di seguito (figura 2). I vari file: **.env**, **.gitignore**, **jest.config.js**, **package.json**, **package-lock.json**, **tsconfig.json** e **swagger.json**, sono necessari per il corretto funzionamento del progetto. In questa sezione però parleremo del file **server.ts**, della cartella **/src**, e della directory **/tests**.

server.ts è il file principale del backend perché importa il modulo app, che serve a configurare i router, **cors** e swagger, e, infine, avvia il server.

/src è la directory dove ci sono tutti i models, i router, **database.ts** per avviare la connessione al database e **script.ts** dove ci sono vari moduli di script che serviranno a delle API.

/tests contiene i file necessari per il testing delle API.

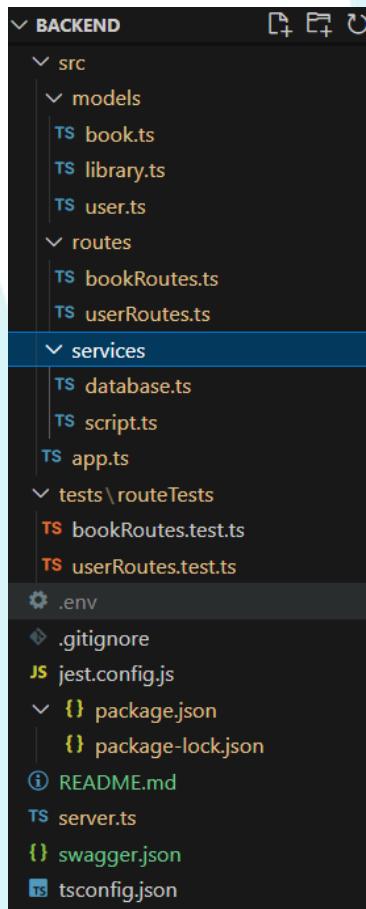


figura 2

3.2 Dipendenze del progetto

Le dipendenze del progetto sono essenziali per il corretto funzionamento di esso e il backend di **XBooks** dipende da:

- **cors**: permette a server esterni di accedere alle funzioni del backend.
- **mongodb**: permette di gestire la connessione al database MongoDB e gli ObjectId.
- **jsonwebtoken**: permette di gestire l'utilizzo dei token.
- **bcryptjs**: permette di comparare e criptare le password, permettendo di assicurare sicurezza e privacy agli utenti.
- **dotenv**: permette di gestire le variabili d'ambiente.
- **express**: è il framework di base per la creazione del server.
- **jest e supertest**: permettono la realizzazione dei testing.
- **swagger-ui-express**: utilizzato per gestire la documentazione delle API.

- **typescript** e i vari **@types**: come linguaggio utilizzato nel backend per assicurare una maggior sicurezza grazie al controllo dei **types**.

3.3 Gestione dati nel database

Come database per questo progetto ci siamo affidati a **MongoDB**: un database non relazionale orientato a documenti. Abbiamo utilizzato una sola **collection** perché avevamo bisogno di memorizzare un solo modello: lo **User**. Questo perché l'utente è stato salvato nel database come nella **figura 3** e implementato come nella **figura 4** che vedremo nella sezione successiva.

```

▶ _id: ObjectId('66cfc54da1cdd2c04796d1f7')
  name : "Pat"
  email : "riccardo.patalocchi@gmail.com"
  password : "$2a$10$7nzzbtSvf73PP.2ZvX9ZTeq7C4v8yf2fl/IY3NZvWtNNeDxEUOBfe"
  ▶ library : Array (2)
    ▶ 0: Object
      libName : "Your Books"
      libId : "1"
      ▶ books : Array (1)
        ▶ 0: Object
          bookId : "88"
          pagesRead : 16
    ▶ 1: Object
      libName : "gjx"
      libId : "lib-1724892657473"
      ▶ books : Array (1)
        ▶ 0: Object
          bookId : "184"
          pagesRead : 0
:
:
```

figura 3

3.4 Specifica delle risorse

Questa sezione sarà fondamentale per il capitolo successivo in cui verrà mostrata l'implementazione delle API.

3.4.1 Estrazione delle risorse

In questo paragrafo andremo a illustrare le risorse estratte dal Class Diagram che potete trovare nel documento precedente (D3).

<risorsa> User

La risorsa User ha come attributi:

- **name**: nickname che comparirà nella web app.
- **email**: credenziale necessaria per fare l'accesso.
- **password**: credenziale necessaria per fare l'accesso.
- **library**: creata lato server, spiegheremo l'implementazione particolare fra poco.
- **_id**: per gestire l'id nel database. Attributo che viene generato lato client.

Il codice della risorsa lo potrete osservare nella **figura 4** sottostante:

```
// Model for User
You, 3 minutes ago | 1 author (You)
export default class User {

    constructor(
        public name: string,
        public email: string,
        public password: string,
        public library: Library[],
        public _id: ObjectId
    ) {
        // Check if the ID is a valid ObjectId
        if(this._id && typeof this._id === 'string') {
            try {
                this._id = new ObjectId(this._id as string);
            } catch(error) {
                throw new Error('Invalid ID format');
            }
        }
    }
}
```

figura 4

Come potrete notare, il campo **library** non è di un tipo built-in di typescript, questo perché avevamo bisogno di creare una struttura particolare che potesse adattarsi alle nostre esigenze.

Nella **figura 5** viene mostrato com'è stata implementata la **library** dell'utente. Abbiamo pensato di inserire nel modello dello user un Array(**container delle librerie**) di tuple(**librerie**) del tipo `[string, string, array]` che a sua volta l'array all'interno(**libri della libreria**) è un array di tuple(**libri**) del tipo `[string, number]` il risultato che viene fuori è: `{[libName, libId, {[bookId, pagesRead], [...]}], [...]}`. In questo modo riusciamo a gestire tutti i libri e le librerie dell'utente. Abbiamo bisogno di due variabili per identificare una singola libreria nel caso che l'Utente chiama più librerie con lo stesso nome, con il **libId** risolviamo questo piccolo problema.

```
// Model for book
You, last week | 1 author (You)
export default interface BookTuple {
  bookId: string;
  pagesRead: number;
}

// Model for library
You, 2 minutes ago | 1 author (You)
export default interface Library {
  libName: string;
  libId: string;
  books: BookTuple[];
}
```

figura 5

API

<risorsa> registrazione: API che permette a un nuovo Utente di registrarsi. È una richiesta POST e ha come parametri: name, email, password e userId.

<risorsa> login: API che permette a un Utente registrato di accedere ai servizi della web app. È una richiesta POST e ha come parametri: email, password.

<risorsa> getUser: API necessaria per ricevere informazione sull'Utente registrato e che ha fatto il login. È una richiesta GET e ha come parametro: userId.

<risorsa> modifyUsername: API che permette all'Utente registrato e che ha fatto il login di modificare il proprio username(attributo **name**). È una richiesta

PUT e ha come parametri: userId e newUsername, inoltre ha bisogno dell'accessToken che viene inserito nell'header authorization della richiesta.

<risorsa> modifyPassword: API che permette all'Utente registrato e che ha fatto il login di modificare la propria password. È una richiesta PUT e ha come parametri: userId, oldPassword e newPassword, inoltre ha bisogno dell'accessToken che viene inserito nell'header authorization della richiesta.

<risorsa> deleteProfile: API che permette all'Utente registrato e che ha fatto il login di eliminare il proprio profilo. È una richiesta DELETE e ha come parametro: userId, inoltre ha bisogno dell'accessToken che viene inserito nell'header authorization della richiesta.

<risorsa> getBook: API del backend necessaria a richiedere il libro cliccato dall'Utente per potergli mostrare le sue specifiche. È una richiesta GET e ha come parametri: libId, bookId e userId.

<risorsa> getSpecLibrary: API necessaria a richiedere una libreria specifica per poterla visualizzare all'Utente una volta cliccata. È una richiesta GET e ha come parametri: libId e userId.

<risorsa> getLibraries: API del backend necessaria a richiedere tutte le librerie dell'Utente per poterle visualizzare nella sezione apposita. È una richiesta GET e ha come parametri: libId e userId.

<risorsa> modifyPages: API che modifica il numero delle pagine lette di un libro specifico dall'Utente. È una richiesta PUT e ha come parametri: bookId, libId, pages e userId.

<risorsa> createLibrary: API che permette all'Utente di creare una nuova libreria. È una richiesta POST e ha come parametri: libName, libId e userId.

<risorsa> addBook: API che permette all'Utente di aggiungere un libro a una libreria preesistente. È una richiesta POST e ha come parametri: bookId, libId e userId.

<risorsa> deleteBook: API che permette all'Utente di togliere un libro da una libreria preesistente. È una richiesta DELETE e ha come parametri: libId, bookId e userId.

<risorsa> deleteLibrary: API che permette all'Utente di eliminare una libreria preesistente. È una richiesta DELETE e ha come parametri: libId e userId.

3.4.2 Modello delle risorse

In questa sezione verranno mostrati i modelli delle risorse.

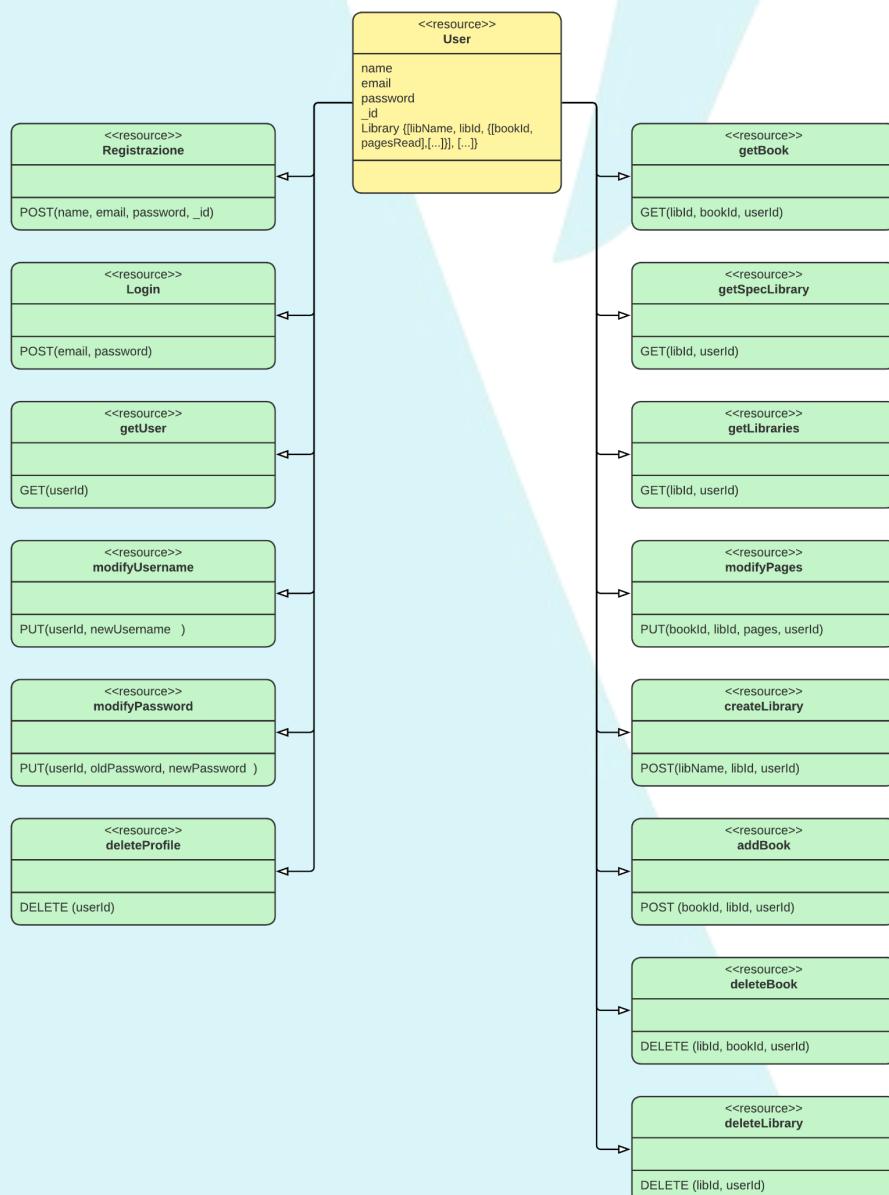


figura 6

Registrazione

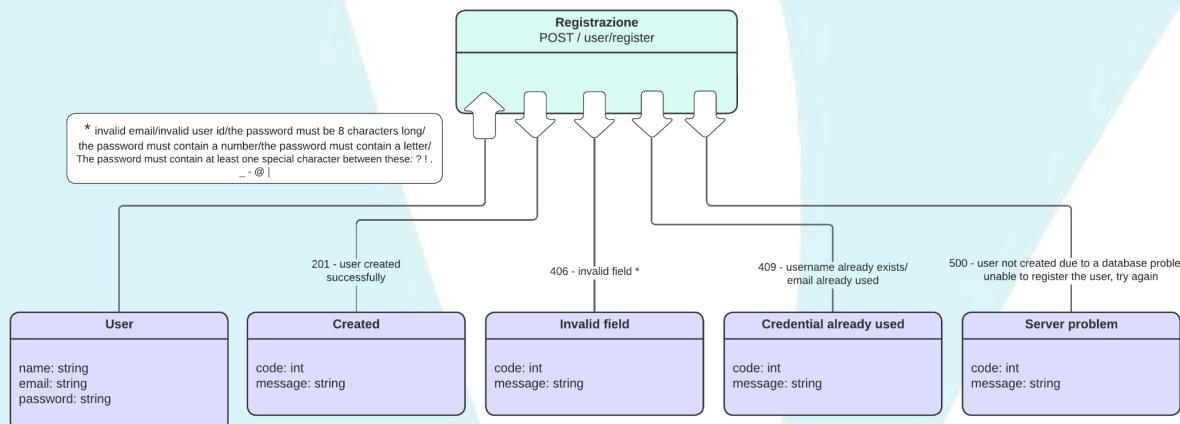


figura 7

Login

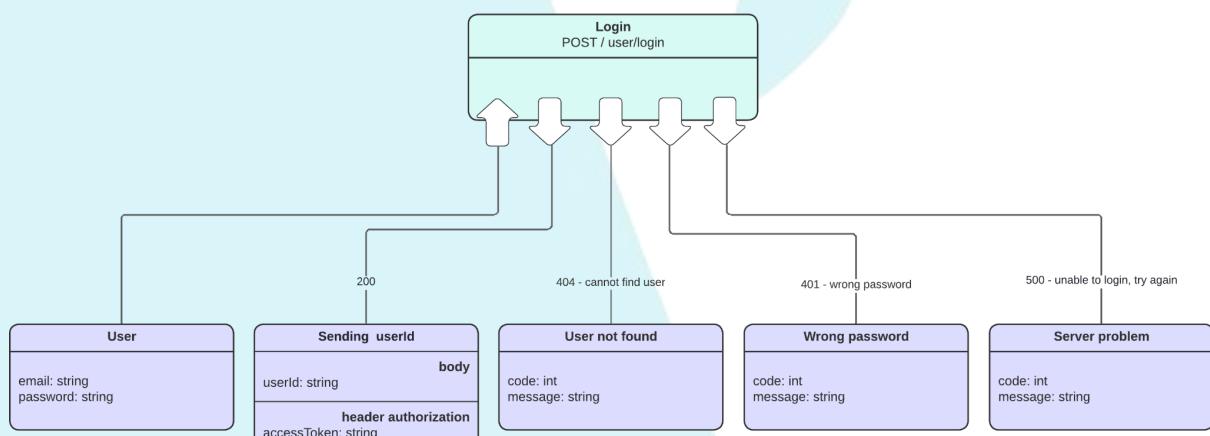


figura 8

getUser

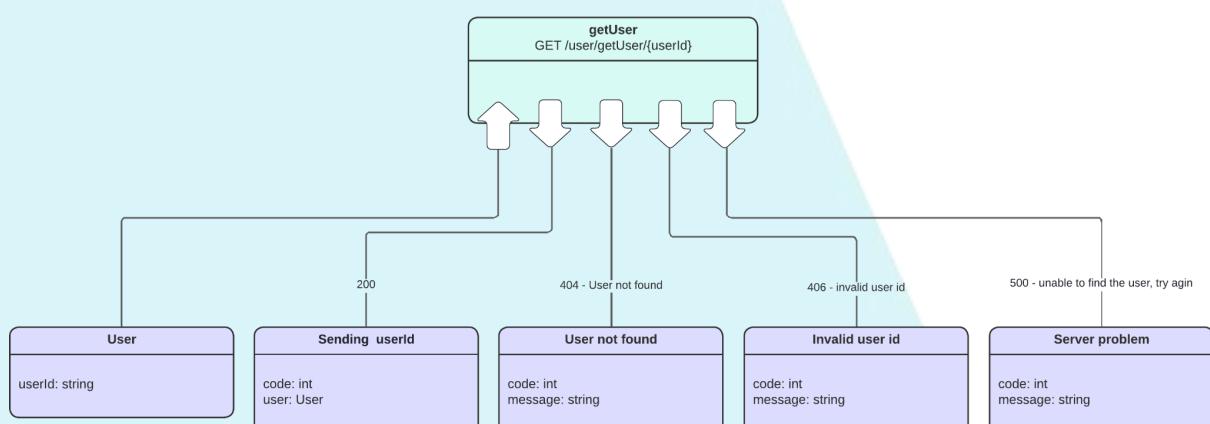


figura 9

modifyUsername

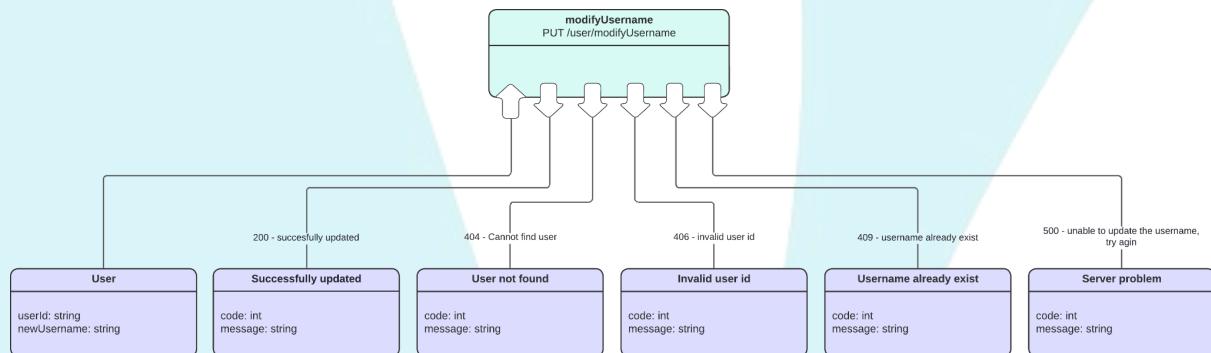


figura 10

modifyPassword

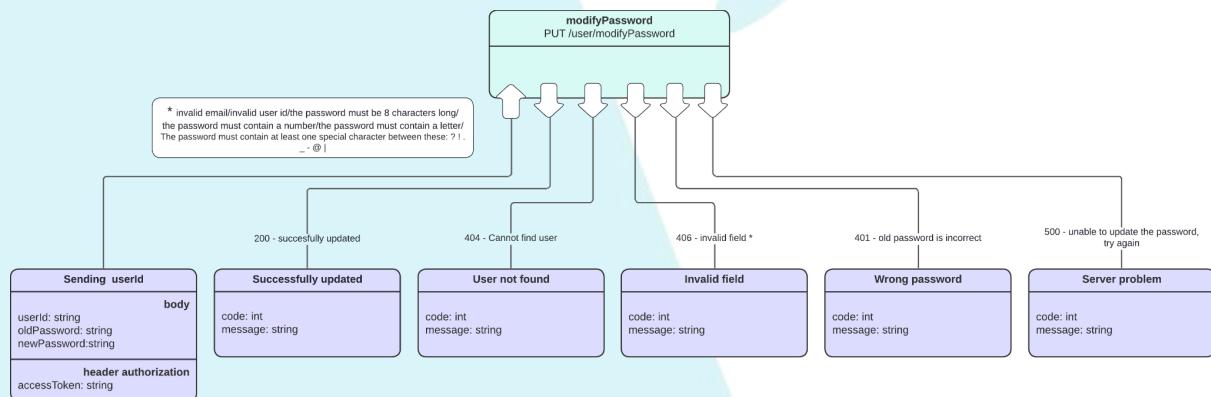


figura 11

deleteProfile

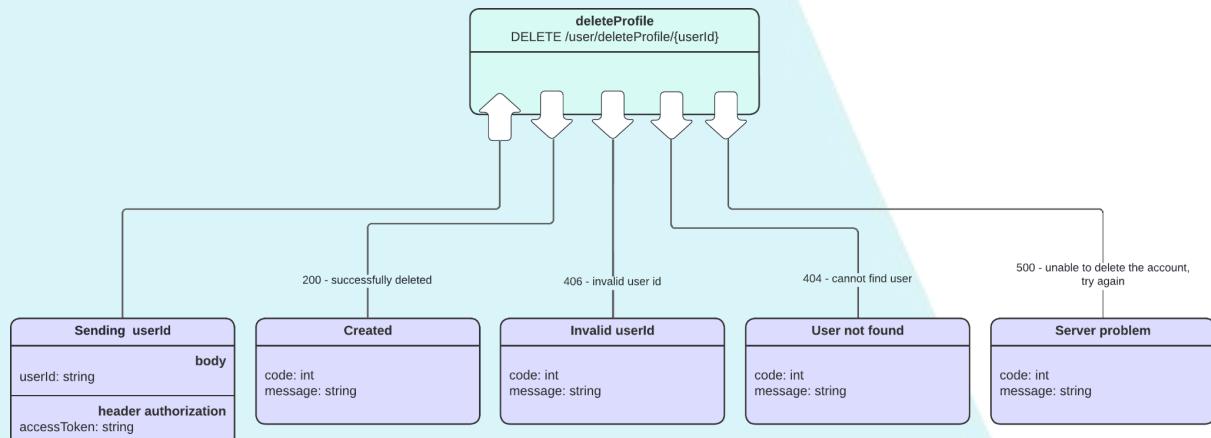


figura 12

getBook

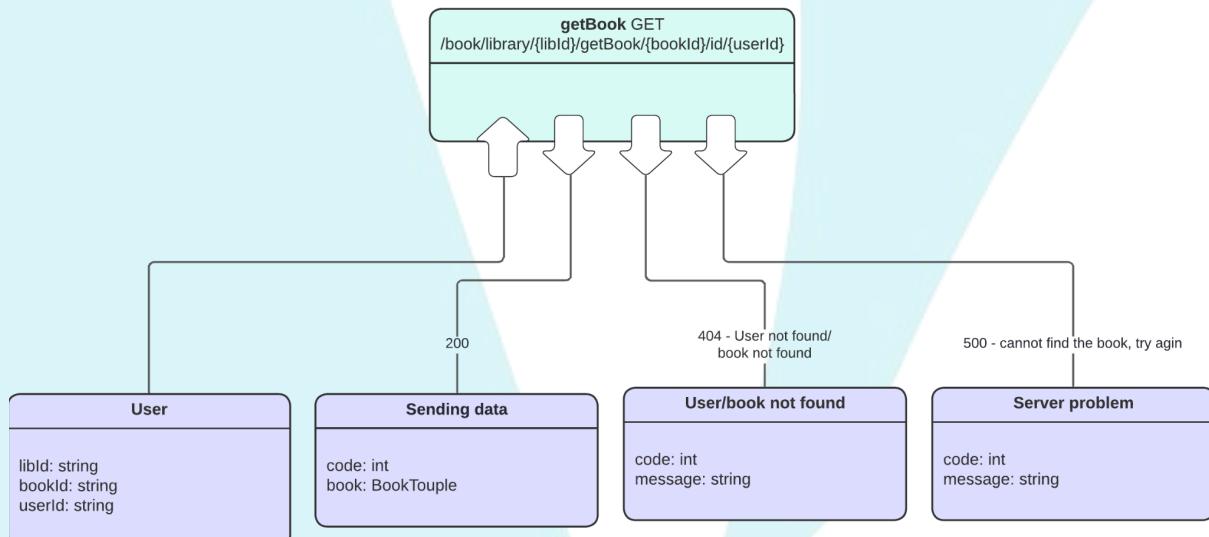


figura 13

getSpecLibrary

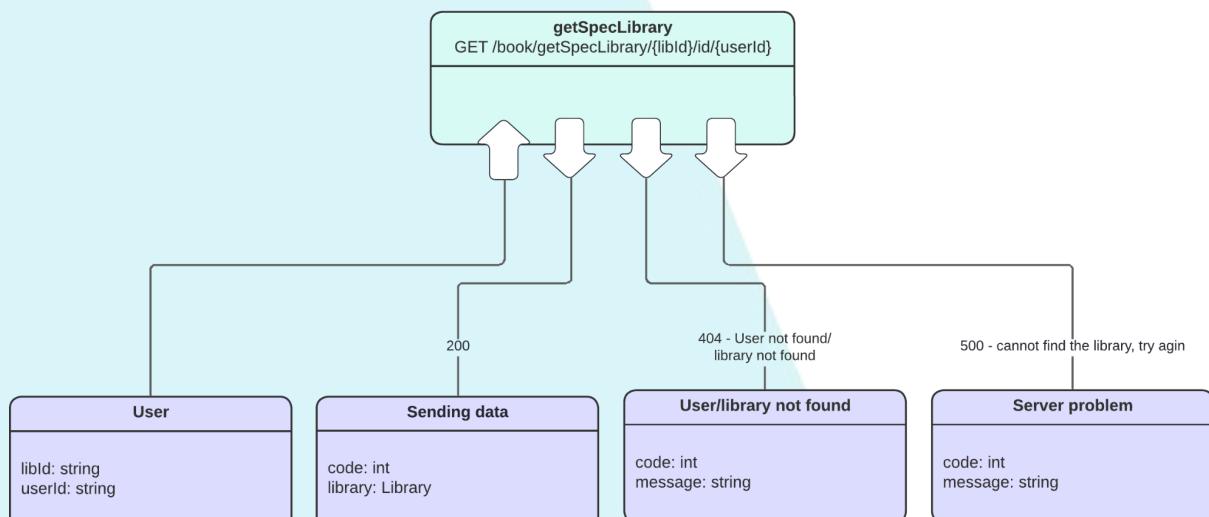


figura 14

getLibraries

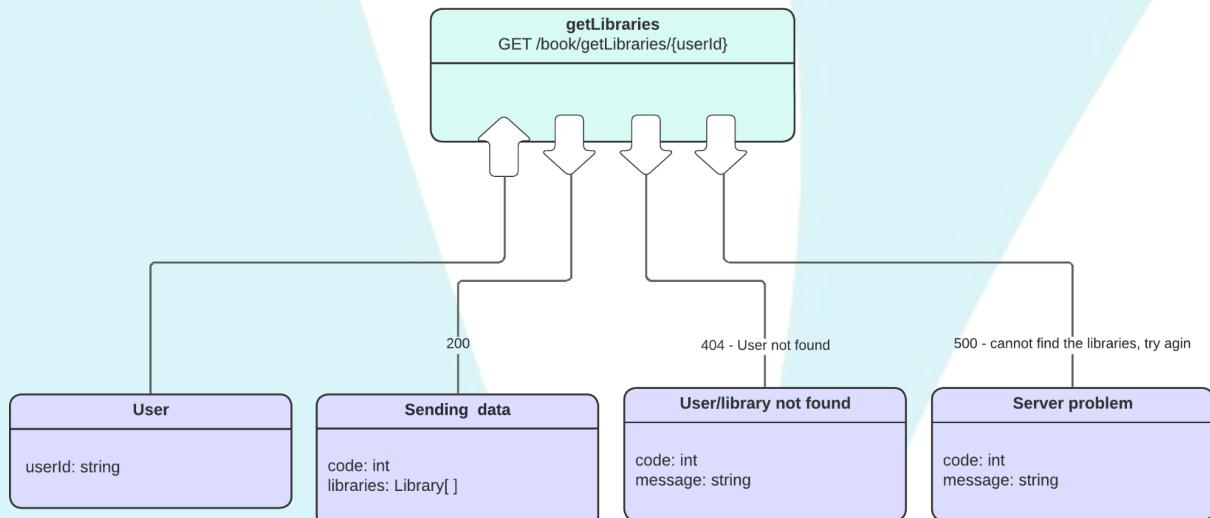


figura 15

modifyPages

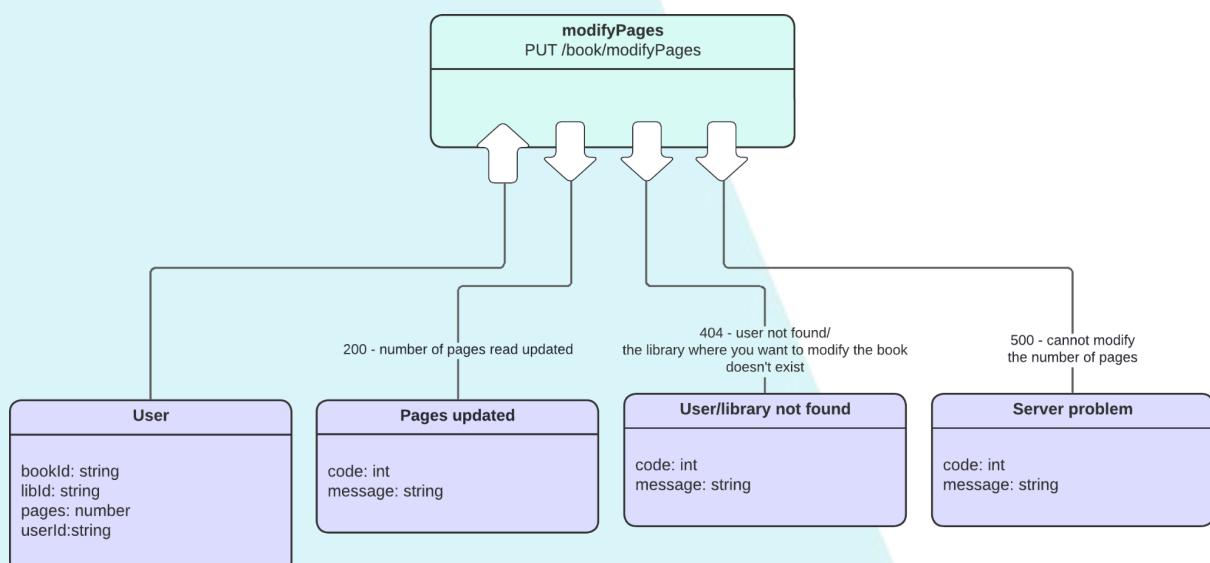


figura 16

createLibrary

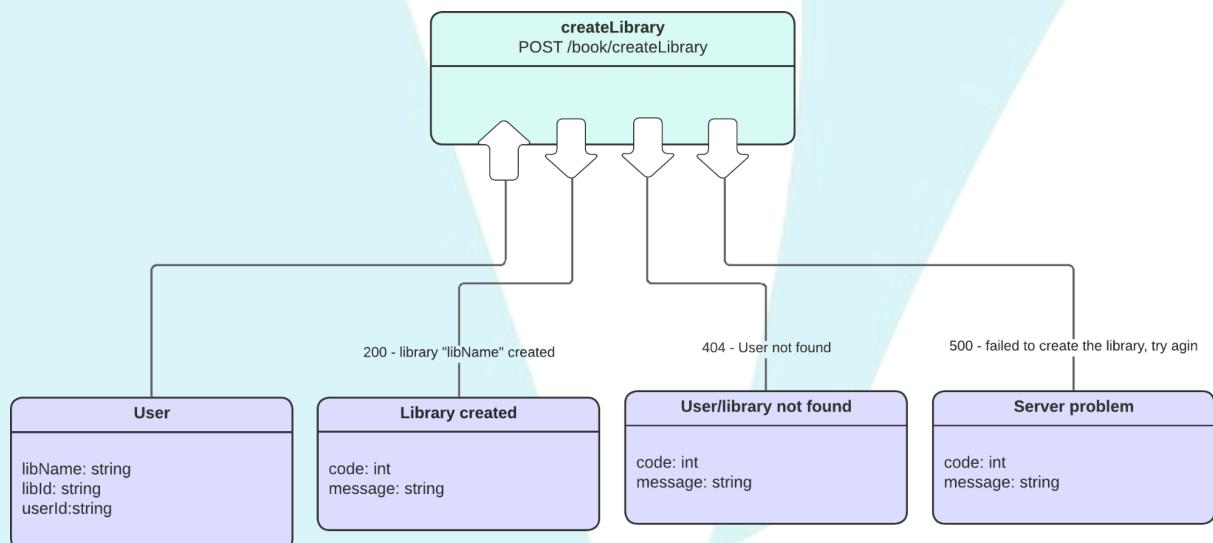


figura 17

addBook

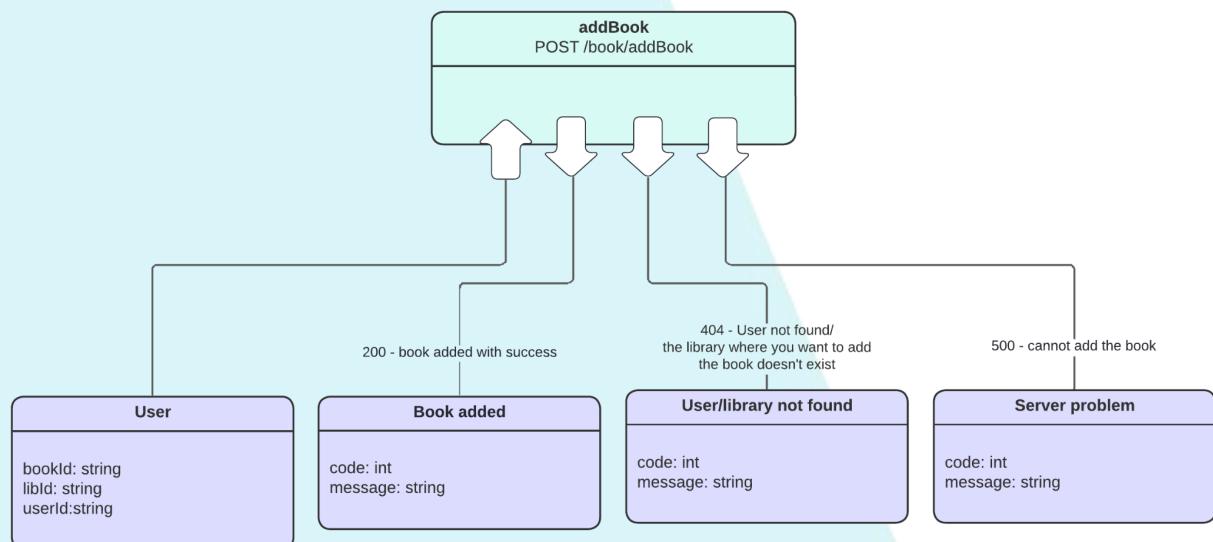
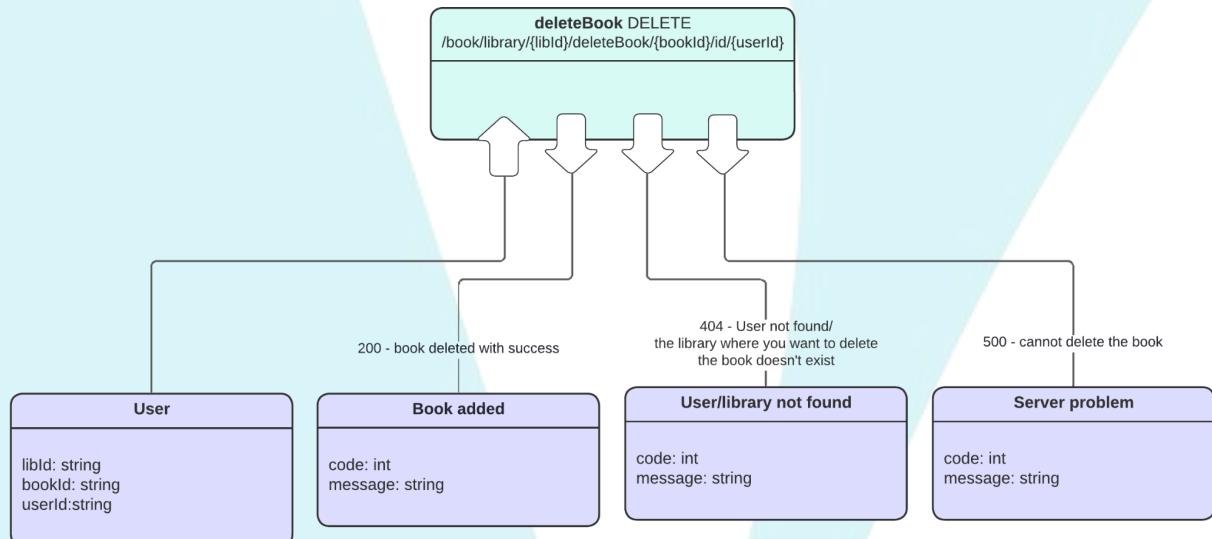
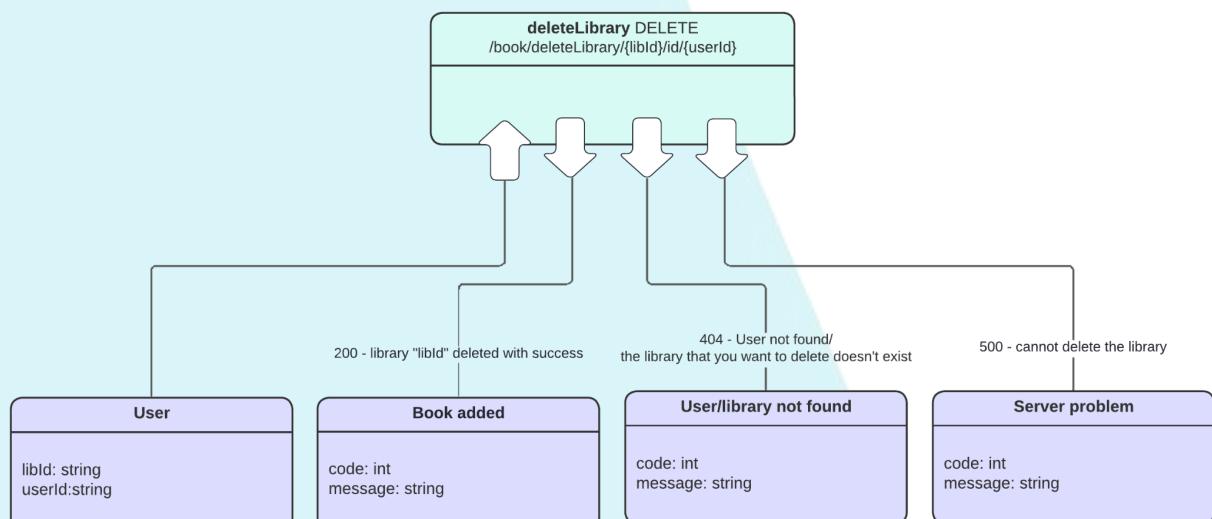


figura 18

deleteBook**figura 19****deleteLibrary****figura 20**

4.API

Abbiamo realizzato un totale di 14 API prese dalle risorse. Le abbiamo divise in due gruppi: UserRoutes e BookRoutes. Di seguito avrete una spiegazione dettagliata su come sono state implementate con immagine del codice annessa.

UserRoutes

Registrazione

Quest'API è disponibile all'indirizzo **/user/register**. È una richiesta POST e richiede nel body della richiesta: **name(string)**, **email(string)**, **password(string)** e **userId(string)**. Aggiunge il campo **library(Library[])** per avere tutti gli attributi dell'User e, dopo aver controllato che non esistano altri utenti con lo stesso nome o email, che l'userId può essere un ObjectId valido per poterlo usare come campo _id nel database, che l'email sia valida con il modulo **validateEmail**, che la password sia valida con il modulo **validatePassword** e, successivamente, dopo averla criptata, salva l'User nel database. Può inviare quattro diversi status code di risposta: **201** con message = “User created successfully” e invia il risultato del salvataggio dell'User nel database, **406** che può avere diversi message = “Invalid email”/“Invalid user id”/“The password must be 8 characters long”/“The password must contain a letter”/“The password must contain a number”/“The password must contain at least one special character between these: ?! . _ - @ !”, **409** con message = “Username already exists” o “Email already used” e infine **500** con message = “User not created due to a database problem” o “Unable to register the user, try again”. Nella **figura 21** sottostante viene mostrato il codice di questa API.

```

91 // API used to register
92 router.post('/register', async (req: Request, res: Response) => {
93
94   try {
95
96     // Set the parameters
97     const name: string = req.body.name;
98     const email: string = req.body.email;
99     const password: string = req.body.password;
100    const prelibRARY: Library = { libName: "Your Books", libId: "1", books: [] };
101    const library: Library[] = [prelibRARY]; // Default library
102    const objectId: string = req.body.accessId;
103
104    // Check if the userId is valid
105    if (!ObjectId.isValid(objectId)) {
106
107      // Check if the email is valid
108      if (!validateEmail(email)) {
109
110        // Check if the password is valid
111        if (!validatePassword(password, req, res)) {
112
113          const userId: ObjectId = new ObjectId(objectId);
114
115          // Hash the password
116          const salt = await bcrypt.genSalt();
117          const hashedPassword = await bcrypt.hash(password, salt) as string;
118
119          // Get the database
120          const db = await databaseService.getDb();
121
122          // Check if the email and the username are already used
123          const existEmail = await db.collection('users').findOne({ email: email }) as User | null;
124          const existUsername = await db.collection('users').findOne({ name: name }) as User | null;
125
126          // If the email and the username are not used, create the user
127          if (!existEmail) {
128            if (!existUsername) {
129
130              const user = new User(name, email, hashedPassword, library, userId) as User | null;
131
132              // Check if the user is not null, then insert it
133              if (user) {
134
135                const result = await db.collection("users").insertOne(user);
136
137                res.status(201).json(result);
138
139              } else {
140
141                res.status(500).send("User not created due to a database problem");
142
143              }
144
145            }
146
147          }
148
149        }
150
151        res.status(400).send("Email already used");
152
153      }
154
155    }
156
157  }
158
159  catch(error: any) {
160
161    res.status(500).send("Unable to register the user, try again");
162
163  }
164
165  })
166
167 });
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
634
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
```

```
//API used to login
router.post('/login', async(req: Request, res: Response) => {
  try{
    // Get the database
    const db = await databaseService.getDb();

    // Set the parameters
    const searchEmail: string = req.body.email;

    const possibleUser = await db?.collection("users").findOne({email: searchEmail }) as User | null;

    // Check if the user exists
    if(!possibleUser) {

      return res.status(404).send('Cannot find user');
    }

    // Check if the password is correct
    if(await bcrypt.compare(req.body.password, possibleUser.password)) {

      // Create the token
      const accessToken = jwt.sign({ email: possibleUser.email, _id: possibleUser._id }, process.env.ACCESS_TOKEN_SECRET!);

      // Send the token and the userId
      res.set('Authorization', `${accessToken}`);
      res.status(200).json({ userId: possibleUser._id });

    } else {

      res.status(401).send('Wrong password');
    }
  } catch (error: any) {
    res.status(500).send('Unable to login, try again');
  }
});
```

figura 22

getUser

Quest'API è disponibile all'indirizzo **/user/getUser/{userId}**. È una richiesta GET e richiede nell'URL il parametro **userId**(*string*). L'API controlla se l'userId è valido come ObjectId, se la risposta dovesse essere affermativa, cerca l'User con il campo `_id` uguale al parametro della richiesta. Se l'User dovesse esistere allora lo invia sotto forma di JSON. Può inviare quattro status code diversi: **404** con message = “User not found”, **200** inviando l'user, **406** con message = “Invalid user id” e **500** con message = “Unable to find the user, try again”. Nella **figura 23** sottostante viene mostrato il codice di questa API.

```
// Api used to get an user
router.get('/getUser/:userId', async(req: Request, res: Response) => {

  try{

    // Get the database
    const db = await databaseService.getDb();

    // Set the parameter
    const userId: string = req.params.userId;

    // Check if the userId is valid
    if(ObjectId.isValid(userId)) {

      const user = await db?.collection("users").findOne( { _id: new ObjectId(userId) } ) as User | null;

      // Check if the user exists, then send it
      if(user) {

        res.status(200).json(user);

      } else {

        res.status(404).send("User not found");
      }

    } else {
      res.status(406).send("Invalid user id");
    }

  }catch(error: any) {

    res.status(500).send("Unable to find the user, try again");
  }

});
```

figura 23

modifyUsername

Quest'API è disponibile all'indirizzo **/user/modifyUsername**. È una richiesta PUT e richiede nel body della richiesta i seguenti parametri: **userId(string)** e **newUsername(string)**, mentre nell'header “**Authorization**” dell'**accessToken** che serve ad identificare l'utente e viene verificato tramite un middleware **verifyToken**. L'API provvede a controllare che l'userId è valido come ObjectId e che esista un User con il campo `_id` corrispondente e controlla se il nuovo username esiste già. In caso dovesse essere unico, procederà a cambiare il campo `name` con `newUsername`. Quest'API può inviare cinque status code: **200** con message = “Successfully updated”, **409** con message = “Username already exists”, **404** con message = “Cannot find user”, **406** con message = “Invalid user id” e **500** con message = “Unable to update the username, try again”. Nella **figura 24** sottostante viene mostrato il codice di questa API.

```

router.put('/modifyUsername', verifyToken, async (req: Request, res: Response) => {
  try {
    // Set the parameters
    const userId: string = req.body.userId;
    const newUsername: string = req.body.newUsername;

    // Check if the userId is valid
    if(ObjectId.isValid(userId)) {

      // Get the database
      const db = await databaseService.getDb();

      const user = await db?.collection("users").findOne({ _id: new ObjectId(userId) });

      // Check if the user exists
      if(user) {

        // Check if the username is already used, if not update it
        const existUsername = await db?.collection('users').findOne({ name: newUsername }) as User | null;
        if(!existUsername){

          await db?.collection('users').updateOne({ _id: new ObjectId(userId) }, { $set: { name: newUsername } });
          res.status(200).send('Successfully updated');

        } else {

          res.status(409).send('Username already exist');
        }
      } else {

        res.status(404).send('Cannot find user');
      }

    } else {
      res.status(406).send('Invalid user id');
    }
  } catch(error: any) {

    res.status(500).send("Unable to update the username, try again");
  }
});

```

figura 24

modifyPassword

Quest'API è disponibile all'indirizzo **/user/modifyPassword**. È una richiesta PUT e richiede nel body della richiesta i seguenti parametri: **userId(string)**, **oldPassword(string)** e **newPassword(string)**, mentre nell'header “**Authorization**” dell'**accessToken** che serve ad identificare l'utente e viene verificato tramite un middleware **verifyToken**. L'API provvede a controllare che l'userId è valido come ObjectId e che esista un User con il campo `_id` corrispondente e controlla se il campo password coincide con oldPassword e se la nuova password è valida chiamando il modulo **validatePassword**. In caso dovesse essere valida, procederà a criptare newPassword e a cambiare il campo password con hashedPassword(newPassword criptata). Quest'API può inviare cinque status code: **200** con message = “Successfully updated”, **401** con message = “Old password is incorrect”, **404** con message = “Cannot find user”, **406** che può avere diversi message = ““Invalid user id”/“The password must be 8 characters”

long"/"The password must contain a letter"/"The password must contain a number"/"The password must contain at least one special character between these: ? ! . _ - @ !" e **500** con message = "Unable to update the password, try again". Nella **figura 25** sottostante viene mostrato il codice di questa API.

```
// API used to modify the password
router.put('/modifyPassword', verifyToken, async (req: Request, res: Response) => {
  try {
    // Set the parameters
    const userId: string = req.body.userId;
    const oldPassword: string = req.body.oldPassword;
    const newPassword: string = req.body.newPassword;

    // Check if the userId is valid
    if(ObjectId.isValid(userId)) {

      // Get the database
      const db = await databaseService.getDb();

      // Check if the user exists
      const user = await db?.collection("users").findOne({ _id: new ObjectId(userId) }) as User | null;

      if(!user) {
        res.status(404).send('Cannot find the user');
      } else {

        // Check if the new password is valid
        if(validatePassword(newPassword, req, res)) {

          // Check if the old password is correct, if it is crypt the new password and update it
          if(await bcrypt.compare(oldPassword, user!.password)) {

            const salt = await bcrypt.genSalt();
            const hashedPassword = await bcrypt.hash(newPassword, salt);

            await db?.collection("users").updateOne({ _id: new ObjectId(userId) }, { $set: { password: hashedPassword } });

            res.status(200).send('Successfully updated');

          } else {
            res.status(401).send('Old password is incorrect');
          }
        }
      }
    } else {
      res.status(406).send('Invalid user id');
    }
  } catch(error: any) {
    res.status(500).send("Unable to update the password, try again");
  }
});
```

figura 25

deleteProfile

Quest'API è disponibile all'indirizzo **/user/deleteProfile/{userId}**. È una richiesta DELETE e richiede nell'URL della richiesta il parametro **userId(string)** mentre nell'header **"Authorization"** dell'**accessToken** che serve ad identificare l'utente e viene verificato tramite un middleware **verifyToken**. L'API provvede a controllare che l'userId è valido come ObjectId e che esista un User con il campo `_id`

corrispondente, se esiste provvede a eliminare l'account. Quest'API può inviare quattro status code: **200** con message = “Successfully deleted”, **404** con message = “Cannot find user”, **406** con message = “Invalid user id” e **500** con message = “Unable to delete the account, try again”. Nella **figura 26** sottostante viene mostrato il codice di questa API.

```
// API used to delete the account
router.delete("/deleteProfile/:userId", verifyToken, async (req: Request, res: Response) => {

  try {
    // Set the parameter
    const userId: string = req.params.userId;

    // Check if the userId is valid
    if(ObjectId.isValid(userId)) {

      // Get the database
      const db = await databaseService.getDb();

      // Check if the user exists, if it does delete it
      const user = await db?.collection("users").findOne( { _id: new ObjectId(userId) } ) as User | null;

      if(!user) {
        res.status(404).send('Cannot find user');
        return;
      } else {
        await db?.collection('users').deleteOne({ _id: new ObjectId(userId) });
        res.status(200).send('Successfully deleted');
      }
    } else {
      res.status(406).send('Invalid user id');
    }
  } catch(error: any) {
    res.status(500).send("Unable to delete the account, try again");
  }
});
```

figura 26

bookRoutes

getBook

Quest'API è disponibile all'indirizzo

/book/library/{libId}/getBook/{bookId}/id/{userId}. È una richiesta GET e richiede nell'URL della richiesta i seguenti parametri: **libId(string)**, **bookId(string)** e **userId(string)**. L'API controlla se l'userId preso dai parametri corrisponde a uno

User, in caso di risposta affermativa, prende il container delle librerie dell'User e cerca il libro specifico usando il modulo **getBookFromLibrary**. Quest'API può inviare tre status code: **404** con message = "User not found" o "Book not found", **200** inviando le informazioni del libro sotto forma di JSON e **500** con message = "Cannot find the book, try again". Nella **figura 27** sottostante viene mostrato il codice di questa API.

```
// API used to find a book
router.get("/library/:libId/getBook/:bookId/:userId", async (req: Request, res: Response) => {
  try {
    // Set the parameters
    const libId: string = req.params.libId;
    const bookId: string = req.params.bookId;
    const userId: string = req.params.userId;

    // Get the database
    const db = await databaseService.getDb();

    // Find the user
    const user = await db?.collection('users').findOne({ _id: new ObjectId(userId) }) as User | null;

    // Check if the user exists
    if(!user) {
      res.status(404).send("User not found");
    } else {
      // Get the library and the book
      const library: Library[] = user!.library;
      const book: BookTuple | null = await getBookfromLibrary(library, libId, bookId);

      // Check if the book exists
      if(book) {
        res.status(200).json(book);
      } else {
        res.status(404).send("Book not found");
      }
    }
  } catch(error: any) {
    res.status(500).send("Cannot find the book, try again");
  }
});
```

figura 27

getSpecLibrary

Quest'API è disponibile all'indirizzo `/book/getSpecLibrary/{libId}/id/{userId}`. È una richiesta GET e richiede nell'URL della richiesta i seguenti parametri: `libId(string)` e `userId(string)`. L'API controlla se l'userId preso dai parametri corrisponde a uno User, in caso di risposta affermativa, prende il container delle librerie dell'User e cerca la libreria corrispondente al libId preso dalla richiesta. Quest'API può inviare tre status code: **404** con message = "User not found" o "Library not found", **200** inviando le informazioni della libreria sotto forma di JSON e **500** con message = "Cannot find the library, try again". Nella **figura 28** sottostante viene mostrato il codice di questa API.

```
// API used to find a library
router.get("/getSpecLibrary/:libId/id/:userId", async (req: Request, res: Response) => {

  try {
    // Set the parameters
    const libId: string = req.params.libId;
    const userId: string = req.params.userId;

    // Get the database
    const db = await databaseService.getDb();

    // Find the user
    const user = await db?.collection('users').findOne({ _id: new ObjectId(userId) }) as User | null;

    // Check if the user exists
    if(!user) {
      res.status(404).send("User not found");
    } else {
      // Get the library
      const library = user!.library.find((lib: Library) => {
        if(lib.libId == libId) return lib;
        return null;
      }) as Library | null;

      // Check if the library exists
      if(library) {
        res.status(200).json(library);
      } else {
        res.status(404).send("Library not found");
      }
    }
  } catch(error: any) {
    res.status(500).send("Cannot find the library, try again");
  }
});
```

figura 28

getLibraries

Quest'API è disponibile all'indirizzo **/book/getLibraries/{userId}**. È una richiesta GET e richiede nell'URL della richiesta il parametro: **userId(string)**. L'API controlla se l'userId preso dai parametri corrisponde a uno User, in caso di risposta affermativa, prende il container delle librerie dell'User e le invia nella risposta.

Quest'API può inviare tre status code: **404** con message = "User not found", **200** inviando le informazioni del container delle librerie sotto forma di JSON e **500** con message = "Cannot find the libraries, try again". Nella **figura 29** sottostante viene mostrato il codice di questa API.

```
// API used to find all the libraries
router.get("/getLibraries/:userId", async (req: Request, res: Response) => {

  try {

    // Set the parameter
    const userId: string = req.params.userId;

    // Get the database
    const db = await databaseService.getDb();

    // Find the user
    const user = await db?.collection("users").findOne({ _id: new ObjectId(userId) }) as User | null;

    // Check if the user exists
    if(!user) {

      res.status(404).send("User not found");

    } else {

      // Get the libraries
      const libraries: Library[] = user!.library;

      res.status(200).json(libraries);
    }

  } catch(error: any) {

    res.status(500).send("Cannot find the libraries, try again");
  }
});

});
```

figura 29

modifyPages

Quest'API è disponibile all'indirizzo **/book/modifyPages**. È una richiesta PUT e richiede nel body della richiesta i parametri: **bookId(string)**, **libId(string)**,

pages(*number*) e **userId**(*string*). L'API controlla se l'userId preso dai parametri corrisponde a uno User, in caso di risposta affermativa, prende il container delle librerie dell'User, cerca la libreria con libId uguale a quello della richiesta, in questa libreria cerca il libro con bookId corrispondente a quello della richiesta e successivamente modifica le pagine lette di quel libro. Quest'API può inviare tre status code: **404** con message = "User not found" o "The library where you want to modify the book doesn't exist", **200** con message = "Number of pages read updated" e **500** con message = "Cannot modify the number of pages". Nella **figura 30** sottostante viene mostrato il codice di questa API.

```
//API used to modify the number of pages read
router.put("/modifyPages", async(req: Request, res: Response) => {

  try {
    // Set the parameters
    const bookId: string = req.body.bookId;
    const libId: string = req.body.libId;
    const pages: number = req.body.pages;
    const userId: string = req.body.userId;

    // Get the database
    const db = await databaseService.getDb();

    // Find the user
    const user = await db?.collection('users').findOne({ _id: new ObjectId(userId) }) as User | null;

    // Check if the user exists
    if(!user) {
      res.status(404).send('User not found');
    } else {
      // Get the libraries' container and check if the library exists, then update the number of pages
      const library: Library[] = user!.library;
      let libraryPresence: boolean = false;

      library.forEach((lib) => {
        if(lib.libId == libId) lib.books.forEach((book) => {
          libraryPresence = true;
          if(book.bookId == bookId) book.pagesRead = pages;
        });
      });

      // Update the number of pages if the library exists
      if(libraryPresence) {
        await db?.collection('users').updateOne( { _id: new ObjectId(userId) }, { $set: { library: library } } );
        res.status(200).send("Number of pages read updated");
      } else {
        res.status(404).send("The library where you want to modify the book doesn't exist");
      }
    }
  } catch(error: any) {
    res.status(500).send("Cannot modify the number of pages");
  }
});
```

figura 30

createLibrary

Quest'API è disponibile all'indirizzo **/book/createLibrary**. È una richiesta POST e richiede nel body della richiesta i parametri: **libName(string)**, **libId(string)** e **userId(string)**. L'API controlla se l'userId preso dai parametri corrisponde a uno User, in caso di risposta affermativa, prende il container delle librerie dell'User, crea la libreria nuova(**newLibrary**) utilizzando i parametri libName e libId, la salva nel campo library dell'User e, successivamente, aggiorna il campo library dell'User nel database. Quest'API può inviare tre status code: **404** con message = "User not found", **200** con message = "Library “**libName**” created" e **500** con message = "Failed to create the library, try again". Nella **figura 31** sottostante viene mostrato il codice di questa API.

```
// API used to create a library
router.post("/createLibrary", async (req: Request, res: Response) => {
  try {
    // Set the parameters
    const libName: string = req.body.libName;
    const libId: string = req.body.libId;
    const userId: string = req.body.userId;

    // Get the database
    const db = await databaseService.getDb();

    // Find the user
    const user = await db?.collection("users").findOne({ _id: new ObjectId(userId) }) as User | null;

    // Check if the user exists
    if(user) {
      // Get the the container of the libraries
      const library: Library[] = user.library;

      // Creating the new library
      const newLibrary: Library = { libName: libName as string, libId: libId as string, books: [] as BookTuple[] };

      // Add the new library to the container
      library.push(newLibrary);

      await db?.collection("users").updateOne( { _id: new ObjectId(userId) }, { $set: { library: library } } );

      res.status(200).send(`Library "${libName}" created`);

    } else {
      res.status(404).send("User not found");
    }
  } catch(error: any) {
    res.status(500).send("Failed to create the library, try again");
  }
});
```

figura 31

addBook

Quest'API è disponibile all'indirizzo **/book/addBook**. È una richiesta POST e richiede nel body della richiesta i parametri: **bookId(string)**, **libId(string)** e **userId(string)**. L'API controlla se l'userId preso dai parametri corrisponde a uno User, in caso di risposta affermativa, prende il container delle librerie dell'User, utilizzando il parametro libId trova la libreria in cui dover salvare il libro creato con il parametro bookId e inizializzando le pagesRead a 0(**newBook**) e, successivamente, aggiorna il campo library dell'User nel database. Quest'API può inviare tre status code: **404** con message = "User not found" o "The library where you want to add the book doesn't exist", **200** con message = "Book added with success" e **500** con message = "Cannot add the book". Nella **figura 32** sottostante viene mostrato il codice di questa API.

```

// API used to add a book to a library
router.post("/addBook", async(req: Request, res: Response) => {
  try{
    // Set the parameters
    const bookId: string = req.body.bookId;
    const libId: string = req.body.libId;
    const userId: string = req.body.userId;

    // Create the new book
    const newBook: BookTuple = { bookId: bookId, pagesRead: 0 };

    // Get the database
    const db = await databaseService.getDb();

    // Find the user
    const user = await db?.collection("users").findOne({ _id: new ObjectId(userId) }) as User | null;

    // Check if the user exists
    if(user) {

      // Get the libraries' container and check if the library exists
      const library: Library[] = user.library;
      let libraryPresence: boolean = false;

      library.forEach((lib) => {
        if(lib.libId === libId) {
          libraryPresence = true;
          lib.books.push(newBook);
        }
      });

      // Add the book if the library exists
      if(libraryPresence) {

        await db?.collection("users").updateOne({ _id: new ObjectId(userId) }, { $set: { library: library } });

        res.status(200).send("Book added with success");

      } else {

        res.status(404).send("The library where you want to add the book doesn't exist");
      }

    } else {

      res.status(404).send("User not found");
    }

  } catch(error: any) {

    res.status(500).send("Cannot add the book");
  }
});

```

figura 32

deleteBook

Quest'API è disponibile all'indirizzo

/book/library/{libId}/deleteBook/{bookId}/id/{userId}. È una richiesta

DELETE e richiede nell'URL della richiesta il parametro: **libId(string)**, **bookId(string)** e **userId(string)**. L'API controlla se l'userId preso dai parametri corrisponde a uno User, in caso di risposta affermativa, prende il container delle librerie dell'User, cerca la libreria dalla quale eliminare il libro tramite il parametro libId e cercando il libro corrispondente confrontando il bookId con i libri della

libreria trovata, lo elimina e aggiorna il campo library dell'User nel database. Quest'API può inviare tre status code: **404** con message = “User not found” o “The library where you want to delete the book doesn't exist”, **200** con message = “Book deleted with success” e **500** con message = “Cannot delete the book”. Nella **figura 33** sottostante viene mostrato il codice di questa API.

```
// API used to delete a book from a library
router.delete("/library/:libId/deleteBook/:bookId/id/:userId", async (req: Request, res: Response) => {
  try {
    // Set the parameters
    const libId: string = req.params.libId;
    const bookId: string = req.params.bookId;
    const userId: string = req.params.userId;

    // Get the database
    const db = await databaseService.getDb();

    // Find the user
    const user = await db?.collection("users").findOne({ _id: new ObjectId(userId) }) as User | null;

    // Check if the user exists
    if(user) {
      // Get the libraries' container and check if the library exists
      const library: Library[] = user.library;
      let libraryPresence: boolean = false;

      library.forEach((lib) => {
        if(lib.libId === libId) {
          libraryPresence = true;
          lib.books = lib.books.filter((book) => book.bookId !== bookId) as BookTuple[];
        }
      });

      // Delete the book if the library exists
      if(libraryPresence) {
        await db?.collection("users").updateOne( { _id: new ObjectId(userId) }, { $set: { library: library } });
        res.status(200).send("Book deleted with success");
      } else {
        res.status(404).send("The library where you want to delete the book doesn't exist");
      }
    } else {
      res.status(404).send("User not found");
    }
  } catch(error: any) {
    res.status(500).send("Cannot delete the book");
  }
});
```

figura 33

deleteLibrary

Quest'API è disponibile all'indirizzo **/book/deleteLibrary/{libId}/id/{userId}**. È una richiesta DELETE e richiede nell'URL della richiesta il parametro: **libId(string)** e **userId(string)**. L'API controlla se l'userId preso dai parametri corrisponde a uno User, in caso di risposta affermativa, prende il container delle librerie dell'User,

cerca la libreria da eliminare utilizzando il libId, una volta trovata la elimina e aggiorna il campo library dell'User nel database. Quest'API può inviare tre status code: **404** con message = “User not found” o “The library that you want to delete doesn't exist”, **200** con message = “Library “{libId}” deleted with success” e **500** con message = “Cannot delete the library”. Nella **figura 34** sottostante viene mostrato il codice di questa API.

```
// API used to delete a library
router.delete("/deleteLibrary/:libId/:userId", async (req: Request, res: Response) => {
  try {
    // Set the parameters
    const libId: string = req.params.libId;
    const userId: string = req.params.userId;

    // Get the database
    const db = await databaseService.getDb();

    // Find the user
    const user = await db?.collection("users").findOne({ _id: new ObjectId(userId) }) as User | null;

    // Check if the user exists
    if(user) {
      // Get the libraries' container and if the libraries exists, delete it
      const library: Library[] = user.library;

      const updateLibrary = library.filter((lib) => {
        return lib.libId !== libId;
      }) as Library[];

      // Delete the library if it's been deleted a library
      if(library.length !== updateLibrary.length) {
        await db?.collection("users").updateOne({ _id: new ObjectId(userId) }, { $set: { library: updateLibrary } });

        res.status(200).send(`Library "${libId}" deleted with success`);

      } else {
        res.status(404).send("The library that you want to delete doesn't exist");
      }
    } else {
      res.status(404).send("User not found");
    }
  } catch(error: any) {
    res.status(500).send("Cannot delete the library");
  }
});
```

figura 34

5.Documentazione delle API

Le API elencate precedentemente sono state documentate utilizzando **Swagger** e il suo modulo **swagger-ui-express**. Abbiamo diviso le API nei due tag **User** e **Books**.

La documentazione completa si può trovare anche al link <https://backend-production-7b98.up.railway.app/api-docs> oppure <http://localhost:{PORT}/api-docs> nel caso che la web app non dovesse essere online e dovreste avviare il server in locale.

Di seguito riportiamo degli esempi dell'interfaccia **Swagger UI** di quattro API di richieste differenti: **GET**(figura 35), **POST**(figura 36), **PUT**(figura 37) e **DELETE**(figura 38).

GET /user/getUser/{userId} Retrieve a user by ID

Retrieve a user by their unique ID from the database.

Parameters

Name	Description
userId * required	The unique ID of the user string (path)
userId	<input type="text"/>

Responses

Code	Description	Links
200	A user object	No links
	Media type application/json	
	Controls Accept header.	
	Example Value Schema	
	<pre>{ "id": "string", "name": "string", "email": "string" }</pre>	
404	User not found	No links
406	Invalid user id	No links
500	Unable to find the user, try again	No links

figura 35

POST /user/register Register a user and encrypt the password

Register a new user and encrypt their password.

Parameters

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "name": "string",
  "email": "string",
  "password": "string",
  "userId": "string"
}
```

Responses

Code	Description	Links
201	User created successfully	No links
406	Possible reason: Invalid email or user id; The password must be 8 char long; The password must contain a letter; The password must contain a number; The password must contain at least one special char between these: ? ! . _ - @	No links
409	Email or Username already exists	No links
500	Unable to register the user, try again	No links

figura 36

PUT /library/modifyPages Modify the number of pages read

Modify the number of pages read for a specific book in a library.

Parameters

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "bookId": "string",
  "libId": "string",
  "pages": 0,
  "userId": "string"
}
```

Responses

Code	Description	Links
200	Number of pages read updated	No links
404	User or library not found	No links
500	Cannot modify the number of pages	No links

figura 37

The screenshot shows a REST API endpoint for deleting a book from a library. The URL is `/library/{libId}/deleteBook/{bookId}/id/{userId}`. The description is "Delete a book from a specific library by its unique ID".

Parameters

Name	Description
<code>libId</code> * required	The unique ID of the library string (path) libId
<code>bookId</code> * required	The unique ID of the book string (path) bookId
<code>userId</code> * required	The unique ID of the user string (path) userId

Responses

Code	Description	Links
200	Book deleted successfully	No links
404	User or library not found	No links
500	Cannot delete the book	No links

figura 38

6.Implementazione del FrontEnd

Per l'implementazione del front-end, ovvero l'interfaccia grafica per permettere all'utente di usufruire al meglio delle API del backend, è stato utilizzato **React** ed il server di sviluppo locale **ViteJS**.

La schermata di login (**figura 39**) dà la possibilità all'utente di accedere con un account precedentemente registrato tramite email e password e la possibilità di navigare alla schermata di registrazione (**figura 40**) per poi, tramite username, email e password creare un nuovo account.

Una volta eseguito l'accesso l'utente si ritroverà nella schermata home (**figura 41**) della webapp, qui vediamo diversi componenti come la navbar nell'header della pagina che ci accompagnerà durante la visita dell'intero sito.

Oltre alla navbar troviamo una zona dedicata al libro del giorno, seguito poi da una serie di collezioni di libri filtrati per argomento o per tendenze.

Cliccando sul titolo di una di queste categorie apparirà all'utente una nuova pagina (**FullPage - figura 42**) con la lista ordinata di tutti i risultati, con la possibilità, a piè di pagina, di navigare tra le varie pagine di risultati se presenti. Questa pagina apparirà all'utente anche nel caso di una ricerca e cliccando sul titolo di una libreria personale (entrambi i casi verranno analizzati in seguito).

Cliccando sulla copertina di un libro, in qualsiasi pagina della webapp in cui è presente, apparirà all'utente la schermata contenente tutti i dettagli riguardanti quel libro nonché un bottone che aprirà un menù a tendina con la lista di tutte le librerie personali dell'utente (**figura 43**). Cliccando sul nome di una libreria nel menù il libro verrà aggiunto o rimosso da essa a seconda se il libro è già contenuto in essa o meno. Oltre a questi componenti a piè di pagina troviamo una barra (inizialmente grigia) con la possibilità di modificare il numero di pagine, scrivendo direttamente il numero nel campo numerico o modificandolo con i bottoni ai lati, che l'utente ha letto di quel libro. Se esso, al momento della modifica delle pagine lette, non è presente in alcuna libreria, verrà automaticamente aggiunto alla libreria base ovvero 'Your books'.

Come accennato in precedenza la navbar ci ha accompagnato durante questo percorso, analizzandola troviamo 3 elementi al suo interno, l'immagine profilo dell'utente, la barra di ricerca ed infine il logo della webapp. Analizzandoli in senso contrario troviamo che cliccando il logo sulla destra l'utente verrà riportato alla schermata home. Attraverso la barra di ricerca l'utente potrà navigare nel database esterno dei libri e se una ricerca produrrà risultati, come accennato in precedenza, questi verranno mostrati tramite una FullPage caratteristicamente identica a quella già analizzata, se la ricerca non dovesse portare a risultati apparirà all'utente la scritta "no result found for {stringa cercata}".

L'ultimo elemento della navbar, ovvero l'immagine profilo, funge da menu all'utente(**figura 44**), infatti se cliccata si aprirà un menu laterale con la possibilità di navigare tra schermata home, my library, user settings e la possibilità di eseguire il logout.

La schermata My library (**figura 45**) conterrà tutte le librerie dell'utente più la possibilità, tramite le icone in alto a destra, di aggiungere o rimuovere librerie, l'utente tuttavia non avrà la possibilità di rimuovere la libreria di base Your books per garantire alla webapp la presenza di una libreria in cui poter inserire i libri

desiderati. Come nominato precedentemente in questa schermata sarà possibile cliccare sul titolo delle librerie, portando così l'utente alla fullpage con tutti i libri contenuti nella libreria selezionata e la possibilità di cliccare direttamente sui libri visualizzati per aprirne la pagina dei dettagli.

La schermata User settings (**figura 46**) permetterà all'utente di modificare username e password e di eliminare l'account, per eseguire queste operazioni basterà cliccare sui bottoni corrispondenti e seguire le istruzioni dei relativi pop-up che appariranno all'utente. Nel caso dell'eliminazione dell'account l'utente verrà riportato alla schermata di login.

Infine se l'utente cliccherà sull'opzione di logout nel menù laterale, esso verrà riportato alla schermata di login e sarà per lui nuovamente necessario accedere per raggiungere il resto della webapp.

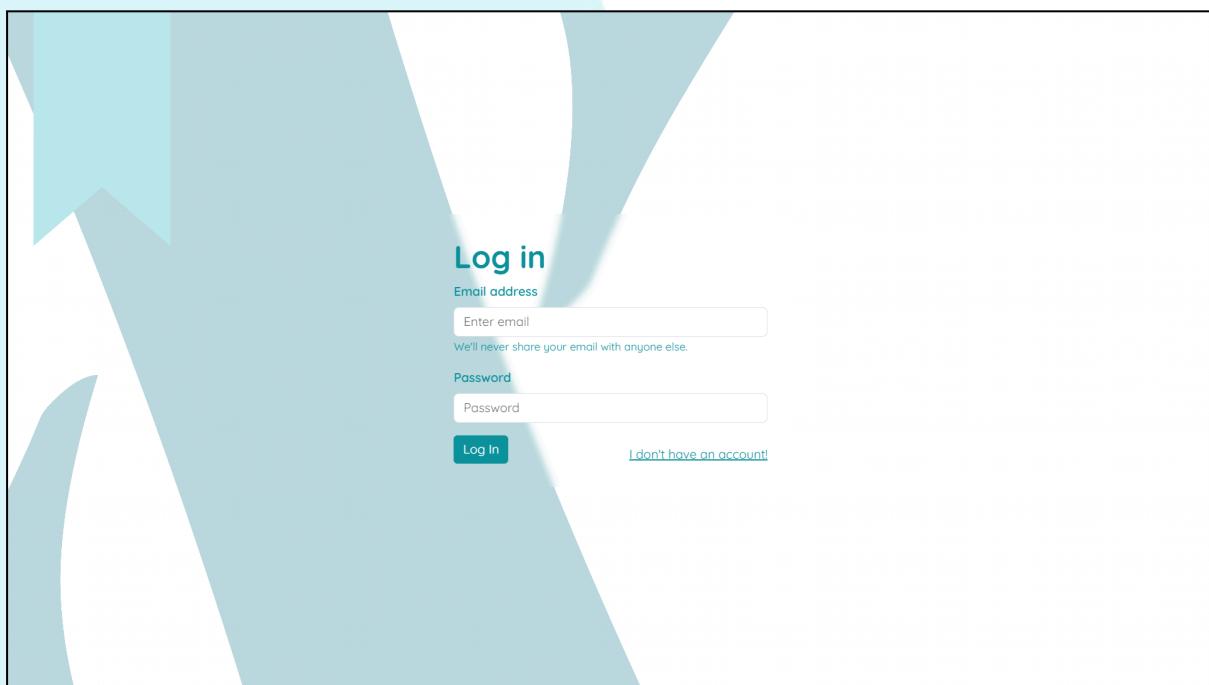


figura 39

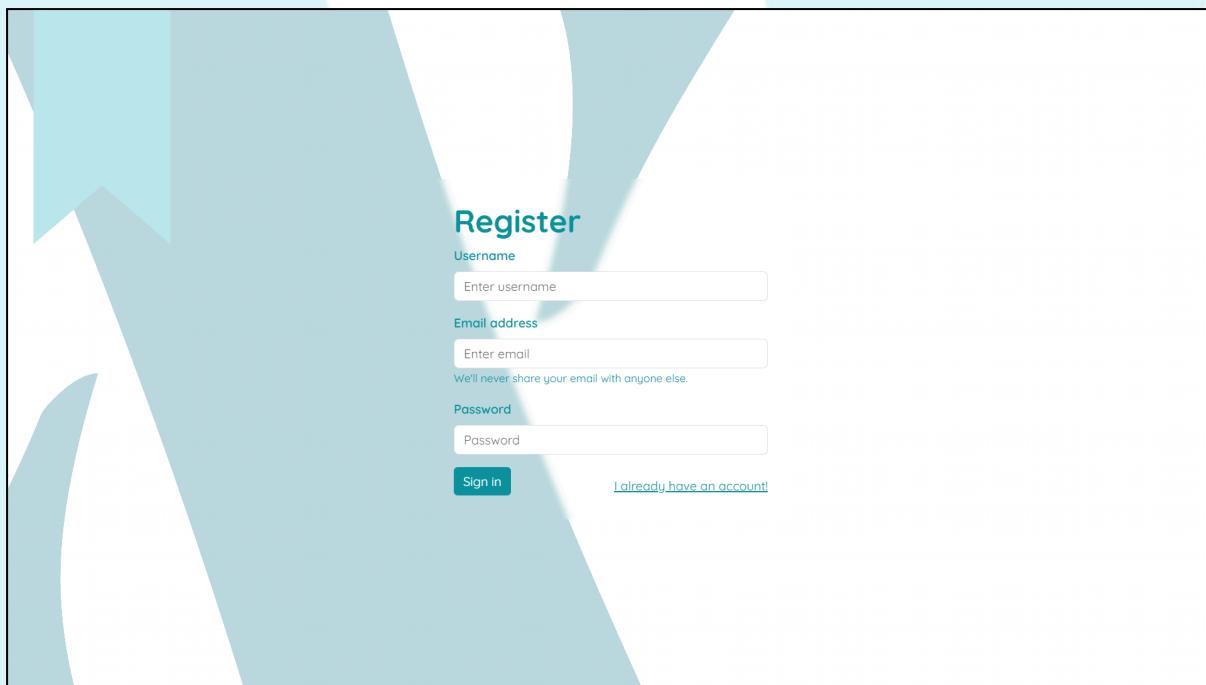


figura 40

The page features a large image of the book cover for "Twilight" by Stephenie Meyer. The cover shows two hands holding a red apple. The title "twilight" is written vertically on the left side of the cover, and the author's name "STEPHENIE MEYER THE INTERNATIONAL BESTSELLER" is at the bottom. Below the image, the text "Book of the day!" is displayed in white. A short summary of the book is provided.

Book of the day!

Twilight
About three things I was absolutely positive. First, Edward was a vampire. Second, there was a part of him—and I didn't know how dominant that part might be—that thirsted for my blood. And third, I was unconditionally and irrevocably in love with him. In the first book of the Twilight Saga, internationally bestselling author Stephenie Meyer introduces Bella Swan and Edward Cullen, a pair of star-crossed lovers whose forbidden relationship ripens against the backdrop of small-town suspicion and a mysterious coven of vampires. This is a love story with bite.

Fan favorites

figura 41

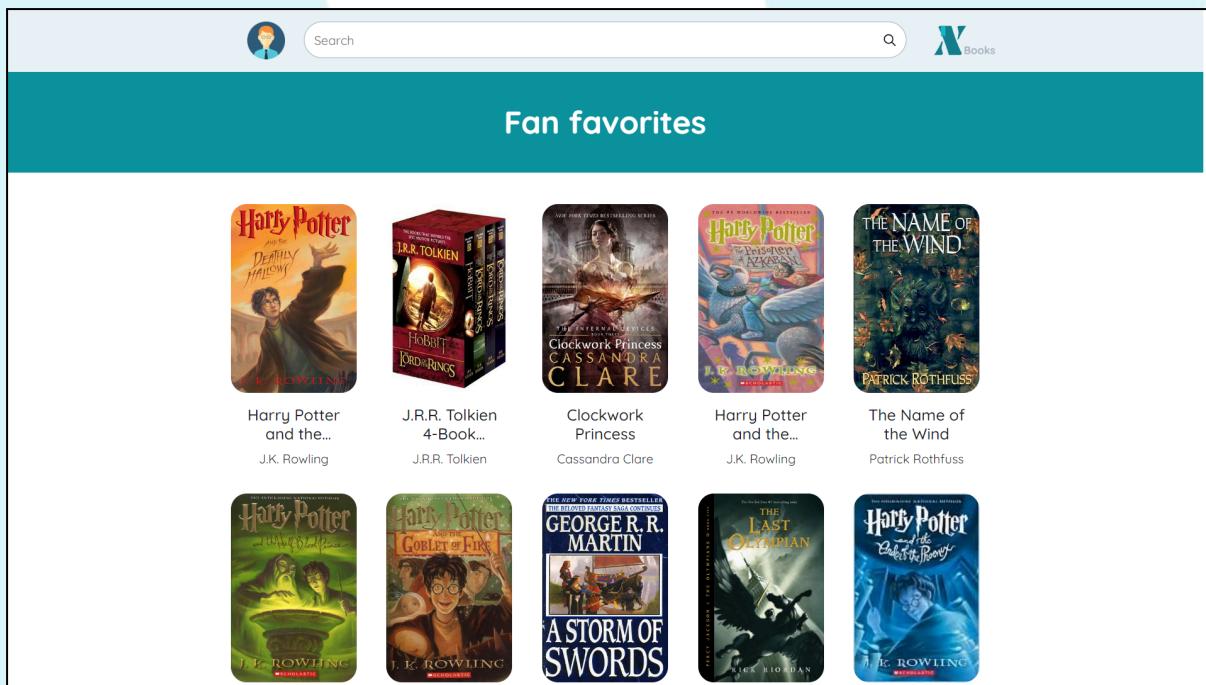


figura 42

Harry Potter and the Deathly Hallows

Description:
Harry Potter is leaving Privet Drive for the last time. But as he climbs into the sidecar of Hagrid's motorbike and they take to the skies, he knows Lord Voldemort and the Death Eaters will not be far behind. The protective charm that has kept him safe until now is broken. But the Dark Lord is breathing fear into everything he loves. And he knows he can't keep hiding. To stop Voldemort, Harry knows he must find the remaining Horcruxes and destroy them. He will have to face his enemy in one final battle. --jkrowling.com

Authors
J.K. Rowling

Genres
Fantasy, Young Adult, Fiction

Libraries ▾

- [Your books](#)
- [Preferiti](#)

- 233 +

of 759 pages

figura 43

The screenshot shows the XBooks application interface. On the left, there is a sidebar with a user profile picture of a person with orange hair, labeled "PuffyEnri4" and "enrico.bolognesi004@gmail.com". Below the profile are navigation links: "Home" (selected), "My Library", "User settings", and "Log Out". The main content area has a dark teal header with the text "Book of the day!". Below it, a book summary for "Twilight" by Stephenie Meyer is displayed, mentioning the vampire romance between Edward Cullen and Bella Swan. A large image of the book cover is shown below the summary. At the bottom of the main content area, there is a section titled "In favorites" with a circular refresh icon. Below this, there are small thumbnail images of several books: "Harry Potter", "The Name of the Wind", "Harry Potter and the Deathly Hallows", "Harry Potter and the Half-Blood Prince", and "Clockwork Princess".

figura 44

The screenshot shows the XBooks application interface. At the top, there is a search bar and a "Books" logo. Below the search bar, there is a section titled "Preferiti" with a circular refresh icon. A message "This list is empty!" is displayed. Below this, there is a section titled "Your books" with a circular refresh icon. There are five book entries, each with a thumbnail image and the title, author, and series information:

- The Curious Incident of...** by Mark Haddon
- The Name of the Wind** by Patrick Rothfuss
- Harry Potter and the...** by J.K. Rowling
- Harry Potter and the Hal...** by J.K. Rowling
- Clockwork Princess** by Cassandra Clare

figura 45

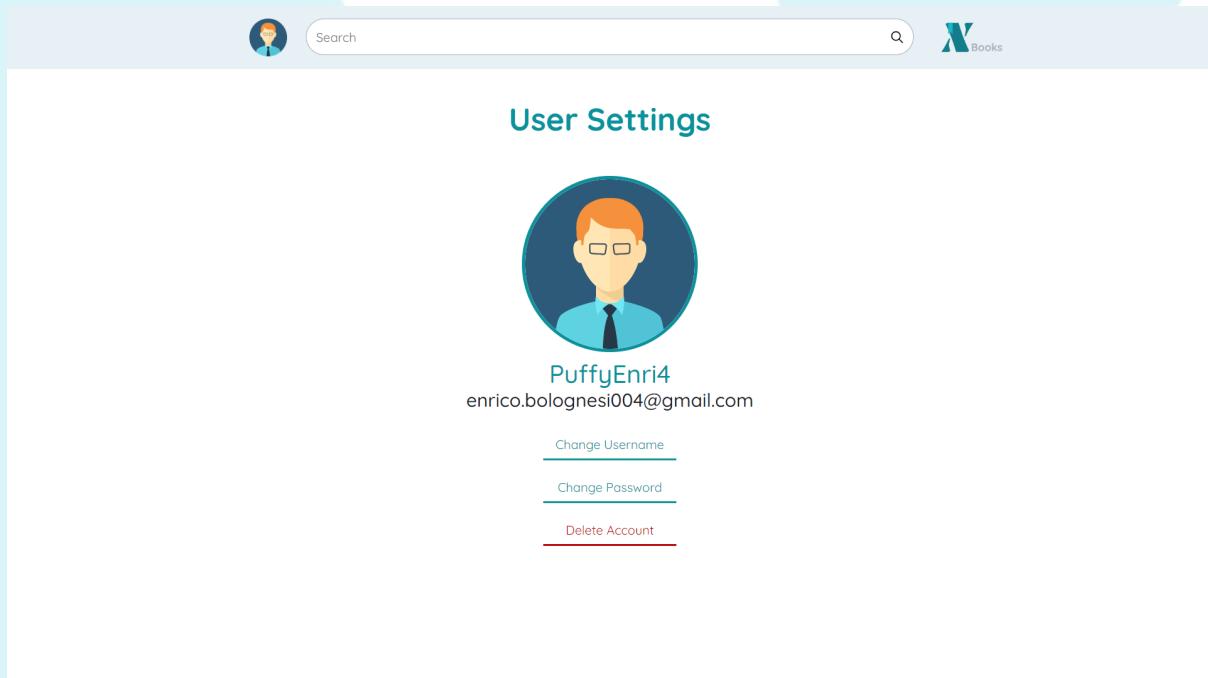


figura 46

7.GitHub repository e deployment

Il progetto è presente su GitHub al link: <https://github.com/G19-IS2023>. Si trovano tre repository: Backend, Frontend e Deliverable.

Per il deployment ci siamo affidati a [Railway](#) per quanto riguarda il backend e [Netlify](#) per il frontend. A ogni commit nelle due repository, il servizio della repository corrispondente creerà la build con le nuove modifiche e effettuerà il deploy.

Il BackEnd lo potete trovare su questo link:

- <https://backend-production-7b98.up.railway.app>

Il FrontEnd lo potete trovare su quest'altro link:

- <https://ephemeral-lily-b2cda5.netlify.app>

Nelle repository del BackEnd e del FrontEnd troverete i rispettivi file **README.md** con le istruzioni per avviare il progetto localmente.

8. Testing delle API

Per eseguire il testing delle API abbiamo utilizzato le librerie **Jest** e **supertest**. Jest permette di gestire i casi di test grazie alla divisione in suite, esegue i test e riporta i risultati nella cartella **/coverage** di cui troverete l'interfaccia nella **figura 47** seguente. supertest invece permette di inviare le richieste agli endpoint. I file di test possono essere trovati nella cartella **/scr/test** in cui troverete un'altra cartella **/routeTests** in cui sono presenti i file contenente il codice dei test che coprono tutte le API implementate nel BackEnd.

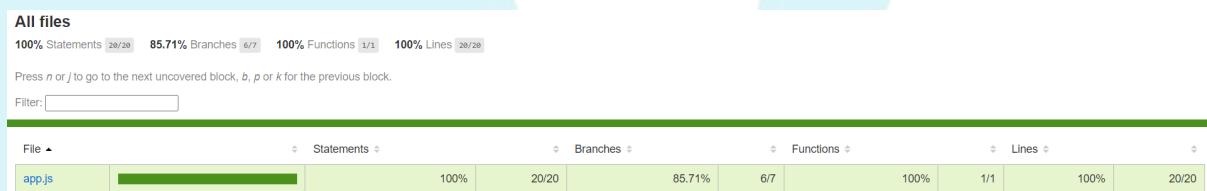


figura 47

```
describe('User Routes', () => {
  describe('GET user/getUser/:userId', () => {
    it('should retrieve user details if the user exists and send status code 200', async () => {
      const response = await request(app)
        .get(`/user/getUser/${existingUserId}`)
      expect(response.statusCode).toBe(200);
    });
    it('should send status code 404 if user does not exists', async () => {
      const response = await request(app)
        .get(`/user/getUser/${notExistingId}`);
      expect(response.statusCode).toBe(404);
      expect(response.text).toEqual("User not found");
    });
    it('should send status code 406 if userId is not valid', async () => {
      const response = await request(app)
        .get("/user/getUser/invalidUserId")
      expect(response.statusCode).toBe(406);
      expect(response.text).toEqual("Invalid user id");
    });
  });
});
```

figura 48 - Esempio caso di test dell'API getUser