

Project Report

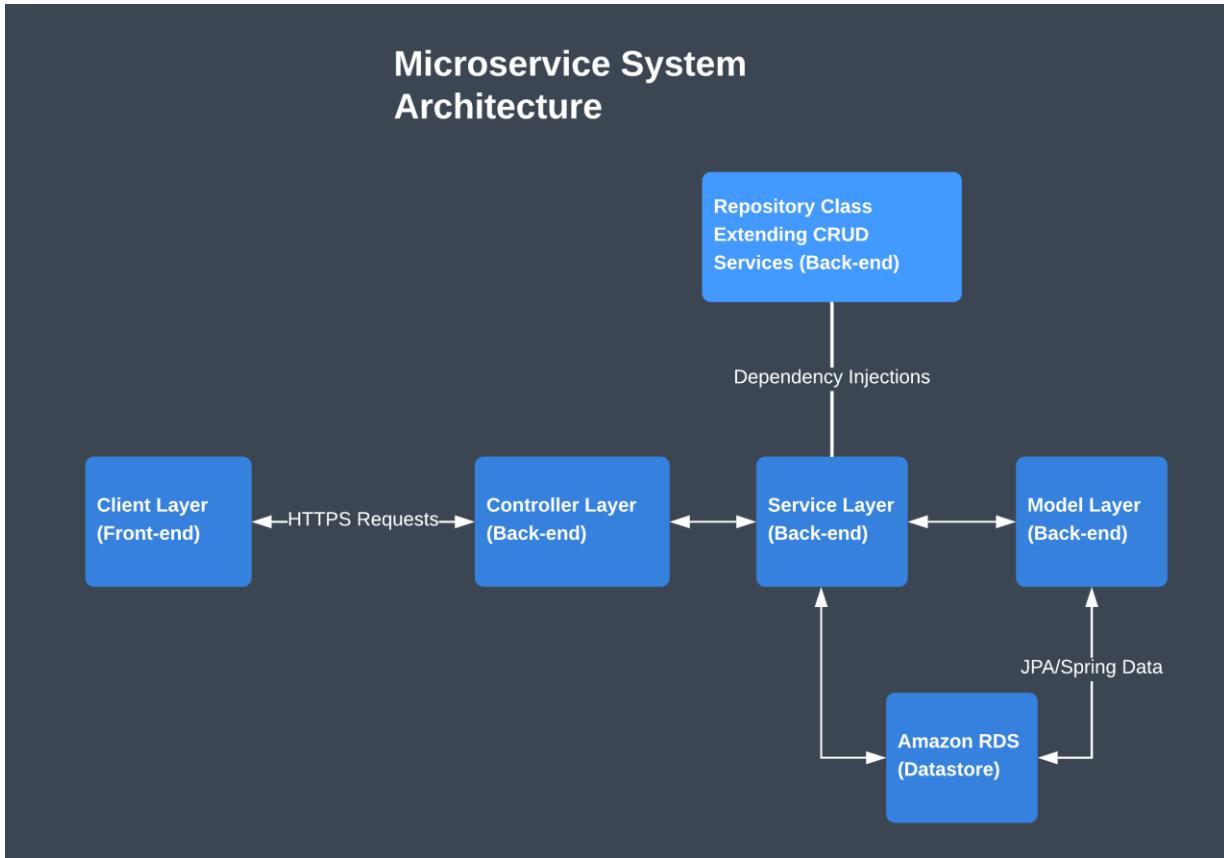
Vision Statement:

We believe that our product is valuable as it demonstrates the team's ability to create a product as requested by a client or product owner. Although we were initially very inexperienced with the language and framework that was required, our team overcame challenges together and was able to quickly learn new technologies such as AWS cloud deployment, CircleCI and Docker. Our work shows that every person on the team is capable, resilient, determined and will address the needs of a customer quickly and effectively. Having documented all our progress, meetings, and retrospectives for each sprint, we are able to illustrate how our initial setbacks and slow progress was only temporary, as our velocity for each sprint increased which shows our adaptability and rapid improvement in skills over time.

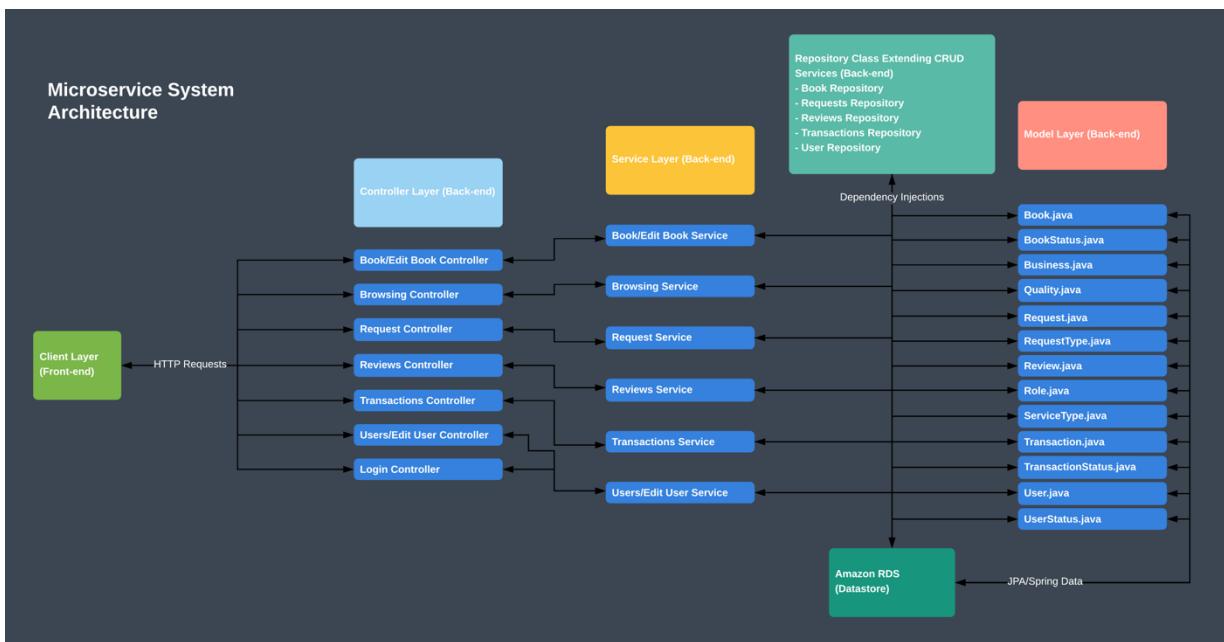
While our product is not suitable for full-scale commercial use, it has the groundwork and potential to be used as a fully functioning e-commerce website for any business that is interested in buying, sharing, and selling books for users and businesses, while admins can oversee and manage all business operations. Given more time, we believe we could continue to adjust and improve our application as required and continue to deliver on all promised functionality as we have done.

System Architecture/Design

Base Diagram:



Full Diagram: (for better quality, this image can be found in the Scrum folder in the root directory)



Refactoring Report

In the original base code provided to us, the starting microservice given was the login microservice, which included a user class and methods for saving new users to a user repository. Due to our unfamiliarity and lack of knowledge in implementing Spring Boot as a back-end framework, all our new classes and services were added to this login microservice, including the books, browsing, transactions, requests, and reviews microservices. This monolithic architecture did have some advantages, such as easy and fast development, testing and tracing, and deployment. These factors were especially helpful as it allowed us to quickly build our starting application with some basic features while we were still learning how to write code for Spring Boot.

However, as our experience and knowledge became more consolidated over the course of the project, we found that there were several disadvantages to our monolithic architecture, such as dependencies between services; for example, if there was an error or bug with compiling any classes related to the users microservice, this would create a ripple effect and inhibit us from running all our other services. This slowed down our development speed and make scaling our services more difficult as most services were dependent on others, making our code unreliable at times and inflexible. To solve these issues, we decided to refactor our design pattern to follow a microservice architecture.

To refactor our code, we initially planned to create a common library or service, such as the new independent users microservice, and then have all other microservices include it as a dependency, making use of asynchronous communication. However, we believed that this would not help us solve all our issues associated with the monolithic architecture and decided to create each microservice so that they could be developed and deployed independently, without being dependent on other services. This allowed us to make changes and updates to scale each service when required and prevented any issues with microservices from causing other services to crash, making our code more resilient.

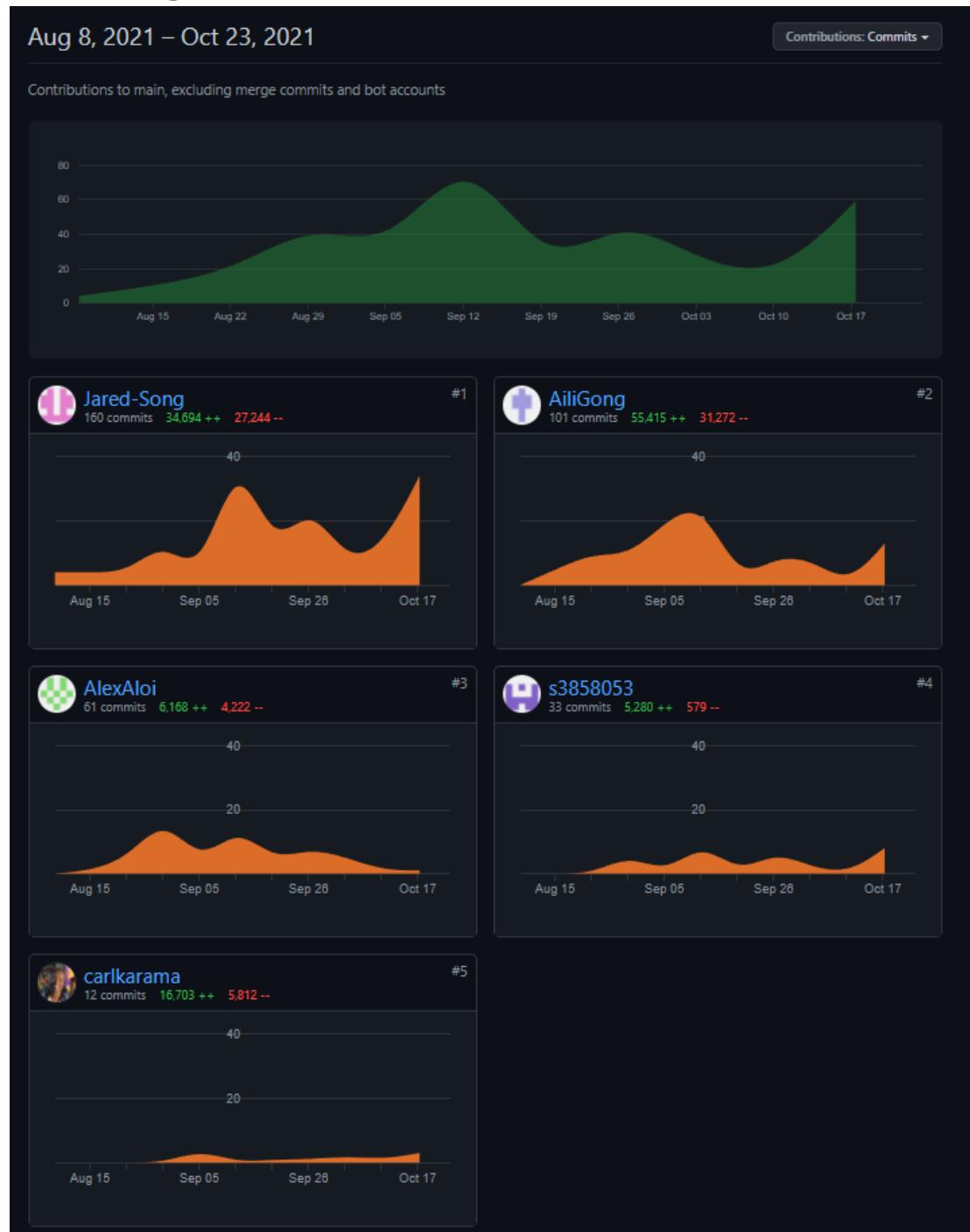
Currently, each microservice is responsible for handling queries only related to their microservice, for example the users microservice is required to register a new user, edit user information, or delete users, while the reviews microservice is required to leave reviews and ratings for users and books. If any microservice is not deployed and run, the other services will still function as required, but the terminated service will not be able to be called or used. Each service is also connected to the same datastore we set up on Amazon RDS, and thus any changes each service makes will be shared across all services.

The microservices we have implemented are as follows:

- Books
- Browsing
- Login
- Reviews
- Requests
- Transactions
- Users

A microservice was created for incentives, however as this was not an original requirement for our product, the directory only contains our models for our possible future implementation of this new service.

Git Organisation





Scrum Process:

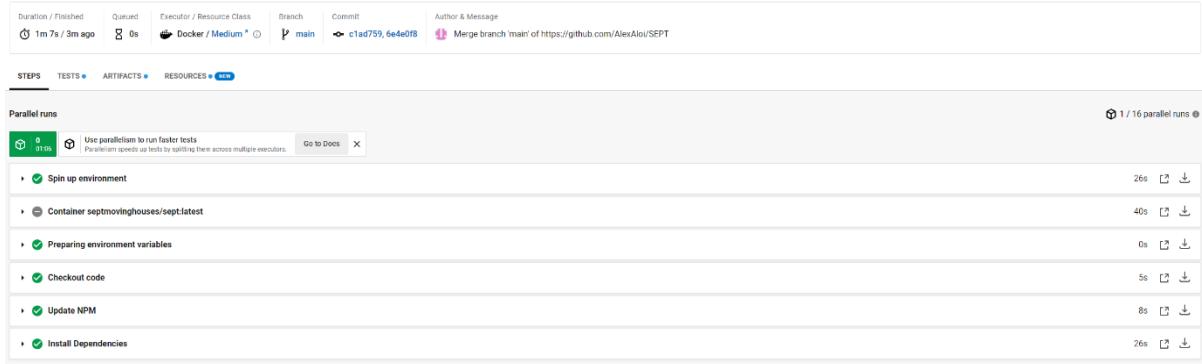
Scrum master: Jared

Our group met at least twice every week typically on Tuesdays and Fridays. Each meeting often went for two hours, while additional sporadic meetings were held each week depending on progress, bugs, errors, or issues with code. If a major issue was present, or there were tasks still incomplete close to deadlines, then meetings would be held to solve these issues immediately, and in some cases we would have more than four meetings a week to deal with these problems as quickly as possible. However, when the velocity of the team was higher than expected, and the group was outperforming predictions, then we followed a standard number of two meetings a week.

Our communication was primarily through Microsoft Teams, where messages were typically sent each day for updates on tasks, progress, or questions about each member's tasks. For tasks that would require two or three members to work together and solve, smaller meetings were held to address these concerns, as often meeting with the entire group was not necessary, and results would be shared at the next scheduled group meeting regardless. Every group member attended all team meetings, with some apologies due to health issues that were communicated ahead of time. Overall, the scrum master is extremely content and happy with the scrum process of the project and group.

Deployment Setup:

After pushing to GitHub, CircleCI will run the front-end tests.



The screenshot shows a CircleCI build interface. At the top, it displays basic build information: Duration / Finished, Queued (0s), Executor / Resource Class (Docker / Medium), Branch (main), Commit (c1ad759, 6e4e0f8), and Author & Message (Merge branch 'main' of https://github.com/AlexAlo/SEPT). Below this, there are tabs for STEPS, TESTS, ARTIFACTS, and RESOURCES. The STEPS tab is selected, showing a list of parallel runs. Each run consists of several steps: Spin up environment, Container septmovinghouses/sept:latest, Preparing environment variables, Checkout code, Update NPM, and Install Dependencies. The total duration for all parallel runs is 1m 7s.

This also ensures the latest Docker container is working.

The next part in the deployment is a batch file, that packages and tests all backend micro-services, and runs them.

```
echo off
for /d %%i in (%~dp0\*) do (
    cd %%i
    echo %%i
    if exist %%~fi\target\*.jar start cmd /C "mvnw package && cd %%~fi\target\ && for /f "delims=" %%x in ('dir /b *.jar') do java -jar %%x"
)
```

If everything above works, then Docker images are built for each micro-service that has been updated.

```
alex@ dell:~/SEPT/BackEnd/Books$ sudo docker build -t moving-houses/books .
[sudo] password for alex:
Sending build context to Docker daemon 84.94MB
Step 1/6 : FROM openjdk:11
--> dd8edf47a855
Step 2/6 : VOLUME /tmp
--> Using cache
--> 8e86371532b6
Step 3/6 : ARG JAR_FILE=target/*.jar
--> Using cache
--> Se4b0094c40d
Step 4/6 : COPY ${JAR_FILE} app.jar
--> Using cache
--> 024ebab0943f9
Step 5/6 : EXPOSE 8082
--> Using cache
--> b3e261982ea7
Step 6/6 : ENTRYPOINT ["java", "-jar", "/app.jar"]
--> Using cache
--> c24ba848a465
Successfully built c24ba848a465
Successfully tagged moving-houses/books:latest
```

These containers are then run on AWS, able to be accessed

```
alexdepell:~/SEPT/BackEnd/Books$ sudo docker run -p 8082:8082 moving-houses/books
.
.
.
:: Spring Boot ::          (v2.3.2.RELEASE)

2021-10-23T12:45:23.188+0000 INFO Starting BookApplication v1.0.0 on 898081224a2b with PID 1 (/app.jar started by root in /)
2021-10-23T12:45:23.204+0000 INFO No active profile set, falling back to default profiles: default
2021-10-23T12:45:25.756+0000 INFO Bootstrapping Spring Data JPA repositories in DEFERRED mode.
2021-10-23T12:45:25.995+0000 INFO Finished Spring Data repository scanning in 266ms. Found 4 JPA repository interfaces.
2021-10-23T12:45:26.949+0000 INFO Bean [org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler@4ae33a11] of type [org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2021-10-23T12:45:26.976+0000 INFO Bean [methodMetadataDataSource] of type [org.springframework.security.access.method.DelegatingMethodSecurityMetadataSource] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2021-10-23T12:45:27.737+0000 INFO Tomcat initialized with port(s): 8082 (http)
2021-10-23T12:45:27.764+0000 INFO Initializing ProtocolHandler ["http-nio-8082"]
2021-10-23T12:45:27.767+0000 INFO Starting service [Tomcat]
2021-10-23T12:45:27.769+0000 INFO Starting Servlet engine: [Apache Tomcat/9.0.37]
2021-10-23T12:45:27.927+0000 INFO Initializing Spring embedded WebApplicationContext
2021-10-23T12:45:28.315+0000 WARN spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2021-10-23T12:45:28.359+0000 INFO Initializing ExecutorService 'applicationTaskExecutor'
2021-10-23T12:45:28.725+0000 INFO HHH0000204: Processing PersistenceUnitInfo [name: default]
2021-10-23T12:45:28.979+0000 INFO HHH000412: Hibernate ORM core version 5.4.18.Final
2021-10-23T12:45:29.659+0000 INFO HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
2021-10-23T12:45:30.105+0000 INFO HikarikPool-1 - Starting...
2021-10-23T12:45:30.588+0000 INFO Creating filter chain: any request, [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@37095ded, org.springframework.web.context.SecurityContextPersistenceFilter@6aefb10e, org.springframework.security.web.header.HeaderWriterFilter@469d003c, org.springframework.web.filter.CorsFilter@2eb79cbe, org.springframework.security.web.authentication.logout.LogoutFilter@2e1792e7, com.rmit.books.security.JwtAuthenticationFilter@1500e009, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@464a4442, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@412037f0, org.springframework.security.web.authentication.logout.LogoutFilter@412037f0]
```

CircleCI:

The screenshot shows a detailed view of a CircleCI pipeline history across several days. The pipeline is named 'SEPT-ci'. It lists multiple branches (main, 536e6f4, 8127655, b63afda, 6e1ab78) with their commit details, start times, durations, and artifact links. Each branch has one or more 'frontend-test' jobs associated with it, which also show their status, duration, and artifacts.

Pipeline	Status	Workflow	Branch / Commit	Start	Duration	Artifacts
SEPT 346	Success	SEPT-ci	main 6e1ab78 Updated README.md in main directory	10m ago	58s	View Artifacts
		Jobs	frontend-test 348		55s	
SEPT 345	Success	SEPT-ci	main 536e6f4 Added README.md for building and running the application locally	14m ago	2m 32s	View Artifacts
		Jobs	frontend-test 347		2m 22s	
SEPT 344	Success	SEPT-ci	main 8127655 Fixed errors in tests	31m ago	1m 6s	View Artifacts
		Jobs	frontend-test 346		1m 0s	
SEPT 343	Success	SEPT-ci	main b63afda Removed unused test	39m ago	1m 33s	View Artifacts
		Jobs	frontend-test 345		1m 28s	
SEPT 342	Success	SEPT-ci	main 6e1ab78 Merge branch 'main' of https://github.com/AlexAlo/SEPT	1h ago	1m 10s	View Artifacts
		Jobs	frontend-test 344		1m 7s	
SEPT 341	Success	SEPT-ci	main 6e1ab78 Merge pull request #36 from AlexAlo/cleanup	2h ago	44s	View Artifacts
		Jobs	frontend-test 343		41s	
SEPT 339	Success	SEPT-ci	main 1e89b73 Added meeting minutes 27	2h ago	1m 24s	View Artifacts
		Jobs	frontend-test 341		1m 20s	
SEPT 338	Success	SEPT-ci	main 97cb8d3 Scrum for M3	2h ago	51s	View Artifacts
		Jobs	frontend-test 340		48s	

0 00:54 Use parallelism to run faster tests Go to Docs

Spin up environment

```

0:00:54 [0] Pulling fs layer
--> c887c289e Pulling fs layer
0:00:56 [0] Pulling fs layer
--> 3de5926e Pulling fs layer
0:00:58 [0] Pulling fs layer
--> 4026582f Pulling fs layer
0:00:59 [0] Pulling fs layer
--> 4b234e03 Pulling fs layer
0:01:00 [0] Pulling fs layer
--> 4d1c1397 Pulling fs layer
0:01:01 [0] Pulling fs layer
--> circleci/node:10.16.1 Pulling fs layer
0:01:01 [0] Using image circleci/node@sha256:55d4fc1e2c353036c43b270d1cf8a9e4bfefde39af73019e90210d10ba1
0:01:01 [0] pull stats: download 160MB in 1.72s (97.46MB/s), extract 160.5MB in 4.11s (41.21MB/s)
0:01:01 [0] time: 5.83s
0:01:01 [0] Starting container repmovinghouses/sept:latest
0:01:01 [0] Digest: sha256:55d4fc1e2c353036c43b270d1cf8a9e4bfefde39af73019e90210d10ba1
0:01:01 [0] status: Downloaded newer image for circleci/node:10.16.1
0:01:01 [0] image cache not found on this host, downloading repmovinghouses/sept:latest
0:01:01 [0] latest: Pulling from repmovinghouses/sept
0:01:01 [0] 
0:01:01 [0] 10d6f6ee: Pulling fs layer
0:01:01 [0] e8f85001: Pulling fs layer
0:01:01 [0] e22ec011: Pulling fs layer
0:01:01 [0] ff3b3c6d: Pulling fs layer
0:01:01 [0] b10239d9: Pulling fs layer
0:01:01 [0] 0f6f7e0f: Pulling fs layer
0:01:01 [0] Digest: sha256:b93a5e1ef09272b5e3e49442b202b962c920666751bea17e77bea414355b747eb/39.47mb
0:01:01 [0] status: Downloaded newer image for repmovinghouses/sept:latest
0:01:01 [0] repmovinghouses/sept:latest
0:01:01 [0] pull stats: download 354.1mb in 6.27s (52.03MB/s), extract 354mb in 5.45s (64.94MB/s)
0:01:01 [0] time to create container: 1.25s
0:01:01 [0] Time to upload agent and config: 773.2220ms
0:01:01 [0] Time to start container: 1.40893281s

```

Container septmovinghouses/sept:latest

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
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
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
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
715
716
717
718
719
719
720
721
722
723
724
725
725
726
727
728
729
729
730
731
732
733
734
735
735
736
737
738
739
739
740
741
742
743
744
744
745
746
746
747
748
749
749
750
751
752
753
754
755
755
756
757
758
758
759
759
760
761
762
763
764
764
765
766
767
767
768
769
769
770
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
782
782
783
783
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
152
```

Testing Documentation

Unit Testing

Testing: Reviews

Tester: Carl Karama

Types of tests for Review Controller

Test Results		2 sec 968 ms
ReviewControllerTest		2 sec 968 ms
✓	Test: addReviewIsBadRequest() [Fail]	255 ms
✓	testGetAllReview()	73 ms
✓	Test: addReview() [Success]	1 sec 590 ms
✓	Test: transactionIDisNull() [Fail]	50 ms
✓	testReviewIDisNull()	1 sec

Test: *addReviewIsBadRequest()*

Test Result: Pass

Implementation Specification: When a bad request is made i.e. bad syntax the server fails to understand triggers this which has a HTTP status code of 400.

Implementation Notes: I assert an HTTP.BAD_REQUEST as expected and the actual test results affirm that when I pass bad syntax through the JSON string which hits simulated mock api endpoint.

```
118  @Test
119  @DisplayName("Test: addReviewIsBadRequest() [Fail]")
120  public void testAddReviewIsBadRequest() throws Exception {
121
122      when(mapValidationErrorService.MapValidationService(ArgumentMatchers.any(BindingResult.class))).thenReturn(null);
123      when(reviewService.addReview(ArgumentMatchers.any(Review.class))).thenReturn(reviews.get(0));
124
125      String input = "";
126
127      RequestBuilder requestBuilder = MockMvcRequestBuilders.post( urlTemplate: "/api/review/addReview"
128
129          .accept(MediaType.APPLICATION_JSON).content(input).contentType(MediaType.APPLICATION_JSON);
130
131      MvcResult result = mockMvc.perform(requestBuilder).andReturn();
132      MockHttpServletResponse response = result.getResponse();
133
134      assertEquals(HttpStatus.BAD_REQUEST.value(), response.getStatus());
135
136  }
```

Test: *getAllReviews()*

Test Result: Pass

Implementation Specification: Returns all the reviews from the database

```
188     @Test
189     public void testGetAllReview() throws Exception {
190
191         when(reviewService.getAllReviews()).thenReturn(reviews);
192
193         mockMvc.perform(
194             MockMvcRequestBuilders.get( urlTemplate: "/api/review/all").accept(MediaType.APPLICATION_JSON))
195             .andExpect(MockMvcResultMatchers.status().isOk());
196     }
197
198 }
199 }
```

Test: addReview()

Test Result: Pass

Implementation Specification: This tests the methods ability to successfully add a review

Implementation Notes: I assert an HTTP.OK as expected and the actual test results affirm that response successfully when I pass a payload through the JSON string which hits the simulated mock API endpoint for addReview.

```
96     @Test
97     @DisplayName("Test: addReview() [Success]")
98     public void testAddReview() throws Exception {
99
100         when(mapValidationErrorResponseService.MapValidationService(ArgumentMatchers.any(BindingResult.class))).thenReturn(null);
101         when(reviewService.addReview(ArgumentMatchers.any(Review.class))).thenReturn(reviews.get(0));
102
103
104         String input = "{\n" +
105             "    \"userRating\": 2,\n" +
106             "    \"bookRating\": 2,\n" +
107             "    \"transactionId\": 3,\n" +
108             "    \"review\": \"It was not the best\"\n" +
109             "}";
110
111         RequestBuilder requestBuilder = MockMvcRequestBuilders.post( urlTemplate: "/api/review/addReview")
112
113             .accept(MediaType.APPLICATION_JSON).content(input).contentType(MediaType.APPLICATION_JSON);
114
115         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
116         MockHttpServletResponse response = result.getResponse();
117
118         assertEquals(HttpStatus.OK.value(), response.getStatus());
119     }
120 }
```

Test: transactionIDisNull()

Test Result: Pass

Implementation Specification: This tests the methods ability to successfully add a review

Implementation Notes: I assert an HTTP.NOT as expected and the actual test results affirm that response successfully when I pass a payload through the JSON string which hits the simulated mock API endpoint for addReview.

```
143     @Test
144     @DisplayName("Test: transactionIDisNull() [Fail]")
145     public void testTransactionIDisNull() throws Exception {
146
147         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class))).thenReturn(null);
148         when(reviewService.addReview(ArgumentMatchers.any(Review.class))).thenReturn(reviews.get(0));
149
150         String input = "{\n" +
151             "    \"userRating\": 2,\n" +
152             "    \"bookRating\": 2,\n" +
153             "    \"transactionId\": null,\n" +
154             "    \"review\": \"It was not the best\"\n" +
155             "}";
156
157         RequestBuilder requestBuilder = MockMvcRequestBuilders.post(urlTemplate: "/api/review/addReview")
158
159             .accept(MediaType.APPLICATION_JSON).content(input).contentType(MediaType.APPLICATION_JSON);
160
161         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
162         MockHttpServletResponse response = result.getResponse();
163
164         assertEquals(HttpStatus.NOT_ACCEPTABLE.value(), response.getStatus());
165     }
```

Types of Tests for Service

▼ Test Results	275 ms
▼ ReviewServiceTest	275 ms
Test: getAllReviews() [Success]	93 ms
Test: incrementRating() [Fail]	69 ms
Test: incrementRating() [Success]	58 ms
Test: addReview() [Fail]	37 ms
Test: addReview() [Success]	18 ms

Test: getAllReviews()

Test Result: Pass

Implementation Specification: This tests the methods ability to successfully get all reviews

```
92     @Test
93     @DisplayName("Test: getAllReviews() [Success]")
94     public void getAllReviews() throws Exception {
95         given(reviewRepository.findAll()).willReturn(reviews);
96         Iterable<Review> reviewIterable = reviewService.getAllReviews();
97         Assert.assertNotNull(reviewIterable);
98     }
```

Test: addReview()

Test Result: Pass

Implementation Specification: This tests the methods ability to successfully add reviews

```
92     @Test
93     @DisplayName("Test: addReview() [Success]")
94     public void testAddReview() {
95         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class))).thenReturn(null);
96         when(reviewService.addReview(ArgumentMatchers.any(Review.class))).thenReturn(reviews.get(0));
97         Review review = reviewService.addReview(reviews.get(0));
98         Assert.assertNotNull(review);
99     }
```

Test: incrementRating()

Test Result: Pass

Implementation Specification: This tests the methods ability to successfully get increment rating reviews

```
92     @Test
93     @DisplayName("Test: incrementRating() [Success]")
94     public void incrementRating() {
95         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class))).thenReturn(null);
96         when(reviewService.addReview(ArgumentMatchers.any(Review.class))).thenReturn(reviews.get(0));
97
98         reviewService.incrementRating(reviews.get(0));
99     }
```

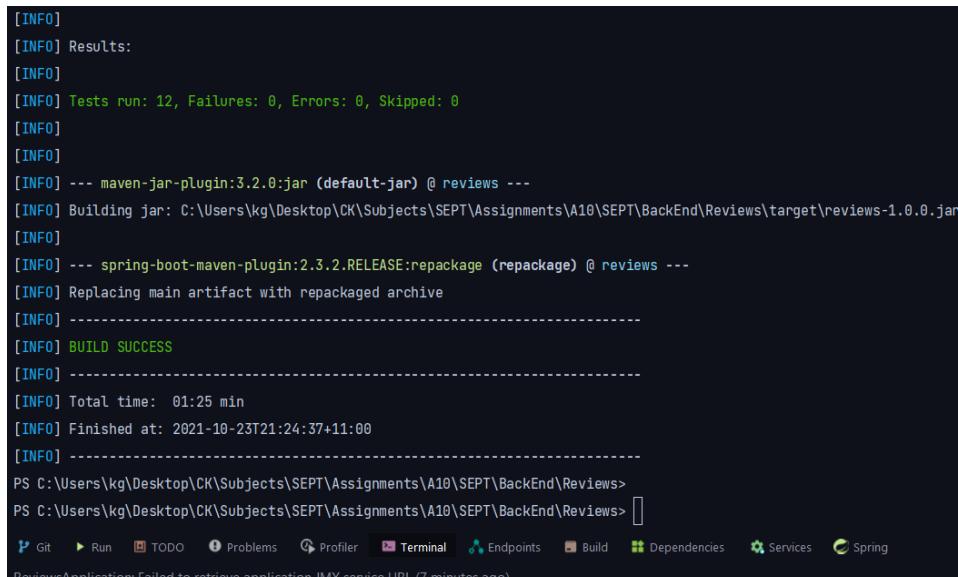
Types of Tests for Repo

▼ Test Results	3 sec 268 ms
▼ ReviewRepositoryTest	3 sec 268 ms
Test: findNonExistingReview [Pass]	1 sec 588 ms
Test: addReview [Pass]	1 sec 680 ms

```

[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ reviews ---
[INFO] Building jar: C:\Users\kg\Desktop\CK\Subjects\SEPT\Assignments\A10\SEPT\BackEnd\Reviews\target\reviews-1.0.0.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.3.2.RELEASE:repackage (repackage) @ reviews ---
[INFO] Replacing main artifact with repackaged archive
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:25 min
[INFO] Finished at: 2021-10-23T21:24:37+11:00
[INFO] -----
PS C:\Users\kg\Desktop\CK\Subjects\SEPT\Assignments\A10\SEPT\BackEnd\Reviews>
PS C:\Users\kg\Desktop\CK\Subjects\SEPT\Assignments\A10\SEPT\BackEnd\Reviews> []

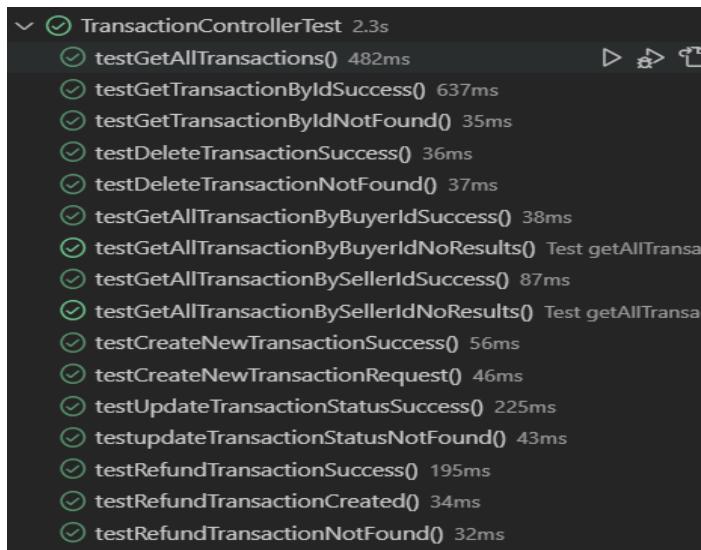
```



Testing: Transactions

Tester: Shannon Dann

Tests for Transaction Controller



```

@Test
@DisplayName("Test getAllTransactions") // test for getting all books
void testGetAllTransactions() throws Exception {
    // Mocking service
    when(transactionService.findAllTransactions()).thenReturn(transactions);
    mockMvc.perform(get("/api/transactions/all").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id", is(1)))
        .andExpect(jsonPath("$.buyer.id", is(1)))
        .andExpect(jsonPath("$.book.id", is(1)))
        .andExpect(jsonPath("$.price", is(99.99)))
        .andExpect(jsonPath("$.status", is(TransactionStatus.PROCESSING.toString())))
        .andExpect(jsonPath("$.quantity", is(3)))
        .andExpect(jsonPath("$.isReviewed", is(false)))

        .andExpect(jsonPath("[1].id", is(2)))
        .andExpect(jsonPath("[1].buyer.id", is(1)))
        .andExpect(jsonPath("[1].book.id", is(1)))
        .andExpect(jsonPath("[1].price", is(99.99)))
        .andExpect(jsonPath("[1].status", is(TransactionStatus.PROCESSING.toString())))
        .andExpect(jsonPath("[1].quantity", is(3)))
        .andExpect(jsonPath("[1].isReviewed", is(false)));
}

```

Test: `testGetAllTransactions()`

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/all” for returning all transactions in the system.

Implementation Notes: Checks the response and makes sure it is HTTP status OK and that all the returned information is correct and in the correct format.

```

@Test
@DisplayName("Test getTransactionById success") // test for getting a transaction sucessfully
void testGetTransactionByIdSuccess() throws Exception {
    // Mocking service
    when(transactionService.findById(1L)).thenReturn(transactions.get(0));
    mockMvc.perform(get("/api/transactions/1").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("id", is(1)).andExpect(jsonPath("buyer.id", is(1))))
        .andExpect(jsonPath("book.id", is(1)).andExpect(jsonPath("price", is(99.99))))
        .andExpect(jsonPath("status", is(TransactionStatus.PROCESSING.toString())))
        .andExpect(jsonPath("quantity", is(3)))
        .andExpect(jsonPath("isReviewed", is(false)));
}

```

Test: `testGetTransactionByIdSuccess()`

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/{id}” for returning a transaction by its ID.

Implementation Notes: Checks the response and makes sure it is HTTP status OK and that all the returned information is correct and in the correct format.

```

@Test
@DisplayName("Test getTransactionById not found") // test for getting a transaction that doesn't exist
void testGetTransactionByIdNotFound() throws Exception {
    // Mocking service
    when(transactionService.findById(3L)).thenReturn(null);

    RequestBuilder requestBuilder = MockMvcRequestBuilders.get("/api/transactions/3")
        .contentType(MediaType.APPLICATION_JSON);
    MvcResult result = mockMvc.perform(requestBuilder).andReturn();

    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
}

```

Test: *testGetTransactionByIdNotFound()*

Test Result: Pass

Implementation Specification: Tests api call get “/api/transactions/{id}” throwing not found when transaction does not exist.

Implementation Notes: Checks the response and makes sure it returns HTTP NOT FOUND as expected.

```

@Test
@DisplayName("Test deleteTransaction success") // test for successfully deleting a transaction
void testDeleteTransactionSuccess() throws Exception {
    // Mocking service
    when(transactionService.findById(1L)).thenReturn(transactions.get(0));

    RequestBuilder requestBuilder = MockMvcRequestBuilders.delete("/api/transactions/1");

    MvcResult result = mockMvc.perform(requestBuilder).andReturn();
    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.OK.value(), response.getStatus());
    assertEquals("Transaction with ID 1 was deleted", response.getContentAsString());
}

```

Test: *testDeleteTransactionSuccess()*

Test Result: Pass

Implementation Specification: Tests api delete call “/api/transactions/{id}” successfully deleting a transaction.

Implementation Notes: Checks the response and makes sure it returns HTTP OK as expected and text saying the transaction was deleted.

```

@Test
@DisplayName("Test deleteTransaction not found") // test for deleting a transaction that doesn't exist
void testDeleteTransactionNotFound() throws Exception {
    // Mocking service
    when(transactionService.findById(3L)).thenReturn(null);

    RequestBuilder requestBuilder = MockMvcRequestBuilders.delete("/api/transactions/3")
        .contentType(MediaType.APPLICATION_JSON);
    MvcResult result = mockMvc.perform(requestBuilder).andReturn();

    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
}

```

Test: *testDeleteTransactionNotFound()*

Test Result: Pass

Implementation Specification: Tests api delete call “/api/transactions/{id}” throwing not found when transaction does not exist

Implementation Notes: Checks the response and makes sure it returns HTTP NOT FOUND as expected.

```
@Test
@DisplayName("Test getAllTransactionByBuyerId success") // test for getting all transactions by buyer ID sucessfully
void testGetAllTransactionByBuyerIdSuccess() throws Exception {
    // Mocking service
    when(transactionService.getAllByBuyerID(1L)).thenReturn(transactions);
    mockMvc.perform(get("/api/transactions/buyer/1").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id", is(1))).andExpect(jsonPath("$.buyer.id", is(1)))
        .andExpect(jsonPath("$.book.id", is(1))).andExpect(jsonPath("$.price", is(99.99)))
        .andExpect(jsonPath("$.status", is(TransactionStatus.PROCESSING.toString())))
        .andExpect(jsonPath("$.quantity", is(3)))
        .andExpect(jsonPath("$.isReviewed", is(false)))

        .andExpect(jsonPath("[1].id", is(2))).andExpect(jsonPath("[1].buyer.id", is(1)))
        .andExpect(jsonPath("[1].book.id", is(1))).andExpect(jsonPath("[1].price", is(99.99)))
        .andExpect(jsonPath("[1].status", is(TransactionStatus.PROCESSING.toString())))
        .andExpect(jsonPath("[1].quantity", is(3)))
        .andExpect(jsonPath("[1].isReviewed", is(false)));
}
```

Test: `testGetAllTransactionByBuyerIdSuccess()`

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/buyer/{id}” for returning transactions with given buyer ID.

Implementation Notes: Checks the response and makes sure it is HTTP status OK and that all the returned information is correct and in the correct format.

```
@Test
@DisplayName("Test getAllTransactionByBuyerId no results") // test for getting a transaction when there are no results
void testGetAllTransactionByBuyerIdNoResults() throws Exception {
    // Mocking service
    Iterable<Transaction> emptyTransactions = new ArrayList<Transaction>();
    when(transactionService.getAllByBuyerID(3L)).thenReturn(emptyTransactions);

    RequestBuilder requestBuilder = MockMvcRequestBuilders.get("/api/transactions/buyer/3")
        .contentType(MediaType.APPLICATION_JSON);
    MvcResult result = mockMvc.perform(requestBuilder).andReturn();

    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.OK.value(), response.getStatus());
    assertEquals("[]", response.getContentAsString());
}
```

Test: `testGetAllTransactionByBuyerIdNoResults()`

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/buyer/{id}” for returning empty list when no transactions are found.

Implementation Notes: Checks the response and makes sure it is HTTP status OK and that an empty string is returned.

```

@test
@DisplayName("Test getAllTransactionBySellerId success") // test for getting all transactions by seller ID sucessfully
void testGetAllTransactionBySellerIdSuccess() throws Exception {
    // Mocking service
    when(transactionService.getAllBySellerID(1L)).thenReturn(transactions);
    mockMvc.perform(get("/api/transactions/seller/1").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id", is(1))).andExpect(jsonPath("$.buyer.id", is(1)))
        .andExpect(jsonPath("$.book.id", is(1))).andExpect(jsonPath("$.price", is(99.99)))
        .andExpect(jsonPath("$.status", is(TransactionStatus.PROCESSING.toString())))
        .andExpect(jsonPath("$.quantity", is(3)))
        .andExpect(jsonPath("$.isReviewed", is(false)));

        .andExpect(jsonPath("[1].id", is(2))).andExpect(jsonPath("[1].buyer.id", is(1)))
        .andExpect(jsonPath("[1].book.id", is(1))).andExpect(jsonPath("[1].price", is(99.99)))
        .andExpect(jsonPath("[1].status", is(TransactionStatus.PROCESSING.toString())))
        .andExpect(jsonPath("[1].quantity", is(3)))
        .andExpect(jsonPath("[1].isReviewed", is(false)));
}

```

Test: `testGetAllTransactionBySellerIdSuccess()`

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/seller/{id}” for returning transactions with given seller ID.

Implementation Notes: Checks the response and makes sure it is HTTP status OK and that all the returned information is correct and in the correct format.

```

@test
@DisplayName("Test getAllTransactionBySellerId no results") // test for getting a transaction when there are no results
void testGetAllTransactionBySellerIdNoResults() throws Exception {
    // Mocking service
    Iterable<Transaction> emptyTransactions = new ArrayList<Transaction>();
    when(transactionService.getAllBySellerID(3L)).thenReturn(emptyTransactions);

    RequestBuilder requestBuilder = MockMvcRequestBuilders.get("/api/transactions/seller/3")
        .contentType(MediaType.APPLICATION_JSON);
    MvcResult result = mockMvc.perform(requestBuilder).andReturn();

    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.OK.value(), response.getStatus());
    assertEquals("[]", response.getContentAsString());
}

```

Test: `testGetAllTransactionBySellerIdNoResults()`

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/seller/{id}” for returning empty list when no transactions are found.

Implementation Notes: Checks the response and makes sure it is HTTP status OK and that an empty string is returned.

```

@test
@DisplayName("Test createNewTransaction success") // test for creating a new transaction successfully
void testCreateNewTransactionSuccess() throws Exception {
    // Mocking service
    when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
        .thenReturn(null);
    when(transactionService.saveTransaction(ArgumentMatchers.any(Transaction.class)))
        .thenReturn(transactions.get(0));
    String inputJson = "{\n" + " \"buyerID\": \"1\", \n" + " \"bookID\": \"1\", \n"
        + " \"price\": \"99.99\", \n" + " \"quantity\": \"3\"\n" + "}";
    RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/transactions/new")
        .accept(MediaType.APPLICATION_JSON)
        .content(inputJson).contentType(MediaType.APPLICATION_JSON);

    MvcResult result = mockMvc.perform(requestBuilder).andReturn();
    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.CREATED.value(), response.getStatus());
}

```

Test: *testCreateNewTransactionSuccess()*

Test Result: *Pass*

Implementation Specification: Tests api post call “/api/transactions/new” for creating a transaction with given information.

Implementation Notes: Checks the response and makes sure it is HTTP status CREATED for the transaction being created successfully.

```
@Test
@DisplayName("Test createNewTransaction badRequest") // test for creating a new transaction with invalid information
void testCreateNewTransactionBadRequest() throws Exception {
    // Mocking service
    when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
        .thenReturn(null);

    String inputJson = "null";
    RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/transactions/new")
        .accept(MediaType.APPLICATION_JSON)
        .content(inputJson).contentType(MediaType.APPLICATION_JSON);

    MvcResult result = mockMvc.perform(requestBuilder).andReturn();
    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.BAD_REQUEST.value(), response.getStatus());
}
```

Test: *testCreateNewTransactionBadRequest()*

Test Result: *Pass*

Implementation Specification: Tests api post call “/api/transactions/new” throwing bad request when given information is invalid.

Implementation Notes: Checks the response and makes sure it is HTTP status BAD REQUEST.

```
@Test
@DisplayName("Test updateTransactionStatus success") // test for updating a transaction status successfully
void testUpdateTransactionStatusSuccess() throws Exception {
    // Mocking service
    when(transactionService.findById(1L)).thenReturn(transactions.get(0));
    when(transactionService.updateTransactionStatus(ArgumentMatchers.any(TransactionStatus.class)),
        ArgumentMatchers.any(Transaction.class))
        .thenReturn(transactions.get(0));

    String inputJson = "\n" + "\"status\":\"PRE_ORDER\"\n" + "}";
    RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/transactions/updateStatus/1")
        .accept(MediaType.APPLICATION_JSON).content(inputJson)
        .contentType(MediaType.APPLICATION_JSON);

    MvcResult result = mockMvc.perform(requestBuilder).andReturn();
    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.OK.value(), response.getStatus());
}
```

Test: *testUpdateTransactionStatusSuccess()*

Test Result: *Pass*

Implementation Specification: Tests api post call “/api/transactions/updateStatus/{id}” updating the status of a given transaction with the given status.

Implementation Notes: Checks the response and makes sure it is HTTP status OK to show the transaction was updated successfully.

```

@Test
@DisplayName("Test updateTransactionStatus not found") // test for updating a transaction status with invalid information
void testupdateTransactionStatusNotFound() throws Exception {
    // Mocking service
    when(transactionService.findById(3L)).thenReturn(null);
    when(transactionService.updateTransactionStatus(ArgumentMatchers.any(TransactionStatus.class)),
        ArgumentMatchers.any(Transaction.class))
        .thenReturn(transactions.get(0));

    String inputJson = "{\n" + "\"status\":\"PRE_ORDER\"\n" + "}";
    RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/transactions/updateStatus/1")
        .accept(MediaType.APPLICATION_JSON).content(inputJson)
        .contentType(MediaType.APPLICATION_JSON);

    MvcResult result = mockMvc.perform(requestBuilder).andReturn();
    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
}

```

Test: *testupdateTransactionStatusNotFound()*

Test Result: Pass

Implementation Specification: Tests api post call “/api/transactions/updateStatus/{id}” throwing not found when transaction doesn’t exist.

Implementation Notes: Checks the response and makes sure it is HTTP status NOT FOUND as expected.

```

@Test
@DisplayName("Test refundTransaction success") // test for refunding transcation successfully
void testRefundTransactionSuccess() throws Exception {
    // Mocking service
    when(transactionService.findById(1L)).thenReturn(transactions.get(0));
    when(transactionService.refundTransaction(transactions.get(0))).thenReturn(true);
    MvcResult result = mockMvc.perform(get("/api/transactions/refund/1")
        .contentType(MediaType.APPLICATION_JSON)).andReturn();
    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.OK.value(), response.getStatus());
    assertEquals("Refund successful.", response.getContentAsString());
}

```

Test: *testRefundTransactionSuccess()*

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/refund/{id}” successfully refunding.

Implementation Notes: Checks the response and makes sure it is HTTP status OK and text saying refund successful.

```

@Test
@DisplayName("Test refundTransaction created") // test for refunding transcation successfully
void testRefundTransactionCreated() throws Exception {
    // Mocking service
    when(transactionService.findById(1L)).thenReturn(transactions.get(0));
    when(transactionService.refundTransaction(transactions.get(0))).thenReturn(false);
    MvcResult result = mockMvc.perform(get("/api/transactions/refund/1")
        .contentType(MediaType.APPLICATION_JSON)).andReturn();
    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.OK.value(), response.getStatus());
    assertEquals("Refund request was created.", response.getContentAsString());
}

```

Test: *testRefundTransactionCreated()*

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/refund/{id}” creating a refund request.

Implementation Notes: Checks the response and makes sure it is HTTP status OK and text saying refund request was created.

```
@Test
@DisplayName("Test refundTransaction not found") // test for refunding transcation and transaction not found
void testRefundTransactionNotFound() throws Exception {
    // Mocking service
    when(transactionService.findById(3L)).thenReturn(null);
    MvcResult result = mockMvc.perform(get("/api/transactions/refund/1")
        .contentType(MediaType.APPLICATION_JSON)).andReturn();
    MockHttpServletResponse response = result.getResponse();

    assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
}
```

Test: *testRefundTransactionNotFound()*

Test Result: Pass

Implementation Specification: Tests api get call “/api/transactions/refund/{id}” throwing not found if transaction is not found.

Implementation Notes: Checks the response and makes sure it is HTTP status NOT FOUND.

Tests for Transaction Service

```
✓ TransactionServiceTest 289ms
  ✓ testFindAllTransactionsSuccess() 102ms
  ✓ testGetAllBySellerIDSuccess() 50ms
  ✓ testGetAllBySellerIDFail() 49ms
  ✓ testGetAllByBuyerIDSuccess() 45ms
  ✓ testGetAllByBuyerIDFail() 43ms
```

```
@Test
@DisplayName("Test findAllTranscations success") // test for finding all transcation
public void testfindAllTransactionsSuccess() throws Exception {
    given(transactionRepository.findAll()).willReturn(transactions);
    Iterable<Transaction> allTransactions = transactionService.findAllTransactions();
    Assert.assertNotNull(allTransactions);
}
```

Test: *testfindAllTransactionsSuccess()*

Test Result: Pass

Implementation Specification: Test for method findAllTransactions successfully returning transactions.

Implementation Notes: Checks to make sure that the transactions returned are not null.

```

@Test
@DisplayName("Test getAllBySellerID success") // test for finding all transaction by seller ID successfully
public void testGetAllBySellerIDSuccess() throws Exception {
    User user = new User();
    given(userRepository.getById(1L)).willReturn(user);
    given(bookRepository.findBySeller(user)).willReturn(books);
    given(transactionRepository.findByBookIn(books)).willReturn(transactions);
    Iterable<Transaction> allTransactions = transactionService.getAllBySellerID(1L);
    Assert.assertNotNull(allTransactions);
}

```

Test: `testGetAllBySellerIDSuccess()`

Test Result: Pass

Implementation Specification: Test for method `getAllBySellerID` successfully returning transactions.

Implementation Notes: Checks to make sure that the transactions returned is not null.

```

@Test
@DisplayName("Test getAllBySellerID fail") // test for finding all transaction by seller ID that doesn't exist
public void testGetAllBySellerIDFail() throws Exception {
    given(userRepository.getById(null)).willReturn(null);
    given(bookRepository.findBySeller(null)).willReturn(null);
    given(transactionRepository.findByBookIn(null)).willReturn(null);
    Iterable<Transaction> allTransactions = transactionService.getAllBySellerID(null);
    Assert.assertNull(allTransactions);
}

```

Test: `testGetAllBySellerIDFail()`

Test Result: Pass

Implementation Specification: Test for method `getAllBySellerID` returning null when there are no transactions.

Implementation Notes: Checks to make sure that no transactions were returned.

```

@Test
@DisplayName("Test getAllByBuyerID success") // test for finding all transaction by buyer ID successfully
public void testGetAllByBuyerIDSuccess() throws Exception {
    User user = new User();
    given(userRepository.getById(1L)).willReturn(user);
    given(transactionRepository.findByBuyer(user)).willReturn(transactions);
    Iterable<Transaction> allTransactions = transactionService.getAllByBuyerID(1L);
    Assert.assertNotNull(allTransactions);
}

```

Test: `testGetAllByBuyerIDSuccess()`

Test Result: Pass

Implementation Specification: Test for method `getAllByBuyerID` successfully returning transactions.

Implementation Notes: Checks to make sure that the transactions returned is not null.

```

@Test
@DisplayName("Test getAllByBuyerID fail") // test for finding all transaction by buyer ID that doesn't exist
public void testGetAllByBuyerIDFail() throws Exception {
    given(userRepository.getById(null)).willReturn(null);
    given(transactionRepository.findByBuyer(null)).willReturn(null);
    Iterable<Transaction> allTransactions = transactionService.getAllByBuyerID(null);
    Assert.assertNull(allTransactions);
}

```

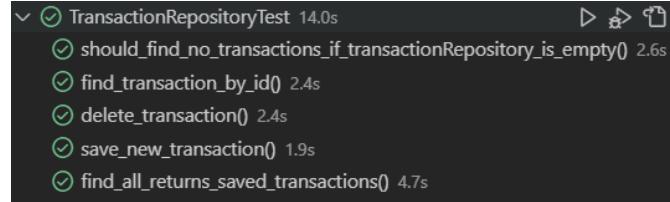
Test: `testGetAllByBuyerIDFail()`

Test Result: Pass

Implementation Specification: Test for method getAllByBuyerID returning null when there are no transactions.

Implementation Notes: Checks to make sure that no transactions were returned.

Tests for Transaction Repository



```
// testing for empty transaction repository
@Test
@Rollback(true)
public void should_find_no_transactions_if_transactionRepository_is_empty() {
    Iterable<Transaction> transactions = transactionRepository.findAll();
    if (transactions.iterator().hasNext()) {
        assertThat(transactions).isNotEmpty();
    } else {
        assertThat(transactions).isEmpty();
    }
}
```

Test: *should_find_no_transactions_if_transactionRepository_is_empty()*

Test Result: Pass

Implementation Specification: Tests repository to check if it is empty.

Implementation Notes: goes through all transactions to make sure they are in the returned list.

```
//test to find a transaction by id
@Test
@Rollback(true)
public void find_transaction_by_id() {
    Book book = new Book();
    book.setId(1L);
    User buyer = new User();
    buyer.setId(1L);

    Transaction transaction = new Transaction();
    transaction.setId(1L);
    transaction.setBuyer(buyer);
    transaction.setBook(book);
    transaction.setPrice(99.99);
    transaction.setStatus(TransactionStatus.PROCESSING);
    transaction.setQuantity(3);
    transaction.setIsReviewed(false);
    transaction.setCreatedDate(new Date());
    transactionRepository.save(transaction);

    Transaction findTransaction = transactionRepository.getById(1L);
    assertNotNull(findTransaction);

    assertThat(findTransaction).hasFieldOrPropertyWithValue("id", 1L);
    assertThat(findTransaction).hasFieldOrPropertyWithValue("price", 99.99);
    assertThat(findTransaction).hasFieldOrPropertyWithValue("status", TransactionStatus.PROCESSING);
    assertThat(findTransaction).hasFieldOrPropertyWithValue("quantity", 3);
    assertThat(findTransaction).hasFieldOrPropertyWithValue("isReviewed", false);
    assertThat(findTransaction.getBook()).hasFieldOrPropertyWithValue("id", 1L);
    assertThat(findTransaction.getBuyer()).hasFieldOrPropertyWithValue("id", 1L);
}
```

Test: *find_transaction_by_id()*

Test Result: Pass

Implementation Specification: Tests repository for finding a transaction by given ID.

Implementation Notes: checks to make sure a transaction was found and then checks the information of that transaction are correct.

```
// test to delete a transaction
@Test
@Rollback(true)
public void delete_transaction() {
    Book book = new Book();
    book.setId(1L);
    User buyer = new User();
    buyer.setId(1L);

    Transaction transaction = new Transaction();
    transaction.setId(1L);
    transaction.setBuyer(buyer);
    transaction.setBook(book);
    transaction.setPrice(99);
    transaction.setStatus(TransactionStatus.PROCESSING);
    transaction.setQuantity(3);
    transaction.setIsReviewed(false);
    transaction.setCreatedAt(new Date());
    Transaction savedTransaction = transactionRepository.save(transaction);

    transactionRepository.delete(savedTransaction);

    Transaction findTransaction = transactionRepository.getById(1L);
    assertNull(findTransaction);
}
```

Test: *delete_transaction()*

Test Result: Pass

Implementation Specification: Tests repository for deleting a transaction.

Implementation Notes: checks to make sure that no transaction is returned once deleted.

```
//test to find a transaction by id
@Test
@Rollback(true)
public void save_new_transaction() {
    Book book = new Book();
    book.setId(1L);
    User buyer = new User();
    buyer.setId(1L);

    Transaction transaction = new Transaction();
    transaction.setId(1L);
    transaction.setBuyer(buyer);
    transaction.setBook(book);
    transaction.setPrice(99.99);
    transaction.setStatus(TransactionStatus.PROCESSING);
    transaction.setQuantity(3);
    transaction.setIsReviewed(false);
    transaction.setCreatedAt(new Date());
    Transaction savedTransaction = transactionRepository.save(transaction);

    assertThat(savedTransaction).hasFieldOrPropertyWithValue("id", 1L);
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("price", 99.99);
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("status", TransactionStatus.PROCESSING);
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("quantity", 3);
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("isReviewed", false);
    assertThat(savedTransaction.getBook()).hasFieldOrPropertyWithValue("id", 1L);
    assertThat(savedTransaction.getBuyer()).hasFieldOrPropertyWithValue("id", 1L);
}
```

Test: *save_new_transaction()*

Test Result: Pass

Implementation Specification: Tests repository for saving a new transaction.

Implementation Notes: checks to make sure that the transaction is saved and that all the transaction information is correct.

```

// test to find all users after saving a new user
@Test
@Rollback(true)
public void find_all_returns_saved_transactions() {
    Book book = new Book();
    book.setId(1L);
    User buyer = new User();
    buyer.setId(1L);

    Transaction transaction = new Transaction();
    transaction.setId(1L);
    transaction.setBuyer(buyer);
    transaction.setBook(book);
    transaction.setPrice(99.99);
    transaction.setStatus(TransactionStatus.PROCESSING);
    transaction.setQuantity(3);
    transaction.setIsReviewed(false);
    transaction.setCreatedAt(new Date());
    transactionRepository.save(transaction);

    Iterable<Transaction> transactions = transactionRepository.findAll();
    Iterator<Transaction> iter = transactions.iterator();
    Transaction savedTransaction = iter.next();
    while(iter.hasNext() && savedTransaction.getId() != 1L) {
        savedTransaction = iter.next();
    }
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("id", 1L);
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("price", 99.99);
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("status", TransactionStatus.PROCESSING);
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("quantity", 3);
    assertThat(savedTransaction).hasFieldOrPropertyWithValue("isReviewed", false);
    assertThat(savedTransaction.getBook()).hasFieldOrPropertyWithValue("id", 1L);
    assertThat(savedTransaction.getBuyer()).hasFieldOrPropertyWithValue("id", 1L);
}

```

Test: `find_all_returns_saved_transactions()`

Test Result: Pass

Implementation Specification: Tests repository returning all transactions after saving.

Implementation Notes: checks to find the transaction that was saved and make sure it is returned with all transactions.

Build success for transactions

```

Results:
Tests run: 22, Failures: 0, Errors: 0, Skipped: 0

--- maven-jar-plugin:3.2.0:jar (default-jar) @ transactions ---
Building jar: C:\Users\shann.DESKTOP-9P41NT8\Documents\sept\test\SEPT\BackEnd\Transactions\target\transactions-1.0.0.jar

--- spring-boot-maven-plugin:2.3.2.RELEASE:repackage (repackage) @ transactions ---
Replacing main artifact with repackaged archive
-----
BUILD SUCCESS
-----
Total time: 53.596 s

```

Books microservice Tests:

Tester: Jared

BookControllerTest.java

```

114     @Test
115     @DisplayName("Test findAllBooks") // test for getting all books
116     void testfindAllBooks() throws Exception {
117         // Mocking service
118         when(bookService.findAllBooks()).thenReturn(books);
119         mockMvc.perform(get("/api/books/all").contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk())
120             .andExpect(jsonPath("$.id", is(1))).andExpect(jsonPath("$.title", is("Book Title")))
121             .andExpect(jsonPath("$.authorName", is("Author Name"))).andExpect(jsonPath("$.sellerId", is(1)))
122             .andExpect(jsonPath("$.isbn", is(123456))).andExpect(jsonPath("$.quantity", is(10)))
123             .andExpect(jsonPath("$.category", is("Book Category")))
124             .andExpect(jsonPath("$.quality", is(Quality.NEW.toString())))
125             .andExpect(jsonPath("$.price", is(99.99))).andExpect(jsonPath("$.ratingNo", is(0)))
126             .andExpect(jsonPath("$.ratingTotal", is(0)))
127             .andExpect(jsonPath("$.serviceType", is(ServiceType.E_BOOK.toString())))
128
129             .andExpect(jsonPath("[1].id", is(2))).andExpect(jsonPath("[1].title", is("Book Title 2")))
130             .andExpect(jsonPath("[1].authorName", is("Author Name 2"))).andExpect(jsonPath("[1].sellerId", is(1)))
131             .andExpect(jsonPath("[1].isbn", is(1234567))).andExpect(jsonPath("[1].quantity", is(5)))
132             .andExpect(jsonPath("[1].category", is("Book Category 2")))
133             .andExpect(jsonPath("[1].quality", is(Quality.USED.toString())))
134             .andExpect(jsonPath("[1].price", is(89.99))).andExpect(jsonPath("[1].ratingNo", is(1)))
135             .andExpect(jsonPath("[1].ratingTotal", is(1)))
136             .andExpect(jsonPath("[1].serviceType", is(ServiceType.PRINT_ON_DEMAND.toString())));
137
138     }
139
140
141     @Test
142     @DisplayName("Test getBook success") // test for getting a book successfully
143     void testGetBookSuccess() throws Exception {
144         // Mocking service
145         when(bookService.findById(1L)).thenReturn(books.get(0));
146
147         mockMvc.perform(get("/api/books/1").contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk())
148             .andExpect(jsonPath("$.id", is(1))).andExpect(jsonPath("$.title", is("Book Title")))
149             .andExpect(jsonPath("$.authorName", is("Author Name"))).andExpect(jsonPath("$.sellerId", is(1)))
150             .andExpect(jsonPath("$.isbn", is(123456))).andExpect(jsonPath("$.quantity", is(10)))
151             .andExpect(jsonPath("$.category", is("Book Category")))
152             .andExpect(jsonPath("$.quality", is(Quality.NEW.toString()))).andExpect(jsonPath("$.price", is(99.99)))
153             .andExpect(jsonPath("$.ratingNo", is(0))).andExpect(jsonPath("$.ratingTotal", is(0)))
154             .andExpect(jsonPath("$.serviceType", is(ServiceType.E_BOOK.toString())));
155
156
157     @Test
158     @DisplayName("Test getBook not found") // test for getting a book that doesn't exist
159     void testGetBookNotFound() throws Exception {
160         // Mocking service
161         when(bookService.findById(3L)).thenReturn(null);
162
163         RequestBuilder requestBuilder = MockMvcRequestBuilders.get("/api/books/3")
164             .contentType(MediaType.APPLICATION_JSON);
165         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
166
167         MockHttpServletResponse response = result.getResponse();
168
169         assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
170         assertEquals("Book with ID 3 was not found", response.getContentAsString());
171     }
172
173
174     @Test
175     @DisplayName("Test addNewBook success") // test for adding a new book successfully
176     void testAddNewBookSuccess() throws Exception {
177         // Mocking service
178         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
179             .thenReturn(null);
180         when(bookService.saveBook(ArgumentMatchers.any(Book.class))).thenReturn(books.get(0));
181         String inputJson = ("{" + " \"title\":\"Title\",\\n" + " \"authorName\":\"Author Name\",\\n"
182             + " \"sellerId\":\"1\",\\n" + " \"category\":\"Category\",\\n" + " \"isbn\":\"12345678\",\\n"
183             + " \"quantity\":99,\\n" + " \"quality\":\"0\",\\n" + " \"bookStatus\":\"0\",\\n"
184             + " \"serviceType\":\"PRINT_ON_DEMAND\",\\n" + " \"price\":\"99.99\"\\n" + "}");
185         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/books/new").accept(MediaType.APPLICATION_JSON)
186             .content(inputJson).contentType(MediaType.APPLICATION_JSON);
187
188         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
189         MockHttpServletResponse response = result.getResponse();
190
191         assertEquals(HttpStatus.OK.value(), response.getStatus());
192     }
193
194     @Test
195     @DisplayName("Test addNewBook null seller error") // test for adding a new book without a seller
196     void testAddNewBookNullSellerError() throws Exception {
197         // Mocking service
198         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
199             .thenReturn(null);
200         when(bookService.saveBook(ArgumentMatchers.any(Book.class))).thenReturn(null);
201         String inputJson = ("{" + " \"title\":\"Book Title\",\\n" + " \"authorName\":\"Author Name\",\\n"
202             + " \"category\":\"Book Category\",\\n" + " \"isbn\":\"123456\",\\n" + " \"quantity\":10,\\n"
203             + " \"quality\":0,\\n" + " \"bookStatus\":0,\\n" + " \"serviceType\":E_BOOK,\\n"
204             + " \"price\":99.99\"\\n" + "}");
205         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/books/new").accept(MediaType.APPLICATION_JSON)
206             .content(inputJson).contentType(MediaType.APPLICATION_JSON);
207
208         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
209         MockHttpServletResponse response = result.getResponse();
210
211         assertEquals(HttpStatus.NOT_ACCEPTABLE.value(), response.getStatus());
212         assertEquals("Unable to add the new book, User id not given!.", response.getContentAsString());
213     }

```

```

214
215     @Test
216     @DisplayName("Test addNewBook duplicate error") // test for adding a duplicate new book
217     void testAddNewBookDuplicateError() throws Exception {
218         // Mocking service
219         when(mapValidationErrorService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
220             .thenReturn(null);
221         when(bookService.saveBook(ArgumentMatchers.any(Book.class))).thenReturn(null);
222         String inputJson = "\n" + " \\'title\'\":\"Book Title\",\\n" + " \\'authorName\'\":\"Author Name\",\\n"
223             + " \\'sellerId\'\":\"1\",\\n" + " \\'category\'\":\"Book Category\",\\n" + " \\'isbn\'\":\"123456\",\\n"
224             + " \\'quantity\'\":\"10\",\\n" + " \\'quality\'\":\"0\",\\n" + " \\'bookStatus\'\":\"0\",\\n"
225             + " \\'serviceType\'\":\"E_BOOK\",\\n" + " \\'price\'\":\"99.99\\n\" + ")";
226         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/books/new").accept(MediaType.APPLICATION_JSON)
227             .content(inputJson).contentType(MediaType.APPLICATION_JSON);
228
229         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
230         MockHttpServletResponse response = result.getResponse();
231
232         assertEquals(HttpStatus.CONFLICT.value(), response.getStatus());
233         assertEquals("Unable to add the new book, a copy of the book already exists.", response.getContentAsString());
234     }
235
236     @Test
237     @DisplayName("Test addNewBook badRequest") // test for adding a new book with invalid information
238     void testAddNewBookBadRequest() throws Exception {
239         // Mocking service
240         when(mapValidationErrorService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
241             .thenReturn(null);
242
243         String inputJson = "null";
244         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/books/new").accept(MediaType.APPLICATION_JSON)
245             .content(inputJson).contentType(MediaType.APPLICATION_JSON);
246
247         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
248         MockHttpServletResponse response = result.getResponse();
249
250         assertEquals(HttpStatus.BAD_REQUEST.value(), response.getStatus());
251     }

```

```

252
253     @Test
254     @DisplayName("Test deleteBook success") // test for successfully deleting a book
255     void testDeleteBookSuccess() throws Exception {
256         // Mocking service
257         when(bookService.findById(1L)).thenReturn(books.get(0));
258
259         RequestBuilder requestBuilder = MockMvcRequestBuilders.delete("/api/books/1");
260
261         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
262         MockHttpServletResponse response = result.getResponse();
263
264         assertEquals(HttpStatus.OK.value(), response.getStatus());
265         assertEquals("Book with ID 1 was deleted", response.getContentAsString());
266     }
267
268     @Test
269     @DisplayName("Test deleteBook not found") // test for deleting a book that doesn't exist
270     void testDeleteBookNotFound() throws Exception {
271         // Mocking service
272         when(bookService.findById(3L)).thenReturn(null);
273
274         RequestBuilder requestBuilder = MockMvcRequestBuilders.delete("/api/books/3");
275
276         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
277         MockHttpServletResponse response = result.getResponse();
278
279         assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
280         assertEquals("Book with ID 3 was not found", response.getContentAsString());
281     }

```

EditBookControllerTest.java

```

107
108
109 @Test
110 @DisplayName("Test editBook success") // test for editing a book successfully
111 void testEditBookSuccess() throws Exception {
112     // Mocking service
113     when(bookService.findById(1L)).thenReturn(books.get(0));
114     when(editBookService.updateBook(ArgumentMatchers.any(BookForm.class), ArgumentMatchers.any(Book.class)))
115         .thenReturn(books.get(0));
116
117     String inputJson = "{\n" + "\"sellerId\":\"1\"\n" + "\"quality\":\"NEW\"\n" + "}";
118     RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editBook/1")
119         .accept(MediaType.APPLICATION_JSON).content(inputJson)
120         .contentType(MediaType.APPLICATION_JSON);
121
122     MvcResult result = mockMvc.perform(requestBuilder).andReturn();
123     MockHttpServletResponse response = result.getResponse();
124
125     assertEquals(HttpStatus.OK.value(), response.getStatus());
126     assertEquals("Successfully updated book details", response.getContentAsString());
127 }
128
129 @Test
130 @DisplayName("Test editBook null seller error") // test for editing a book so it is sold by an invalid seller
131 void testEditBookNullSellerError() throws Exception {
132     // Mocking service
133     when(bookService.findById(1L)).thenReturn(books.get(0));
134     String inputJson = "{\n" + "\"quality\":\"NEW\"\n" + "}";
135     RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editBook/1")
136         .accept(MediaType.APPLICATION_JSON).content(inputJson)
137         .contentType(MediaType.APPLICATION_JSON);
138
139     MvcResult result = mockMvc.perform(requestBuilder).andReturn();
140     MockHttpServletResponse response = result.getResponse();
141
142     assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
143     assertEquals("Unable to add the new book details, User to tie to not found!", response.getContentAsString());
144 }
145
146 @Test
147 @DisplayName("Test editBook duplicate error") // test for editing a book so it becomes a duplicate copy of
148 // another book
149 void testEditBookDuplicateError() throws Exception {
150     // Mocking service
151
152     when(bookService.findById(1L)).thenReturn(books.get(0));
153     when(editBookService.updateBook(ArgumentMatchers.any(BookForm.class), ArgumentMatchers.any(Book.class)))
154         .thenReturn(null);
155     String inputJson = "{\n" + "\"title\":\"Book Title\",\\n" + "\"authorName\":\"Author Name\",\\n"
156         + " \\\"sellerId\":\"1\",\\n" + "\\\"category\":\"Book Category\",\\n"
157         + " \\\"isbn\":\"123456\",\\n" + "\\\"quantity\":\"10\",\\n" + "\\\"quality\":\"0\",\\n"
158         + " \\\"bookStatus\":\"0\",\\n" + "\\\"serviceType\":\"E_BOOK\",\\n"
159         + " \\\"price\":\"99.99\"\\n" + "}";
160     RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editBook/1")
161         .accept(MediaType.APPLICATION_JSON).content(inputJson)
162         .contentType(MediaType.APPLICATION_JSON);
163
164     MvcResult result = mockMvc.perform(requestBuilder).andReturn();
165     MockHttpServletResponse response = result.getResponse();
166
167     assertEquals(HttpStatus.CONFLICT.value(), response.getStatus());
168     assertEquals("Unable to save details for book, a copy of the book already exists.", response.getContentAsString());
169 }
170
171 @Test
172 @DisplayName("Test editBook not found") // test for editing a book that doesn't exist
173 void testEditBookNotFound() throws Exception {
174     // Mocking service
175     when(userService.findById(3L)).thenReturn(null);
176
177     String inputJson = "{\n" + "\\\"sellerId\":\"1\",\\n" + "\\\"quality\":\"NEW\"\n" + "}";
178     RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editBook/3")
179         .accept(MediaType.APPLICATION_JSON).content(inputJson)
180         .contentType(MediaType.APPLICATION_JSON);
181
182     MvcResult result = mockMvc.perform(requestBuilder).andReturn();
183     MockHttpServletResponse response = result.getResponse();
184
185     assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
186     assertEquals("Book with ID 3 was not found", response.getContentAsString());
187 }

```

BookRepositoryTest.java

```
34     // testing for empty book repository
35
36     @Test
37     @Rollback(true)
38     public void should_find_no_books_if_bookRepository_is_empty() {
39         Iterable<Book> books = bookRepository.findAll();
40         if (books.iterator().hasNext()) {
41             assertThat(books).isNotEmpty();
42         } else {
43             assertThat(books).isEmpty();
44         }
45     }
46
47     // test to find a book by id
48     @Test
49     @Rollback(true)
50     public void find_book_by_id() {
51         Book book1 = new Book();
52         book1.setId(1L);
53         book1.setTitle("Book Title");
54         book1.setAuthorName("Author Name");
55         book1.setISBN(123456);
56         book1.setQuantity(10);
57         book1.setCategory("Book Category");
58         book1.setQuality(Quality.NEW);
59         book1.setPrice(99.99);
60         book1.setRatingNo(0);
61         book1.setRatingTotal(0);
62         book1.setServiceType(ServiceType.E_BOOK);
63         bookRepository.save(book1);
64
65         Book findBook = bookRepository.findById(1L).orElse(null);
66         assertNotNull(findBook);
67         assertThat(findBook).hasFieldOrPropertyWithValue("id", 1L);
68         assertThat(findBook).hasFieldOrPropertyWithValue("title", "Book Title");
69         assertThat(findBook).hasFieldOrPropertyWithValue("authorName", "Author Name");
70         assertThat(findBook).hasFieldOrPropertyWithValue("isbn", 123456);
71         assertThat(findBook).hasFieldOrPropertyWithValue("quantity", 10);
72         assertThat(findBook).hasFieldOrPropertyWithValue("category", "Book Category");
73         assertThat(findBook).hasFieldOrPropertyWithValue("quality", Quality.NEW);
74         assertThat(findBook).hasFieldOrPropertyWithValue("price", 99.99);
75         assertThat(findBook).hasFieldOrPropertyWithValue("ratingNo", 0);
76         assertThat(findBook).hasFieldOrPropertyWithValue("ratingTotal", 0);
77         assertThat(findBook).hasFieldOrPropertyWithValue("serviceType", ServiceType.E_BOOK);
78     }

```

```

75
79 // test to delete a book
80 @Test
81 @Rollback(true)
82 public void delete_book() {
83     Book book1 = new Book();
84     book1.setId(0L);
85     book1.setTitle("Book Title");
86     book1.setAuthorName("Author Name");
87     book1.setISBN(123456);
88     book1.setQuantity(10);
89     book1.setCategory("Book Category");
90     book1.setQuality(Quality.NEW);
91     book1.setPrice(99.99);
92     book1.setRatingNo(0);
93     book1.setRatingTotal(0);
94     book1.setServiceType(ServiceType.E_BOOK);
95     bookRepository.save(book1);
96     bookRepository.delete(book1);
97     Book findBook = bookRepository.findById(0L).orElse(null);
98     assertNull(findBook);
99 }
100
101 // test to save a new book
102 @Test
103 @Rollback(true)
104 public void save_new_book() {
105     Book book1 = new Book();
106     book1.setId(1L);
107     book1.setTitle("Book Title");
108     book1.setAuthorName("Author Name");
109     book1.setISBN(123456);
110     book1.setQuantity(10);
111     book1.setCategory("Book Category");
112     book1.setQuality(Quality.NEW);
113     book1.setPrice(99.99);
114     book1.setRatingNo(0);
115     book1.setRatingTotal(0);
116     book1.setServiceType(ServiceType.E_BOOK);
117     bookRepository.save(book1);
118
119     Book findBook = bookRepository.findById(1L).orElse(null);
120     assertNotNull(findBook);
121 }
122

```

```

123 // test to see if a book exists with valid params
124 @Test
125 @Rollback(true)
126 public void book_exists_returns_true_if_book_exists() {
127     Book book1 = new Book();
128     book1.setId(1L);
129     book1.setTitle("Book Title");
130     book1.setAuthorName("Author Name");
131     book1.setISBN(123456);
132     book1.setQuantity(10);
133     book1.setCategory("Book Category");
134     book1.setQuality(Quality.NEW);
135     book1.setPrice(99.99);
136     book1.setRatingNo(0);
137     book1.setRatingTotal(0);
138     book1.setServiceType(ServiceType.E_BOOK);
139     book1.setBookStatus(BookStatus.AVAILABLE);
140     User seller = new User();
141     seller.setId(1L);
142     book1.setSeller(seller);
143     Request request = new Request();
144     request.setId(1L);
145     book1.setRequest(request);
146     bookRepository.save(book1);
147
148     boolean newbookExists = bookRepository.bookExists(seller, "Book Title".toLowerCase(),
149             "Author Name".toLowerCase(), "Book Category".toLowerCase(), 123456, Quality.NEW);
150     assertEquals(true, newbookExists);
151 }
152

```

```

152
153 // test to see if a book exists with invalid params
154 @Test
155 @Rollback(true)
156 public void book_exists_returns_false_if_book_does_not_exist() {
157     Book book1 = new Book();
158     book1.setId(1L);
159     book1.setTitle("Book Title");
160     book1.setAuthorName("Author Name");
161     book1.setISBN(123456);
162     book1.setQuantity(10);
163     book1.setCategory("Book Category");
164     book1.setQuality(Quality.NEW);
165     book1.setPrice(99.99);
166     book1.setRatingNo(0);
167     book1.setRatingTotal(0);
168     book1.setServiceType(ServiceType.E_BOOK);
169     book1.setBookStatus(BookStatus.AVAILABLE);
170     User seller = new User();
171     seller.setId(1L);
172     book1.setSeller(seller);
173     Request request = new Request();
174     request.setId(1L);
175     book1.setRequest(request);
176     bookRepository.save(book1);
177
178     boolean newbookExists = bookRepository.bookExists(seller, "Wrong Book Title".toLowerCase(),
179             "Author Name".toLowerCase(), "Book Category".toLowerCase(), 123456, Quality.NEW);
180     assertEquals(false, newbookExists);
181 }
182

183 // test to find a book with valid params
184 @Test
185 @Rollback(true)
186 public void find_with_params_returns_correct_book_if_book_exists() {
187     Book book1 = new Book();
188     book1.setId(1L);
189     book1.setTitle("Book Title");
190     book1.setAuthorName("Author Name");
191     book1.setISBN(123456);
192     book1.setQuantity(10);
193     book1.setCategory("Book Category");
194     book1.setQuality(Quality.NEW);
195     book1.setPrice(99.99);
196     book1.setRatingNo(0);
197     book1.setRatingTotal(0);
198     book1.setServiceType(ServiceType.E_BOOK);
199     book1.setBookStatus(BookStatus.AVAILABLE);
200     User seller = new User();
201     seller.setId(1L);
202     book1.setSeller(seller);
203     Request request = new Request();
204     request.setId(1L);
205     book1.setRequest(request);
206     bookRepository.save(book1);
207
208     Book findBook = bookRepository.findWithParams(seller, "Book Title".toLowerCase(), "Author Name".toLowerCase(),
209             "Book Category".toLowerCase(), 123456, Quality.NEW);
210     assertEquals(book1.getTitle(), findBook.getTitle());
211     assertEquals(book1.getAuthorName(), findBook.getAuthorName());
212     assertEquals(book1.getISBN(), findBook.getISBN());
213     assertEquals(book1.getQuantity(), findBook.getQuantity());
214     assertEquals(book1.getCategory(), findBook.getCategory());
215     assertEquals(book1.getQuality(), findBook.getQuality());
216     assertEquals(book1.getPrice(), findBook.getPrice());
217     assertEquals(book1.getRatingNo(), findBook.getRatingNo());
218     assertEquals(book1.getRatingTotal(), findBook.getRatingTotal());
219     assertEquals(book1.getServiceType(), findBook.getServiceType());
220     assertEquals(book1.getBookStatus(), findBook.getBookStatus());
221     assertEquals(book1.getSeller().getId(), findBook.getSeller().getId());
222 }
223

224 // test to find a book with valid params
225 @Test
226 @Rollback(true)
227 public void find_with_params_returns_null_book_if_book_does_not_exist() {
228     User seller = new User();
229     seller.setId(1L);
230
231     Book findBook = bookRepository.findWithParams(seller, "Book Title".toLowerCase(), "Author Name".toLowerCase(),
232             "Book Category".toLowerCase(), 123456, Quality.NEW);
233     assertNull(findBook);
234
235 }
236 }
```

BookServiceTest.java

```

75
76     @Test
77     @DisplayName("Test findAllBooks success") // test for finding all books
78     public void testfindAllBooksSuccess() throws Exception {
79         given(bookRepository.findAll()).willReturn(books);
80         Iterable<Book> allBooks = bookService.findAllBooks();
81         Assert.assertNotNull(allBooks);
82     }
83
84     @Test
85     @DisplayName("Test findBySeller success") // test for finding all books by seller successfully
86     public void testFindBySellerSuccess() throws Exception {
87         User user = new User();
88         given(bookRepository.findBySeller(user)).willReturn(books);
89         Iterable<Book> allBooks = bookService.getAllBySeller(user);
90         Assert.assertNotNull(allBooks);
91     }
92
93     @Test
94     @DisplayName("Test findBySeller fail") // test for finding all books by seller which doesn't exist
95     public void testFindBySellerFail() throws Exception {
96         given(bookRepository.findBySeller(null)).willReturn(null);
97         Iterable<Book> allBooks = bookService.getAllBySeller(null);
98         Assert.assertNull(allBooks);
99     }

```

Browsing microservice Tests:

Tester: Jared

BrowsingControllerTest.java

```

149
150
151     @Test
152     @DisplayName("Test getByTitle") // test for getting by book title
153     void testGetByTitle() throws Exception {
154         // Mocking service
155         when(browsingService.findAllByTitle("title")).thenReturn(books);
156         mockMvc.perform(get("/api/browse/title/title").contentType(MediaType.APPLICATION_JSON))
157             .andExpect(status().isOk()).andExpect(jsonPath("$.0.title", is("Book Title")))
158             .andExpect(jsonPath("$.1.title", is("Book Title 2")));
159
160     @Test
161     @DisplayName("Test getByAuthorName") // test for getting by author name
162     void testGetByAuthorName() throws Exception {
163         // Mocking service
164         when(browsingService.findAllByAuthorName("author")).thenReturn(books);
165         mockMvc.perform(get("/api/browse/authorName/author").contentType(MediaType.APPLICATION_JSON))
166             .andExpect(status().isOk()).andExpect(jsonPath("$.0.authorName", is("Author Name")))
167             .andExpect(jsonPath("$.1.authorName", is("Author Name 2")));
168
169     @Test
170     @DisplayName("Test getBySeller") // test for getting by seller
171     void testGetBySeller() throws Exception {
172         // Mocking service
173         when(browsingService.findAllBySeller(ArgumentMatchers.any(User.class))).thenReturn(newBooks);
174         mockMvc.perform(get("/api/browse/sellerId/1").contentType(MediaType.APPLICATION_JSON))
175             .andExpect(status().isOk())
176             .andExpect(jsonPath("$.0.quality", is(Quality.NEW.toString())));
177     }
178
179     @Test
180     @DisplayName("Test getByCategory") // test for getting by category
181     void testGetByCategory() throws Exception {
182         // Mocking service
183         when(browsingService.findAllByCategory("category")).thenReturn(books);
184         mockMvc.perform(get("/api/browse/category/category").contentType(MediaType.APPLICATION_JSON))
185             .andExpect(status().isOk()).andExpect(jsonPath("$.0.category", is("Book Category")))
186             .andExpect(jsonPath("$.1.category", is("Book Category 2")));
187     }
188

```

```

189  @Test
190  @DisplayName("Test getByISBN") // test for getting by isbn
191  void testGetByISBN() throws Exception {
192      // Mocking service
193      when(browsingService.findAllByISBN(123)).thenReturn(books);
194      mockMvc.perform(get("/api/browse/isbn/123").contentType(MediaType.APPLICATION_JSON))
195          .andExpect(status().isOk()).andExpect(jsonPath("$.isbn", is(123456)))
196          .andExpect(jsonPath("$.isbn", is(1234567)));
197  }
198
199
200  @Test
201  @DisplayName("Test getNewBooks") // test for getting all new books
202  void testGetNewBooks() throws Exception {
203      // Mocking service
204      when(browsingService.findAllNewBooks()).thenReturn(books);
205      mockMvc.perform(get("/api/browse/new").contentType(MediaType.APPLICATION_JSON))
206          .andExpect(status().isOk())
207          .andExpect(jsonPath("$.quality", is(Quality.NEW.toString())));
208  }
209
210  @Test
211  @DisplayName("Test getUsedBooks") // test for getting all used books
212  void testGetUsedBooks() throws Exception {
213      // Mocking service
214      when(browsingService.findAllUsedBooks()).thenReturn(usedBooks);
215      mockMvc.perform(get("/api/browse/used").contentType(MediaType.APPLICATION_JSON))
216          .andExpect(status().isOk())
217          .andExpect(jsonPath("$.quality", is(QualityUSED.toString())));
218  }
219
220  @Test
221  @DisplayName("Test sortByPriceHighToLow") // test for getting books sorted from highest to lowest price
222  void testSortByPriceHighToLow() throws Exception {
223      // Mocking service
224      when(browsingService.sortByHighestPrice()).thenReturn(books);
225      mockMvc.perform(get("/api/browse/price/high").contentType(MediaType.APPLICATION_JSON))
226          .andExpect(status().isOk()).andExpect(jsonPath("$.price", is(99.99)))
227          .andExpect(jsonPath("$.price", is(89.99)));
228  }
229
230  @Test
231  @DisplayName("Test sortByPriceLowToHigh") // test for getting all sorted from lowest to highest price
232  void testSortByPriceLowToHigh() throws Exception {
233      // Mocking service
234      when(browsingService.sortByLowestPrice()).thenReturn(books2);
235      mockMvc.perform(get("/api/browse/price/low").contentType(MediaType.APPLICATION_JSON))
236          .andExpect(status().isOk()).andExpect(jsonPath("$.price", is(89.99)))
237          .andExpect(jsonPath("$.price", is(99.99)));
238  }
239
240  @Test
241  @DisplayName("Test sortByAlphabet") // test for getting all sorted in alphabetical title order
242  void testSortByAlphabet() throws Exception {
243      // Mocking service
244      when(browsingService.sortByAlphabet()).thenReturn(books);
245      mockMvc.perform(get("/api/browse/alphabet").contentType(MediaType.APPLICATION_JSON))
246          .andExpect(jsonPath("$.title", is("Book Title")))
247          .andExpect(jsonPath("$.title", is("Book Title 2")));
248  }
249
250  @Test
251  @DisplayName("Test bestSellers") // test for getting all sorted by highest rating
252  void testBestSellers() throws Exception {
253      // Mocking service
254      when(browsingService.sortByHighestRating(2)).thenReturn(books2);
255      mockMvc.perform(get("/api/browse/bestsellers/2").contentType(MediaType.APPLICATION_JSON))
256          .andExpect(jsonPath("$.ratingTotal", is(1)))
257          .andExpect(jsonPath("$.ratingTotal", is(0)));
258  }
259
260  @Test
261  @DisplayName("Test random") // test for getting two random
262  void testRandom() throws Exception {
263      // Mocking service
264      when(browsingService.random(2)).thenReturn(books2);
265      mockMvc.perform(get("/api/browse/random/2").contentType(MediaType.APPLICATION_JSON))
266          .andExpect(jsonPath("$.id", is(2))).andExpect(jsonPath("$.id", is(1)));
267  }
268

```

BookRepositoryTest.java

```
30 // testing for browsing when books have a given title
31 @Test
32 @Rollback(true)
33 public void test_if_books_exist_with_title() {
34     Iterable<Book> books = bookRepository.findByTitle("peril");
35     assertThat(books).isNotEmpty();
36 }
37
38 // testing for browsing when no books have a given title
39 @Test
40 @Rollback(true)
41 public void test_empty_result_if_no_books_with_title() {
42     Iterable<Book> books = bookRepository.findByTitle("TestTitle");
43     assertThat(books).isEmpty();
44 }
45
46 // testing for browsing when books have a given author's name
47 @Test
48 @Rollback(true)
49 public void test_if_books_exist_with_author_name() {
50     Iterable<Book> books = bookRepository.findByAuthorName("dave");
51     assertThat(books).isNotEmpty();
52 }
53
54 // testing for browsing when no books have a given author's name
55 @Test
56 @Rollback(true)
57 public void test_empty_result_if_no_books_with_author_name() {
58     Iterable<Book> books = bookRepository.findByAuthorName("authorName");
59     assertThat(books).isEmpty();
60 }
61
62 // testing for browsing when books have a given category
63 @Test
64 @Rollback(true)
65 public void test_if_books_exist_with_category() {
66     Iterable<Book> books = bookRepository.findByCategory("business");
67     assertThat(books).isNotEmpty();
68 }
69
```

```
70 // testing for browsing when no books have a given category
71 @Test
72 @Rollback(true)
73 public void test_empty_result_if_no_books_with_category() {
74     Iterable<Book> books = bookRepository.findByCategory("TestCategory");
75     assertThat(books).isEmpty();
76 }
77
78 // testing for browsing when books have a given isbn
79 @Test
80 @Rollback(true)
81 public void test_if_books_exist_with_isbn() {
82     Iterable<Book> books = bookRepository.findByisbn("198218291");
83     assertThat(books).isNotEmpty();
84 }
85
86 // testing for browsing when no books have a given isbn
87 @Test
88 @Rollback(true)
89 public void test_empty_result_if_no_books_with_isbn() {
90     Iterable<Book> books = bookRepository.findByisbn("falseisbn");
91     assertThat(books).isEmpty();
92 }
93
94 // testing for browsing when books have a given quality - new
95 @Test
96 @Rollback(true)
97 public void test_if_books_exist_with_quality_new() {
98     Iterable<Book> books = bookRepository.findByQuality(Quality.NEW);
99     Iterator<Book> iter = books.iterator();
100    while (iter.hasNext()) {
101        Book book = iter.next();
102        assertThat(book.getQuality()).isEqualTo(Quality.NEW);
103    }
104 }
105
106 // testing for browsing when books have a given quality - used
107 @Test
108 @Rollback(true)
109 public void test_if_books_exist_with_quality_used() {
110     Iterable<Book> books = bookRepository.findByQuality(Quality.USED);
111     Iterator<Book> iter = books.iterator();
112     while (iter.hasNext()) {
113         Book book = iter.next();
114         assertThat(book.getQuality()).isEqualTo(Quality.USED);
115     }
116 }
117 }
```

```

118 // testing for browsing by books sorted by highest price
119 @Test
120 @Rollback(true)
121 public void test_sort_by_highest_price() {
122     Iterable<Book> books = bookRepository.sortByHighestPrice();
123     Iterator<Book> iter = books.iterator();
124     double maxPrice = Integer.MAX_VALUE;
125     while (iter.hasNext()) {
126         Book book = iter.next();
127         double price = book.getPrice();
128         assertThat(price <= maxPrice).isTrue();
129         maxPrice = price;
130     }
131 }
132
133 // testing for browsing by books sorted by lowest price
134 @Test
135 @Rollback(true)
136 public void test_sort_by_lowest_price() {
137     Iterable<Book> books = bookRepository.sortByLowestPrice();
138     Iterator<Book> iter = books.iterator();
139     double minPrice = Integer.MIN_VALUE;
140     while (iter.hasNext()) {
141         Book book = iter.next();
142         double price = book.getPrice();
143         assertThat(price >= minPrice).isTrue();
144         minPrice = price;
145     }
146 }
147
148 // testing for browsing by books sorted by alphabetical title order
149 @Test
150 @Rollback(true)
151 public void test_sort_by_alphabet() {
152     Iterable<Book> books = bookRepository.sortByAlphabet();
153     Iterator<Book> iter = books.iterator();
154     String prevTitle = "a";
155     while (iter.hasNext()) {
156         Book book = iter.next();
157         String title = book.getTitle();
158         assertThat(prevTitle.compareTo(title)).isGreaterThan(-1);
159         title = prevTitle;
160     }
161 }
162
163 // testing for browsing by books sorted by newest release
164 @Test
165 @Rollback(true)
166 public void test_sort_by_newest_release() {
167     Iterable<Book> books = bookRepository.sortByNewestRelease(10);
168     Iterator<Book> iter = books.iterator();
169     Date newestDate = new Date();
170     while (iter.hasNext()) {
171         Book book = iter.next();
172         Date date = book.getCreated_At();
173         assertThat(newestDate.compareTo(date)).isGreaterThan(-1);
174         newestDate = date;
175     }
176 }
177
178 // testing for browsing by books sorted by highest rating
179 @Test
180 @Rollback(true)
181 public void test_sort_by_highest_rating() {
182     Iterable<Book> books = bookRepository.sortByHighestRating(10);
183     Iterator<Book> iter = books.iterator();
184     double maxRating = 5.0;
185     while (iter.hasNext()) {
186         Book book = iter.next();
187         double rating = book.getRatingTotal()/book.getRatingNo();
188         assertThat(maxRating >= rating).isTrue();
189         maxRating = rating;
190     }
191 }
192 }
193

```

BrowsingServiceTest.java

```
132  @Test
133  @DisplayName("Test findByTitle") // test for finding by title
134  public void testFindByTitle() throws Exception {
135      given(bookRepository.findByTitle("title")).willReturn(books);
136      Iterable<Book> books = browsingService.findAllByTitle("title");
137      Assert.assertNotNull(books);
138  }
139
140
141  @Test
142  @DisplayName("Test findByAuthorName") // test for finding by author name
143  public void testFindByAuthorName() throws Exception {
144      given(bookRepository.findByAuthorName("author")).willReturn(books);
145      Iterable<Book> books = browsingService.findAllByTitle("author");
146      Assert.assertNotNull(books);
147  }
148
149
150  @Test
151  @DisplayName("Test findByisbn") // test for finding by isbn
152  public void testFindByisbn() throws Exception {
153      given(bookRepository.findByisbn("123456")).willReturn(books);
154      Iterable<Book> books = browsingService.findAllByISBN(123456);
155      Assert.assertNotNull(books);
156  }
157
158  @Test
159  @DisplayName("Test findByCategory") // test for finding by category
160  public void testFindByCategory() throws Exception {
161      given(bookRepository.findByCategory("book")).willReturn(books);
162      Iterable<Book> books = browsingService.findAllByTitle("book");
163      Assert.assertNotNull(books);
164  }
165
166  @Test
167  @DisplayName("Test findAllNewBooks") // test for finding all new books
168  public void testfindAllNewBooks() throws Exception {
169      given(bookRepository.findByQuality(Quality.NEW)).willReturn(newBooks);
170      Iterable<Book> books = browsingService.findAllNewBooks();
171      Iterator<Book> iter = books.iterator();
172      while (iter.hasNext()) {
173          Book book = iter.next();
174          assertThat(book.getQuality()).isEqualTo(Quality.NEW);
175      }
176  }
```

```

176     @Test
177     @DisplayName("Test findAllUsedBooks") // test for finding all used books
178     public void testfindAllUsedBooks() throws Exception {
179         given(bookRepository.findByQuality(Quality.USED)).willReturn(usedBooks);
180         Iterable<Book> books = browsingService.findAllUsedBooks();
181         Iterator<Book> iter = books.iterator();
182         while (iter.hasNext()) {
183             Book book = iter.next();
184             assertThat(book.getQuality()).isEqualTo(Quality.USED);
185         }
186     }
187
188     @Test
189     @DisplayName("Test sortByHighestPrice") // test for sorting by highest price
190     public void testSortByHighestPrice() throws Exception {
191         given(bookRepository.sortByHighestPrice()).willReturn(books);
192         Iterable<Book> books = browsingService.sortByHighestPrice();
193         Iterator<Book> iter = books.iterator();
194         double maxPrice = Integer.MAX_VALUE;
195         while (iter.hasNext()) {
196             Book book = iter.next();
197             double price = book.getPrice();
198             assertThat(price <= maxPrice).isTrue();
199             maxPrice = price;
200         }
201     }
202
203     @Test
204     @DisplayName("Test sortByLowestPrice") // test for sorting by lowest price
205     public void testSortByLowestPrice() throws Exception {
206         given(bookRepository.sortByHighestPrice()).willReturn(books2);
207         Iterable<Book> books = browsingService.sortByLowestPrice();
208         Iterator<Book> iter = books.iterator();
209         double minPrice = Integer.MIN_VALUE;
210         while (iter.hasNext()) {
211             Book book = iter.next();
212             double price = book.getPrice();
213             assertThat(price >= minPrice).isTrue();
214             minPrice = price;
215         }
216     }
217

```

```

218     @Test
219     @DisplayName("Test sortByAlphabet") // test for sorting by alphabetical title order
220     public void testSortByAlphabet() throws Exception {
221         given(bookRepository.sortByAlphabet()).willReturn(books);
222         Iterable<Book> books = bookRepository.sortByAlphabet();
223         Iterator<Book> iter = books.iterator();
224         String prevTitle = "a";
225         while (iter.hasNext()) {
226             Book book = iter.next();
227             String title = book.getTitle();
228             assertThat(prevTitle.compareTo(title)).isGreaterThan(-1);
229             title = prevTitle;
230         }
231     }
232
233     @Test
234     @DisplayName("Test sortByNewestRelease") // testing for browsing by books sorted by newest release
235     public void testSortByNewestRelease() {
236         Iterable<Book> books = bookRepository.sortByNewestRelease(10);
237         Iterator<Book> iter = books.iterator();
238         Date newestDate = new Date();
239         while (iter.hasNext()) {
240             Book book = iter.next();
241             Date date = book.getCreated_At();
242             assertThat(newestDate.compareTo(date)).isGreaterThan(-1);
243             newestDate = date;
244         }
245     }
246
247     @Test
248     @DisplayName("Test sortByHighestRating") // testing for browsing by books sorted by highest rating
249     public void testSortByHighestRating() {
250         Iterable<Book> books = bookRepository.sortByHighestRating(10);
251         Iterator<Book> iter = books.iterator();
252         double maxRating = 5.0;
253         while (iter.hasNext()) {
254             Book book = iter.next();
255             double rating = book.getRatingTotal()/book.getRatingNo();
256             assertThat(maxRating >= rating).isTrue();
257             maxRating = rating;
258         }
259     }
260 }

```

Login microservice Tests:

Tester: Jared

LoginControllerTest.java

```
86  @Test
87  @DisplayName("Test login success") // test for logging in successfully
88  void testLoginSuccess() throws Exception {
89      // Mocking service
90      when(mockValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
91          .thenReturn(null);
92
93      String inputJson = "{\n" + "    \"username\":\"JohnDoe\",\\n" + "    \"password\":\"password\"\n" + "}";
94      RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/users/login")
95          .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
96
97      MvcResult result = mockMvc.perform(requestBuilder).andExpect(jsonPath("success", is(true))).andReturn();
98      MockHttpServletResponse response = result.getResponse();
99
100     assertEquals(HttpStatus.OK.value(), response.getStatus());
101 }
102
103 @Test
104 @DisplayName("Test login failure") // test for failing to log in with incorrect user information
105 void testLoginFailure() throws Exception {
106     // Mocking service
107     String inputJson = "{\n" + "    \"username\":\"WrongUsername\",\\n" + "    \"password\":\"password\"\n" + "}";
108     RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/users/login")
109         .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
110     MvcResult result = mockMvc.perform(requestBuilder).andExpect(jsonPath("token", is("Bearer null"))).andReturn();
111     MockHttpServletResponse response = result.getResponse();
112
113     assertEquals(HttpStatus.OK.value(), response.getStatus());
114 }
115
116 }
117 }
```

Requests microservice Tests:

Tester: Jared

RequestsControllerTest.java

```
136  @Test
137  @DisplayName("Test findAllRequests") // test for getting all requests
138  void testFindAllRequests() throws Exception {
139      // Mocking service
140      when(requestService.findAllRequests()).thenReturn(requests);
141      mockMvc.perform(get("/api/requests/all").contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk())
142          .andExpect(jsonPath("$.id", is(1))).andExpect(jsonPath("$.objectId", is(1)))
143          .andExpect(jsonPath("$.request", is("New Book"))).andExpect(jsonPath("$.requestType", is(RequestType.NEW_BOOK_LISTING.toString())))
144          .andExpect(jsonPath("$.user.id", is(1)));
145
146          .andExpect(jsonPath("$.id", is(2))).andExpect(jsonPath("$.objectId", is(2)))
147          .andExpect(jsonPath("$.request", is("New Book"))).andExpect(jsonPath("$.requestType", is(RequestType.NEW_BOOK_LISTING.toString())))
148          .andExpect(jsonPath("$.user.id", is(1)));
149
150 }
151
152 @Test
153 @DisplayName("Test getRequest success") // test for getting a request successfully
154 void testGetRequestSuccess() throws Exception {
155     // Mocking service
156     when(requestService.findById(1L)).thenReturn(requests.get(0));
157     mockMvc.perform(get("/api/requests/1").contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk())
158         .andExpect(jsonPath("$.id", is(1))).andExpect(jsonPath("$.objectId", is(1)))
159         .andExpect(jsonPath("$.request", is("New Book")))
160         .andExpect(jsonPath("$.requestType", is(RequestType.NEW_BOOK_LISTING.toString())))
161         .andExpect(jsonPath("$.user.id", is(1)));
162 }
163
164 @Test
165 @DisplayName("Test getRequest not found") // test for getting a request that doesn't exist
166 void testGetRequestNotFound() throws Exception {
167     // Mocking service
168     when(requestService.findById(0L)).thenReturn(null);
169
170     RequestBuilder requestBuilder = MockMvcRequestBuilders.get("/api/requests/0")
171         .contentType(MediaType.APPLICATION_JSON);
172     MvcResult result = mockMvc.perform(requestBuilder).andReturn();
173     MockHttpServletResponse response = result.getResponse();
174
175     assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
176     assertEquals("Request with ID 0 was not found", response.getContentAsString());
177 }
```

```

179
180     @Test
181     @DisplayName("Test addNewRequest success") // test for adding a new request successfully
182     void testAddNewRequestSuccess() throws Exception {
183         // Mocking service
184         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
185             .thenReturn(null);
186         when(requestService.saveRequest(ArgumentMatchers.any(Request.class))).thenReturn(requests.get(0));
187         String inputJson = ("{" + " \"userId\":\"1\"", "\n" + " \"objectId\":\"1\"", "\n"
188             + " \"requestType\":\"1\""\n" + "}");
189         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/requests/new").accept(MediaType.APPLICATION_JSON)
190             .content(inputJson).contentType(MediaType.APPLICATION_JSON);
191
192         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
193         MockHttpServletResponse response = result.getResponse();
194
195         assertEquals(HttpStatus.CREATED.value(), response.getStatus());
196     }
197
198     @Test
199     @DisplayName("Test addNewRequest null user error") // test for adding a new request without a user
200     void testAddNewRequestNullUserError() throws Exception {
201         // Mocking service
202         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
203             .thenReturn(null);
204         when(requestService.saveRequest(ArgumentMatchers.any(Request.class))).thenReturn(requests.get(0));
205         String inputJson = ("{" + " \"objectId\":\"1\"", "\n" + " \"requestType\":\"1\""\n" + "}");
206         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/requests/new").accept(MediaType.APPLICATION_JSON)
207             .content(inputJson).contentType(MediaType.APPLICATION_JSON);
208
209         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
210         MockHttpServletResponse response = result.getResponse();
211
212         assertEquals(HttpStatus.NOT_ACCEPTABLE.value(), response.getStatus());
213         assertEquals("Unable to add the new request, User id not given!.", response.getContentAsString());
214     }

```

```

216
217     @Test
218     @DisplayName("Test addNewRequest duplicate error") // test for adding a duplicate new request
219     void testAddNewRequestDuplicateError() throws Exception {
220         // Mocking service
221         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
222             .thenReturn(null);
223         when(requestService.saveRequest(ArgumentMatchers.any(Request.class))).thenReturn(null);
224         String inputJson = ("{" + " \"userId\":\"1\"", "\n" + " \"objectId\":\"1\"", "\n"
225             + " \"requestType\":\"1\""\n" + "}");
226         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/requests/new").accept(MediaType.APPLICATION_JSON)
227             .content(inputJson).contentType(MediaType.APPLICATION_JSON);
228
229         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
230         MockHttpServletResponse response = result.getResponse();
231
232         assertEquals(HttpStatus.CONFLICT.value(), response.getStatus());
233         assertEquals("Unable to add the new request, a copy of the request already exists.", response.getContentAsString());
234     }
235
236     @Test
237     @DisplayName("Test addNewRequest badRequest") // test for adding a new request with invalid information
238     void testAddNewRequestBadRequest() throws Exception {
239         // Mocking service
240         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
241             .thenReturn(null);
242
243         String inputJson = "null";
244         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/requests/new").accept(MediaType.APPLICATION_JSON)
245             .content(inputJson).contentType(MediaType.APPLICATION_JSON);
246
247         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
248         MockHttpServletResponse response = result.getResponse();
249
250         assertEquals(HttpStatus.BAD_REQUEST.value(), response.getStatus());
251     }

```

```

252
253     @Test
254     @DisplayName("Test deleteRequest success") // test for successfully deleting a request
255     void testDeleteRequestSuccess() throws Exception {
256         // Mocking service
257         when(requestService.findById(1L)).thenReturn(requests.get(0));
258
259         RequestBuilder requestBuilder = MockMvcRequestBuilders.delete("/api/requests/1");
260
261         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
262         MockHttpServletResponse response = result.getResponse();
263
264         assertEquals(HttpStatus.OK.value(), response.getStatus());
265         assertEquals("Request with ID 1 was deleted", response.getContentAsString());
266     }
267
268     @Test
269     @DisplayName("Test deleteRequest not found") // test for deleting a request that doesn't exist
270     void testDeleteRequestNotFound() throws Exception {
271         // Mocking service
272         when(requestService.findById(0L)).thenReturn(null);
273
274         RequestBuilder requestBuilder = MockMvcRequestBuilders.delete("/api/requests/0");
275
276         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
277         MockHttpServletResponse response = result.getResponse();
278
279         assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
280         assertEquals("Request with ID 0 was not found", response.getContentAsString());
281     }
282 }

```

RequestsRepositoryTest.java

```
27
28     // testing for empty request repository
29     @Test
30     @Rollback(true)
31     public void should_find_no_requests_if_requestRepository_is_empty() {
32         Iterable<Request> requests = requestRepository.findAll();
33         if (requests.iterator().hasNext()) {
34             assertThat(requests).isNotEmpty();
35         } else {
36             assertThat(requests).isEmpty();
37         }
38     }
39
40     // test to find a request by id
41     @Test
42     @Rollback(true)
43     public void find_request_by_id() {
44         Request request1 = new Request();
45         request1.setId(1L);
46         requestRepository.save(request1);
47
48         Request findRequest = requestRepository.findById(1L).orElse(null);
49         assertNotNull(findRequest);
50     }
51
52     // test to find a request by non-existing id
53     @Test
54     @Rollback(true)
55     public void find_request_by_nonexisting_id() {
56         Request findRequest = requestRepository.findById(0L).orElse(null);
57         assertNull(findRequest);
58     }
59
60
61     // test to delete a request
62     @Test
63     @Rollback(true)
64     public void delete_request() {
65         Request request1 = new Request();
66         request1.setId(1L);
67         requestRepository.save(request1);
68         requestRepository.delete(request1);
69         Request findRequest = requestRepository.findById(1L).orElse(null);
70         assertNull(findRequest);
71     }
72
73     // test to save a new request
74     @Test
75     @Rollback(true)
76     public void save_new_request() {
77         Request request1 = new Request();
78         request1.setId(1L);
79         requestRepository.save(request1);
80
81         Request findRequest = requestRepository.findById(1L).orElse(null);
82         assertNotNull(findRequest);
83
84         Request request2 = new Request();
85         request2.setId(2L);
86         requestRepository.save(request2);
87
88         Request findRequest2 = requestRepository.findById(2L).orElse(null);
89         assertNotNull(findRequest2);
90     }
91 }
```

RequestsServiceTest.java

```
52     @Test
53     @DisplayName("Test findAllRequests success") // test for finding all requests
54     public void testfindAllRequestsSuccess() throws Exception {
55         given(requestRepository.findAll()).willReturn(requests);
56         Iterable<Request> allRequests = requestService.findAllRequests();
57         Assert.assertNotNull(allRequests);
58     }
59
60     @Test
61     @DisplayName("Test saveRequest success") // test for saving a request
62     public void testSaveRequestSuccess() throws Exception {
63         given(requestRepository.save(request1)).willReturn(request1);
64         Request approvedRequest = requestService.saveRequest(request1);
65         Assert.assertNotNull(approvedRequest);
66     }
67
68     @Test
69     @DisplayName("Test saveRequest failure") // test for failing to save a request
70     public void testSaveRequestFailure() throws Exception {
71         given(requestRepository.save(request2)).willReturn(null);
72         Request approvedRequest = requestService.saveRequest(request1);
73         Assert.assertNull(approvedRequest);
74     }
75 }
```

Users microservice Tests:

Tester: Jared

EditUserControllerTest.java

```
72     @Test
73     @DisplayName("Test editUser success") // test for editing a user successfully
74     void testEditUserSuccess() throws Exception {
75         // Mocking service
76         when(userService.findById(1L)).thenReturn(users.get(0));
77         when(userService.updateUser(ArgumentMatchers.any(UserForm.class), ArgumentMatchers.any(User.class)))
78             .thenReturn(users.get(0));
79
80         String inputJson = "{\n" +
81             "    \"username\":\"newUsername\",\\n" +
82             "    \"password\":\"newPassword\"\\n" +
83             "}";
84         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editUser/1")
85             .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
86
87         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
88         MockHttpServletResponse response = result.getResponse();
89
90         assertEquals(HttpStatus.OK.value(), response.getStatus());
91         assertEquals("Successfully updated user details", response.getContentAsString());
92     }
93
94     @Test
95     @DisplayName("Test editUser username taken") // test for trying to change a username to one that is already taken
96     void testEditUserUsernameTaken() throws Exception {
97         // Mocking service
98         when(userService.findById(1L)).thenReturn(users.get(0));
99         when(userService.updateUser(ArgumentMatchers.any(UserForm.class), ArgumentMatchers.any(User.class)))
100             .thenReturn(null);
101
102         String inputJson = "{\n" +
103             "    \"username\":\"JohnDoe2\\n" +
104             "}";
105         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editUser/1")
106             .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
107
108         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
109         MockHttpServletResponse response = result.getResponse();
110
111         assertEquals(HttpStatus.CONFLICT.value(), response.getStatus());
112         assertEquals("Unable to save details, Username 'JohnDoe2' already taken", response.getContentAsString());
113     }
114 }
```

```

115     @Test
116     @DisplayName("Test editUser not found") // test for editing a user that doesn't exist
117     void testEditUserNotFound() throws Exception {
118         // Mocking service
119         when(userService.findById(3L)).thenReturn(null);
120
121         String inputJson = "{\n" +
122             "    \"username\":\"newUsername\",\n" +
123             "    \"password\":\"newPassword\"\n" +
124             "}";
125         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editUser/3")
126             .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
127
128         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
129         MockHttpServletResponse response = result.getResponse();
130
131         assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
132         assertEquals("User with ID 3 was not found", response.getContentAsString());
133     }
134
135     @Test
136     @DisplayName("Test editPassword success") // test for changing a user's password successfully
137     void testEditPasswordSuccess() throws Exception {
138         // Mocking service
139         when(userService.findById(1L)).thenReturn(users.get(0));
140         when(userService.updateUserPassword(ArgumentMatchers.any(UserForm.class), ArgumentMatchers.any(User.class)))
141             .thenReturn(users.get(0));
142
143         String inputJson = "{\n" +
144             "    \"password\":\"newPassword\"\n" +
145             "}";
146         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editUser/password/1")
147             .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
148
149         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
150         MockHttpServletResponse response = result.getResponse();
151
152         assertEquals(HttpStatus.OK.value(), response.getStatus());
153         assertEquals("Successfully updated password", response.getContentAsString());
154     }
155
156
157     @Test
158     @DisplayName("Test editPassword invalid password") // test for changing a user's password to an invalid password
159     void testEditPasswordInvalid() throws Exception {
160         // Mocking service
161         when(userService.findById(1L)).thenReturn(users.get(0));
162         when(userService.updateUserPassword(ArgumentMatchers.any(UserForm.class), ArgumentMatchers.any(User.class)))
163             .thenReturn(null);
164
165         String inputJson = "{\n" +
166             "    \"password\":\"\""
167             "}";
168         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editUser/password/1")
169             .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
170
171         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
172         MockHttpServletResponse response = result.getResponse();
173
174         assertEquals(HttpStatus.NOT_ACCEPTABLE.value(), response.getStatus());
175         assertEquals("Unable to update invalid password", response.getContentAsString());
176     }
177
178     @Test
179     @DisplayName("Test editPassword user not found") // test for changing a user's password when the user doesn't exist
180     void testEditPasswordUserNotFound() throws Exception {
181         // Mocking service
182         when(userService.findById(3L)).thenReturn(null);
183
184         String inputJson = "{\n" +
185             "    \"password\":\"newPassword\"\n" +
186             "}";
187         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/editUser/3")
188             .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
189
190         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
191         MockHttpServletResponse response = result.getResponse();
192
193         assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
194         assertEquals("User with ID 3 was not found", response.getContentAsString());
195     }

```

UserControllerTest.java

```
87
88     @Test
89     @DisplayName("Test findAllUsers") // test for getting all users
90     void testfindAllUsers() throws Exception {
91         // Mocking service
92         when(userService.findAllUsers()).thenReturn(users);
93         mockMvc.perform(get("/api/users/all").contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk())
94             .andExpect(jsonPath("$.id", is(1))).andExpect(jsonPath("$.email", is("johndoe@gmail.com")))
95             .andExpect(jsonPath("$.username", is("JohnDoe")))
96             .andExpect(jsonPath("$.fullName", is("John Doe")))
97             .andExpect(jsonPath("$.password", is("password")))
98             .andExpect(jsonPath("$.address", is("1 John Street, Doeland")))
99
100            .andExpect(jsonPath("$[1].id", is(2))).andExpect(jsonPath("$.email", is("johndoe2@gmail.com")))
101            .andExpect(jsonPath("$.username", is("JohnDoe2")))
102            .andExpect(jsonPath("$.fullName", is("John Doe")))
103            .andExpect(jsonPath("$.password", is("password")))
104            .andExpect(jsonPath("$.address", is("2 John Street, Doeland")));
105
106    }
107
108    @Test
109    @DisplayName("Test getUser success") // test for getting a user successfully
110    void testGetUserSuccess() throws Exception {
111        // Mocking service
112        when(userService.findById(1L)).thenReturn(users.get(0));
113
114        mockMvc.perform(get("/api/users/1").contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk())
115            .andExpect(jsonPath("id", is(1))).andExpect(jsonPath("email", is("johndoe@gmail.com")))
116            .andExpect(jsonPath("username", is("JohnDoe"))).andExpect(jsonPath("fullName", is("John Doe")))
117            .andExpect(jsonPath("password", is("password")))
118            .andExpect(jsonPath("address", is("1 John Street, Doeland")));
119    }
120
```

```
121
122     @Test
123     @DisplayName("Test getUser not found") // test for getting a user that doesn't exist
124     void testGetUserNotFound() throws Exception {
125         // Mocking service
126         when(userService.findById(3L)).thenReturn(null);
127
128         RequestBuilder requestBuilder = MockMvcRequestBuilders.get("/api/users/3")
129             .contentType(MediaType.APPLICATION_JSON);
130         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
131
132         MockHttpServletResponse response = result.getResponse();
133
134         assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
135         assertEquals("User with ID 3 was not found", response.getContentAsString());
136     }
137
138     @Test
139     @DisplayName("Test registerUser success") // test for registering a user successfully
140     void testRegisterUserSuccess() throws Exception {
141         // Mocking service
142         when(mapValidationErrorMessageService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
143             .thenReturn(null);
144         when(userService.saveUser(ArgumentMatchers.any(User.class))).thenReturn(users.get(0));
145
146         String inputJson = "\n" +
147             "    \"username\":\"TestUsername\", \n" +
148             "    \"password\":\"password\"\n" + "}";
149         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/users/register")
150             .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
151
152         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
153         MockHttpServletResponse response = result.getResponse();
154
155         assertEquals(HttpStatus.CREATED.value(), response.getStatus());
156     }

```

```

156     @Test
157     @DisplayName("Test registerUser badRequest") // test for registering a user with invalid credentials
158     void testRegisterUserBadRequest() throws Exception {
159         // Mocking service
160         when(mapValidationService.MapValidationService(ArgumentMatchers.any(BindingResult.class)))
161             .thenReturn(null);
162         when(userService.saveUser(ArgumentMatchers.any(User.class))).thenReturn(users.get(0));
163
164         String inputJson = "null";
165         RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/api/users/register")
166             .accept(MediaType.APPLICATION_JSON).content(inputJson).contentType(MediaType.APPLICATION_JSON);
167
168         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
169         MockHttpServletResponse response = result.getResponse();
170
171         assertEquals(HttpStatus.BAD_REQUEST.value(), response.getStatus());
172     }
173
174     @Test
175     @DisplayName("Test deleteUser success") // test for successfully deleting a user
176     void testDeleteUserSuccess() throws Exception {
177         // Mocking service
178         when(userService.findById(1L)).thenReturn(users.get(0));
179
180         RequestBuilder requestBuilder = MockMvcRequestBuilders.delete("/api/users/1");
181
182         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
183         MockHttpServletResponse response = result.getResponse();
184
185         assertEquals(HttpStatus.OK.value(), response.getStatus());
186         assertEquals("User with ID 1 was deleted", response.getContentAsString());
187     }
188
189     @Test
190     @DisplayName("Test deleteUser not found") // test for deleting a user that doesn't exist
191     void testDeleteUserNotFound() throws Exception {
192         // Mocking service
193         when(userService.findById(3L)).thenReturn(null);
194
195         RequestBuilder requestBuilder = MockMvcRequestBuilders.delete("/api/users/3");
196
197         MvcResult result = mockMvc.perform(requestBuilder).andReturn();
198         MockHttpServletResponse response = result.getResponse();
199
200         assertEquals(HttpStatus.NOT_FOUND.value(), response.getStatus());
201         assertEquals("User with ID 3 was not found", response.getContentAsString());
202     }
203 }
204

```

UserRepositoryTest.java

```

31 // testing for empty user repository
32 @Test
33 @Rollback(true)
34 public void should_find_no_users_if_userRepository_is_empty() {
35     Iterable<User> users = userRepository.findAll();
36     if (users.iterator().hasNext()) {
37         assertThat(users).isNotEmpty();
38     } else {
39         assertThat(users).isEmpty();
40     }
41 }
42
43 // test to find a user by username
44 @Test
45 @Rollback(true)
46 public void find_user_by_username() {
47     User user = new User();
48     user.setId(1L);
49     user.setEmail("johndoe@gmail.com");
50     user.setUsername("JohnDoe");
51     user.setFullName("John Doe");
52     user.setPassword("password");
53     user.setAddress("1 John Street, DoeLand");
54     user.setRole(Role.USER_NORMAL);
55     user.setUserStatus(UserStatus.ENABLED);
56     userRepository.save(user);
57
58     User findUser = userRepository.findByUsername("JohnDoe");
59     assertNotNull(findUser);
60     assertThat(findUser).hasFieldOrPropertyWithValue("email", "johndoe@gmail.com");
61     assertThat(findUser).hasFieldOrPropertyWithValue("username", "JohnDoe");
62     assertThat(findUser).hasFieldOrPropertyWithValue("fullName", "John Doe");
63     assertThat(findUser).hasFieldOrPropertyWithValue("password", "password");
64     assertThat(findUser).hasFieldOrPropertyWithValue("address", "1 John Street, DoeLand");
65     assertThat(findUser).hasFieldOrPropertyWithValue("role", Role.USER_NORMAL);
66     assertThat(findUser).hasFieldOrPropertyWithValue("userStatus", UserStatus.ENABLED);
67 }
68
69 // test to delete a user
70 @Test
71 @Rollback(true)
72 public void delete_user() {
73     User user = new User();
74     user.setId(0L);
75     user.setEmail("johndoe@gmail.com");
76     user.setUsername("JohnDoe");
77     user.setFullName("John Doe");
78     user.setPassword("password");
79     user.setAddress("1 John Street, DoeLand");
80     user.setRole(Role.USER_NORMAL);
81     user.setUserStatus(UserStatus.ENABLED);
82     User savedUser = userRepository.save(user);
83     userRepository.delete(savedUser);
84
85     Iterable<User> users = userRepository.findAll();
86     Iterator<User> iter = users.iterator();
87     while(iter.hasNext()) {
88         User iterUser = iter.next();
89         assertThat(iterUser.getUsername()).isNotEqualTo(savedUser.getUsername());
90     }
91 }
92

```

UserServiceTest.java

```
59     @Test
60     @DisplayName("Test findByUsername success") // test for finding a user by username successfully
61     public void testFindByUsernameSuccess() throws Exception {
62         String username = user1.getUsername();
63         given(userRepository.findByUsername(username)).willReturn(user1);
64         User user = userService.findByUsername(username);
65         Assert.assertEquals(user.getEmail(), "johndoe@gmail.com");
66         Assert.assertEquals(user.getUsername(), "JohnDoe");
67         Assert.assertEquals(user.getFullName(), "John Doe");
68         Assert.assertEquals(user.getPassword(), "password");
69         Assert.assertEquals(user.getAddress(), "1 John Street, Doeland");
70     }
71
72     @Test
73     @DisplayName("Test findByUsername fail") // test for finding a user by username which doesn't exist
74     public void testFindByUsernameFail() throws Exception {
75         String username = "null";
76         given(userRepository.findByUsername(username)).willReturn(null);
77         User user = userService.findByUsername(username);
78         Assert.assertNull(user);
79     }
80 }
```

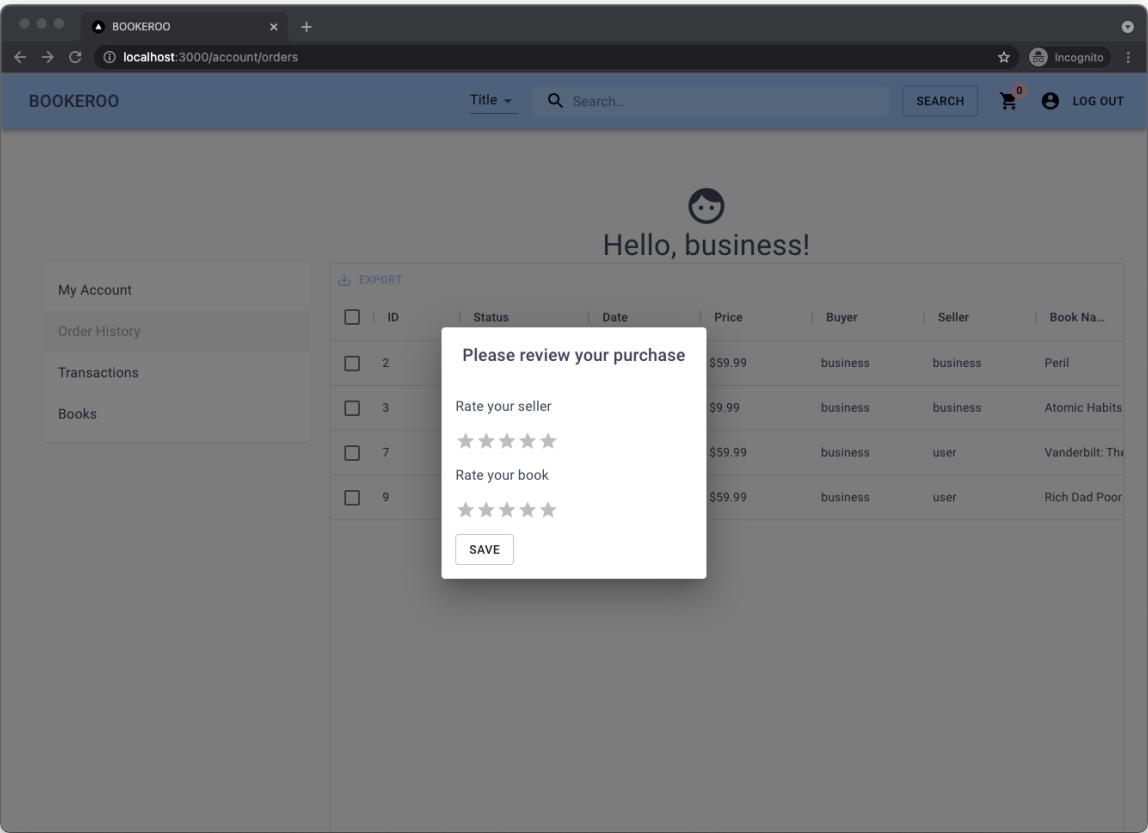
```
93     // test to save a new user
94     @Test
95     @Rollback(true)
96     public void save_new_user() {
97         User user = new User();
98         user.setId(1L);
99         user.setEmail("johndoe@gmail.com");
100        user.setUsername("JohnDoe");
101        user.setFullName("John Doe");
102        user.setPassword("password");
103        user.setAddress("1 John Street, Doeland");
104        user.setRole(Role.USER_NORMAL);
105        user.setUserStatus(UserStatus.ENABLED);
106        userRepository.save(user);
107
108        User savedUser = userRepository.save(user);
109
110        assertThat(savedUser).hasFieldOrPropertyWithValue("email", "johndoe@gmail.com");
111        assertThat(savedUser).hasFieldOrPropertyWithValue("username", "JohnDoe");
112        assertThat(savedUser).hasFieldOrPropertyWithValue("fullName", "John Doe");
113        assertThat(savedUser).hasFieldOrPropertyWithValue("password", "password");
114        assertThat(savedUser).hasFieldOrPropertyWithValue("address", "1 John Street, Doeland");
115        assertThat(savedUser).hasFieldOrPropertyWithValue("role", Role.USER_NORMAL);
116        assertThat(savedUser).hasFieldOrPropertyWithValue("userStatus", UserStatus.ENABLED);
117    }
118 }
```

```
119     // test to find all users after saving a new user
120     @Test
121     @Rollback(true)
122     public void find_all_returns_saved_user() {
123         User user = new User();
124         user.setId(1L);
125         user.setEmail("johndoe@gmail.com");
126         user.setUsername("JohnDoe");
127         user.setFullName("John Doe");
128         user.setPassword("password");
129         user.setAddress("1 John Street, Doeland");
130         user.setRole(Role.USER_NORMAL);
131         user.setUserStatus(UserStatus.ENABLED);
132         userRepository.save(user);
133
134         Iterable<User> users = userRepository.findAll();
135
136         User savedUser = users.iterator().next();
137         assertThat(savedUser.getEmail().equals("johndoe@gmail.com"));
138         assertThat(savedUser.getUsername().equals("JohnDoe"));
139         assertThat(savedUser.getFullName().equals("John Doe"));
140         assertThat(savedUser.getAddress().equals("1 John Street, Doeland"));
141         assertThat(savedUser.getRole().equals(Role.USER_NORMAL));
142         assertThat(savedUser.getUserStatus().equals(UserStatus.ENABLED));
143     }
144 }
```

Acceptance Testing

*For new user stories in sprint 4

As a customer, I want to add a review to a book I purchased, so I can share my opinion.

Given: A user is login, and there is a transaction is ready to be review
When: The user clicks the review button
Expect: A review dialog will pop up
Screenshot:  A screenshot of a web browser window titled 'BOOKEROO' at 'localhost:3000/account/orders'. The page displays a table of transactions with columns for ID, Status, Date, Price, Buyer, Seller, and Book Name. Overlaid on the table is a white rectangular dialog box with a dark gray border. At the top of the dialog is a small circular icon with a face and the text 'Hello, business!'. Inside the dialog, there is a message 'Please review your purchase' followed by two rating scales. The first scale is labeled 'Rate your seller' and has five stars filled. The second scale is labeled 'Rate your book' and also has five stars filled. At the bottom of the dialog is a single button labeled 'SAVE'.

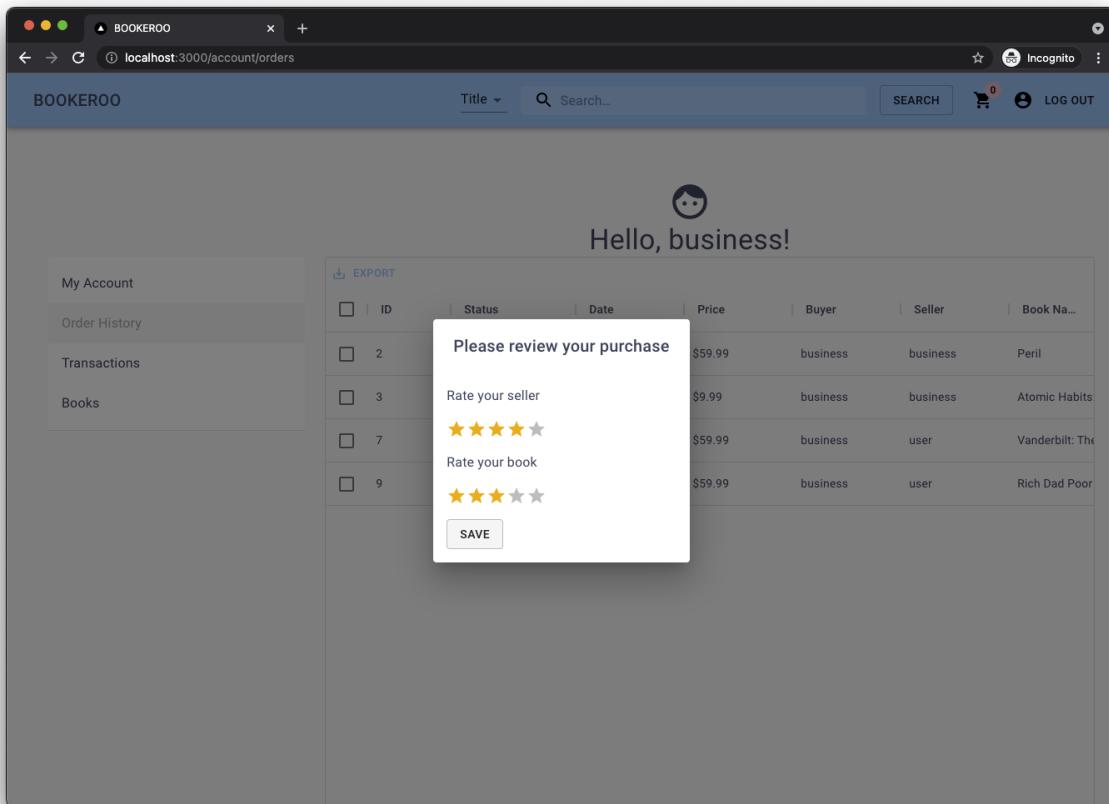
Given: A user is login, and the review dialog is open

When:

The user adds the rating to the book and click save

Expect:

The review will be updated, and back end will be updated

Screenshot:

As a customer, I want to add a review to a user who sold an item to me, so I can rate the seller.

Given:

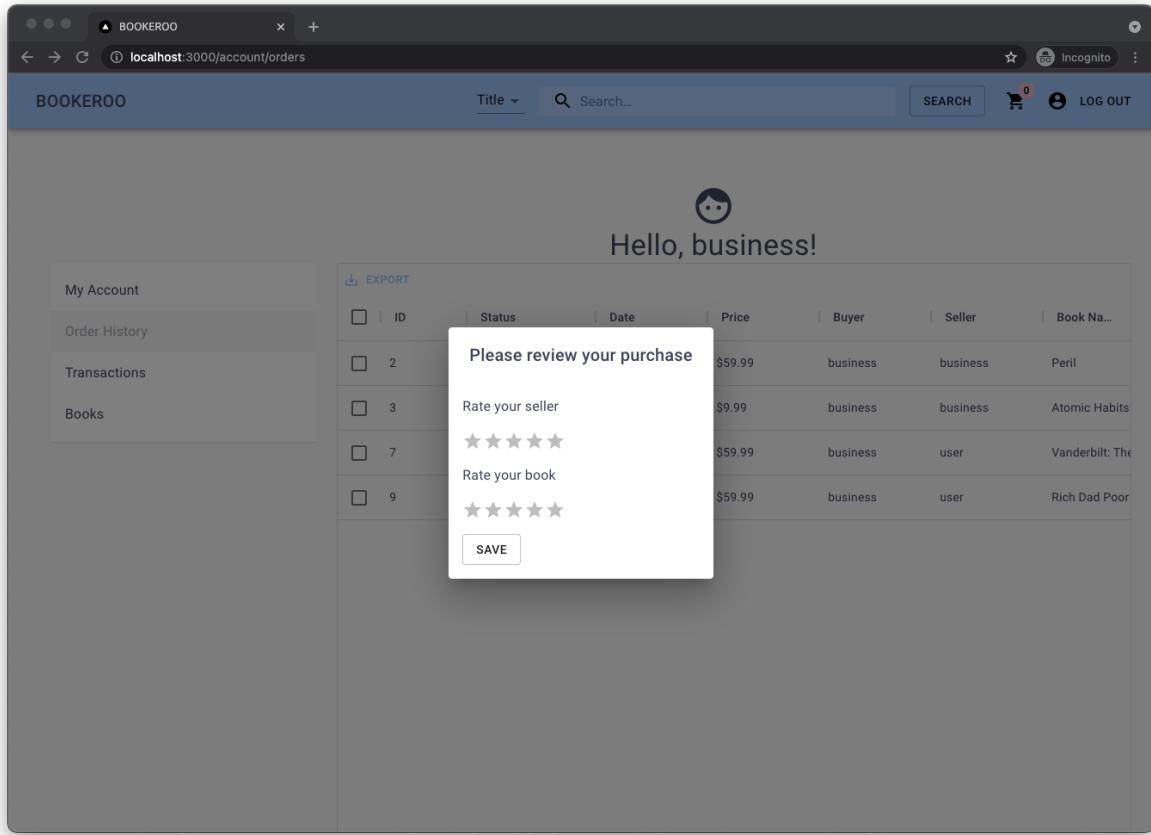
A user is login, and there is s transaction is ready to be review

When:

The user clicks the review button

Expect:

A review dialog will pop up

Screenshot:**Given:**

A user is login, and the review dialog is open

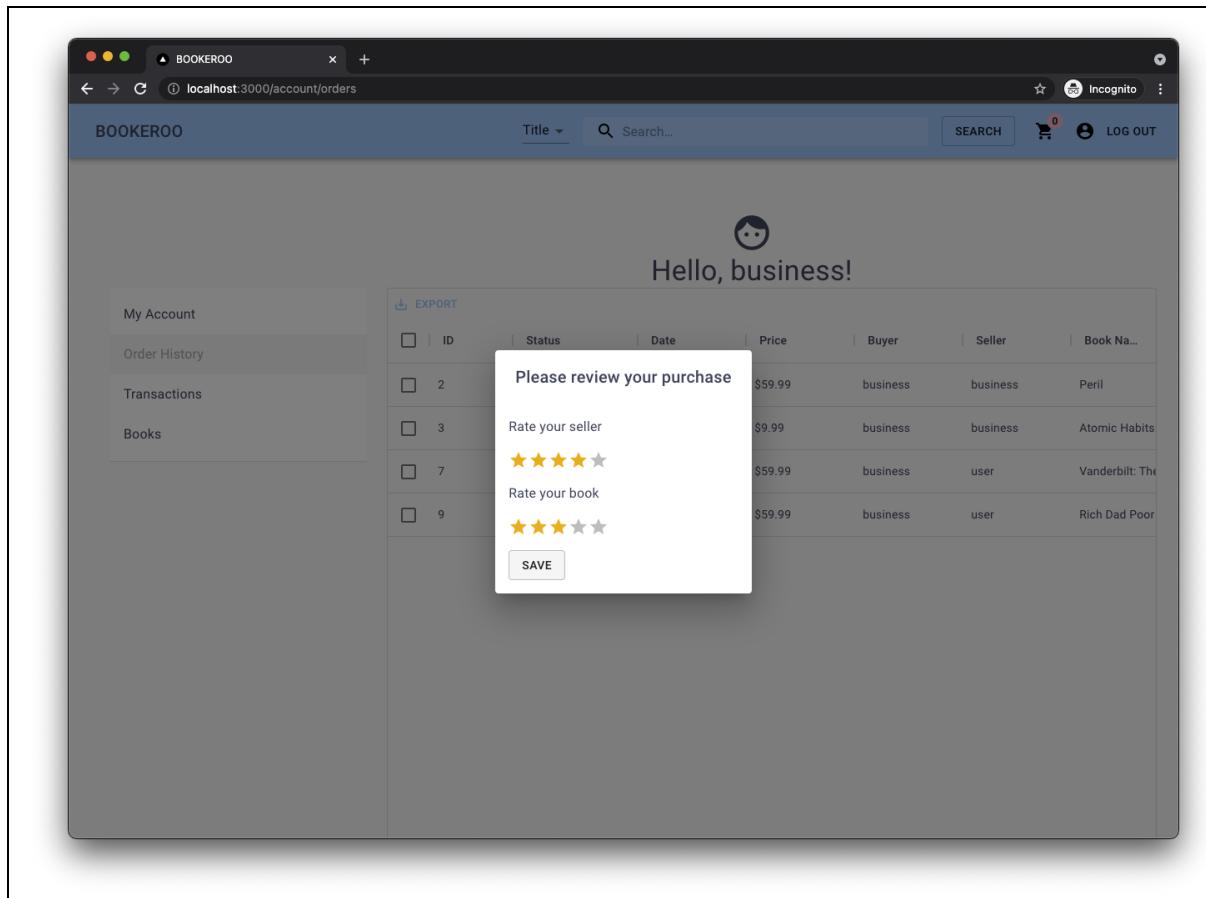
When:

The user adds the rating to the seller and click save

Expect:

The review will be updated, and back end will be updated

Screenshot:



As a customer, I want to be able to filter my search, so I can find the items I am interested in more easily.

Given:

A user wants to use the search function

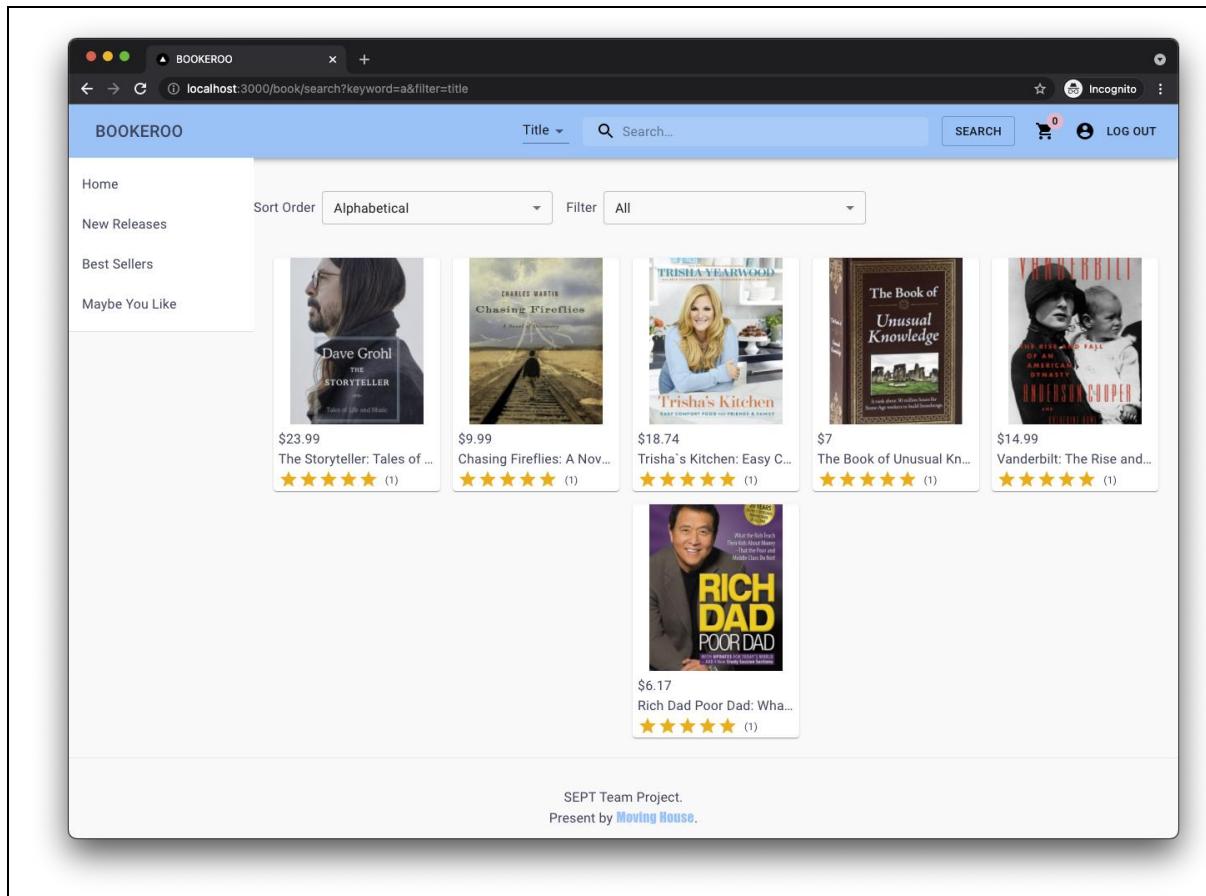
When:

The user is on the search result page

Expect:

There is a small search bar on the top

Screenshot:



As a customer, I want to be able to filter my search results so that I can view only used books for sale.

Given:

The user is on the search result page

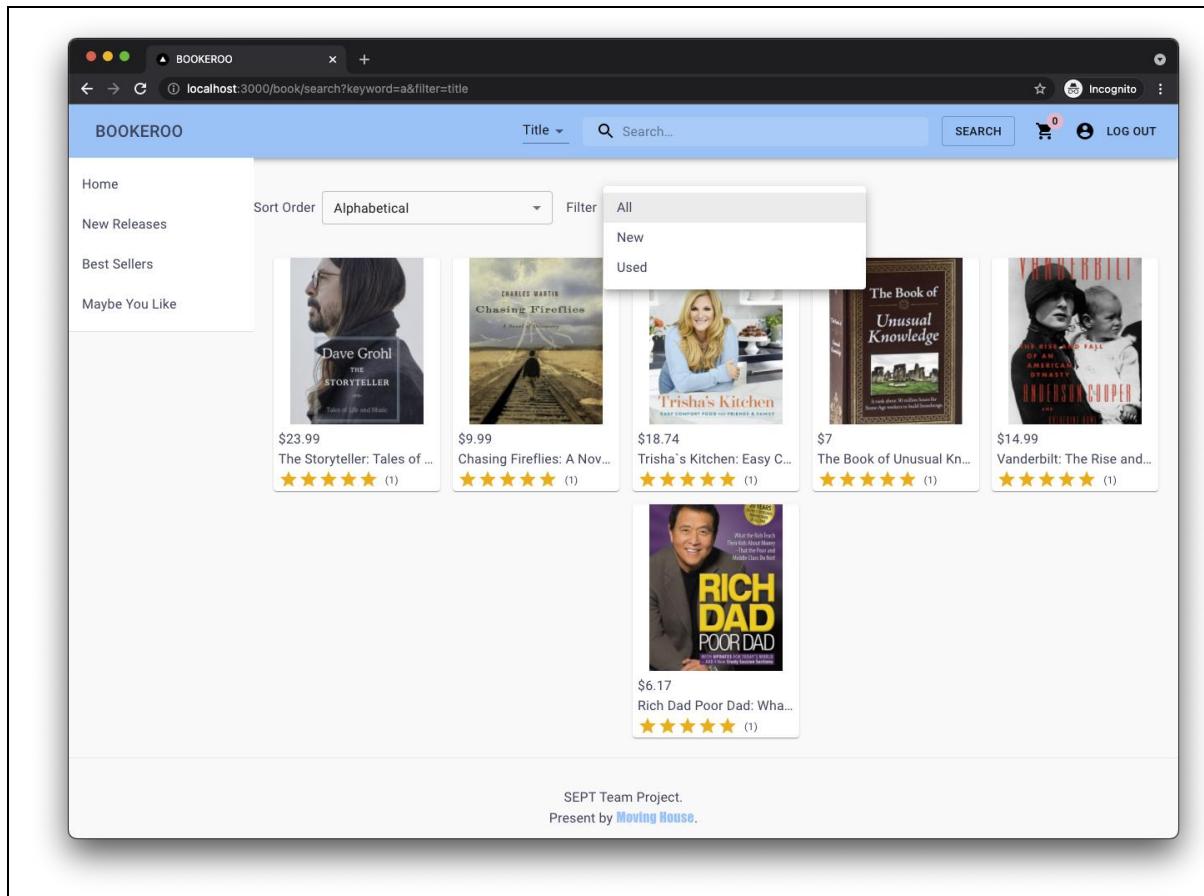
When:

The user clicks the filter button

Expect:

The result could be flited by USED

Screenshot:



As a customer, I want to be able to filter my search results so that I can view only new books for sale.

Given:

The user is on the search result page

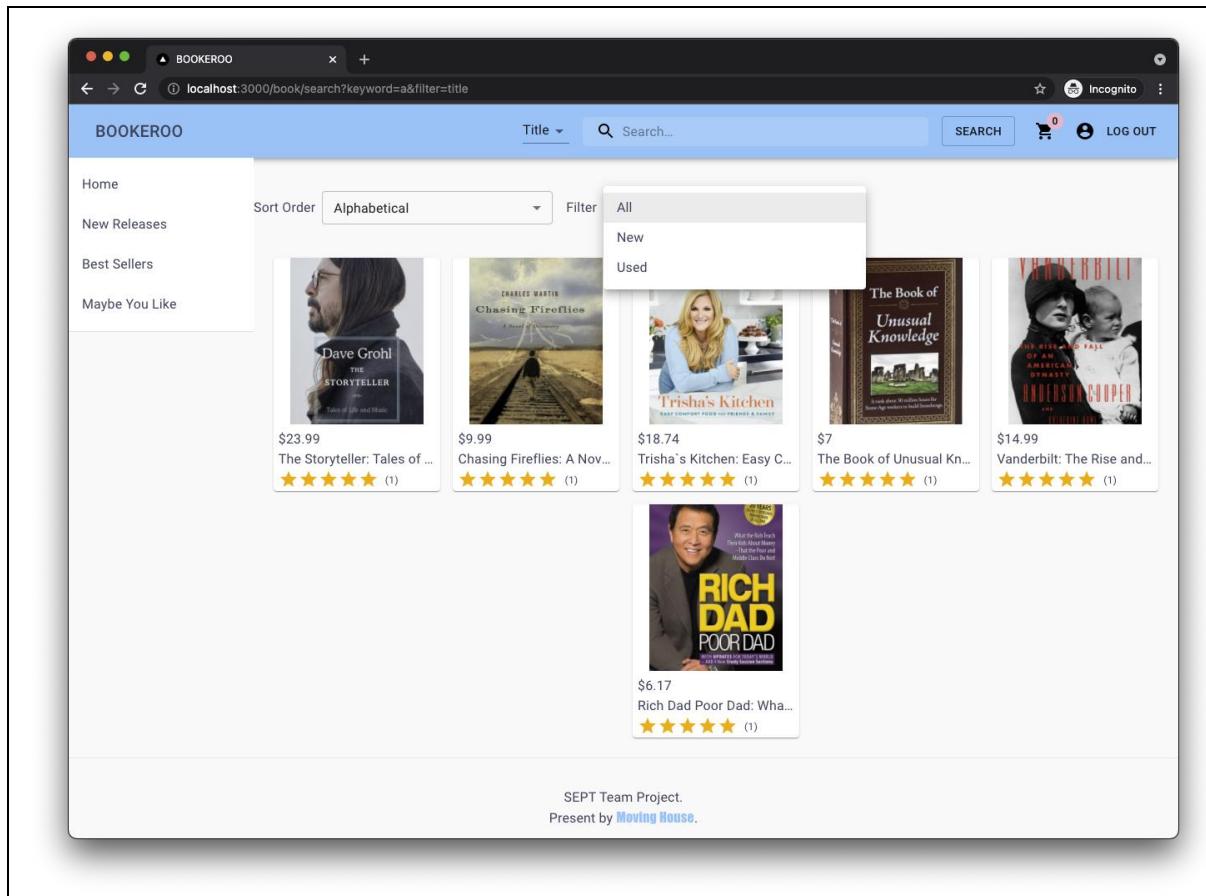
When:

The user clicks the filter button

Expect:

The result could be flited by NEW

Screenshot:



As a business user, I want to change my account type to a regular user account, so that I can remove my business' details from my account.

Given:

A business user is logged in, and on the account page

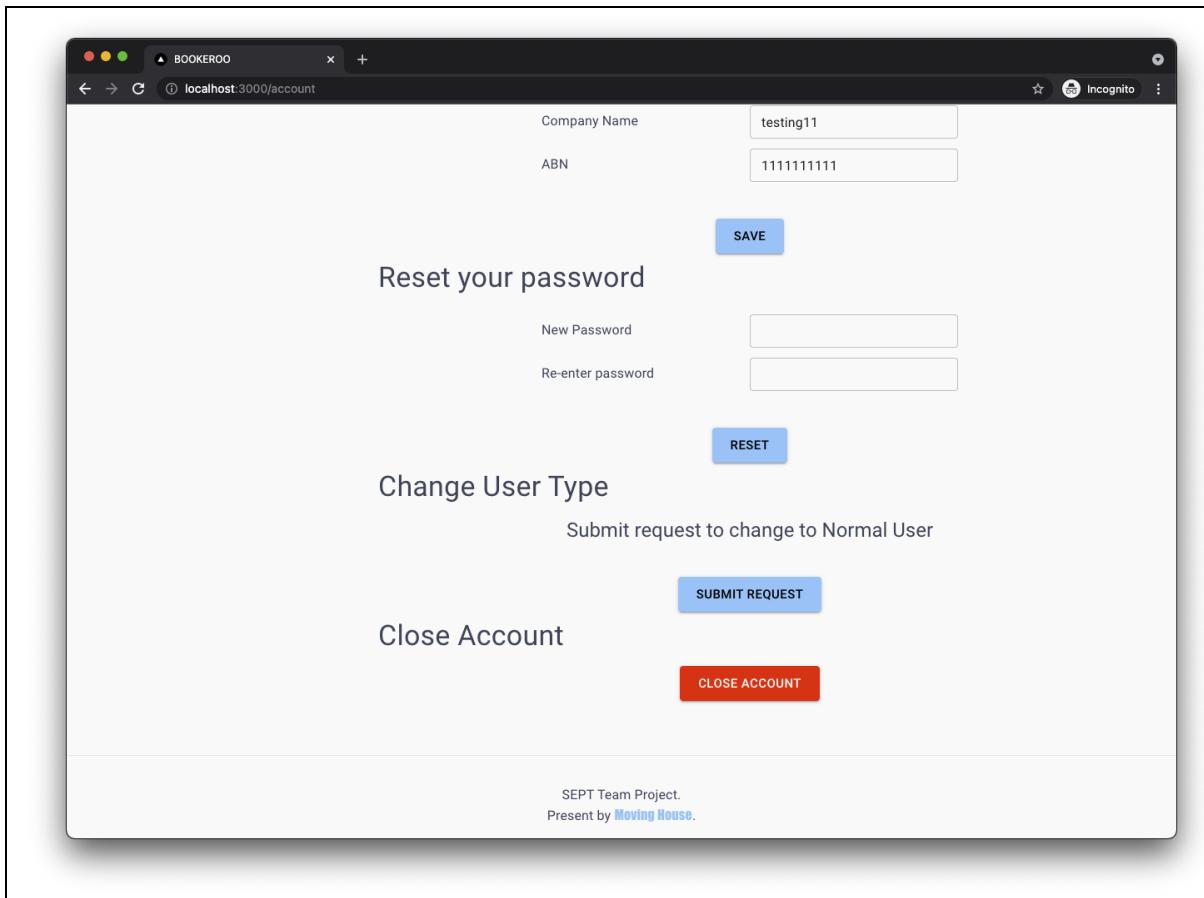
When:

The user clicks the button for change account type

Expect:

A new request will be created for admin' approval

Screenshot:



As a customer, I want to change my account type to a business account, so that I can add my business' name and ABN to my account details.

Given:

A normal user is logged in, and on the account page

When:

The user adds the new company name and abn, and clicks the button for change account type

Expect:

A new request will be created for admin' approval

Screenshot:

The screenshot shows a web application interface with three main sections:

- Reset your password:** Contains fields for "New Password" and "Re-enter password" with a "SAVE" button.
- Change User Type:** Contains fields for "Company Name" (with value "new business") and "ABN" (with value "12345678") with a "RESET" button.
- Close Account:** Contains a "SUBMIT REQUEST" button and a "CLOSE ACCOUNT" button.

At the bottom of the page, there is a footer note: "SEPT Team Project. Present by [Moving House](#)".

As an admin, I want to sort all pending registrations from business users so I can view the oldest one first.

Given:

The admin is logged in, and on the request page

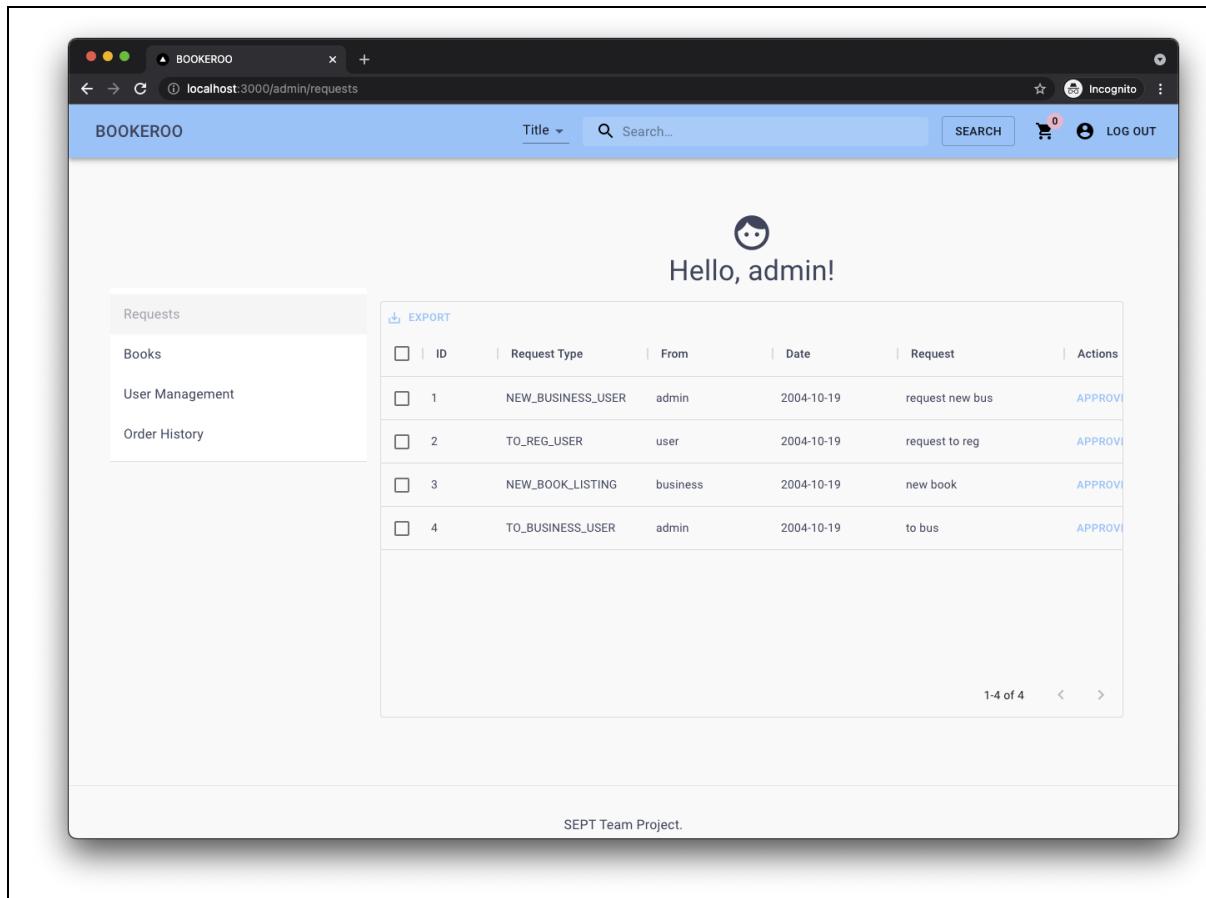
When:

Admin clicks the arrow near the date column

Expect:

The table of request will be sorted by date

Screenshot:



As a business user, I want to adjust the amount of stock for an item, so that item stock can be increased or decreased.

Given:

A business user is logged in and on the book management page

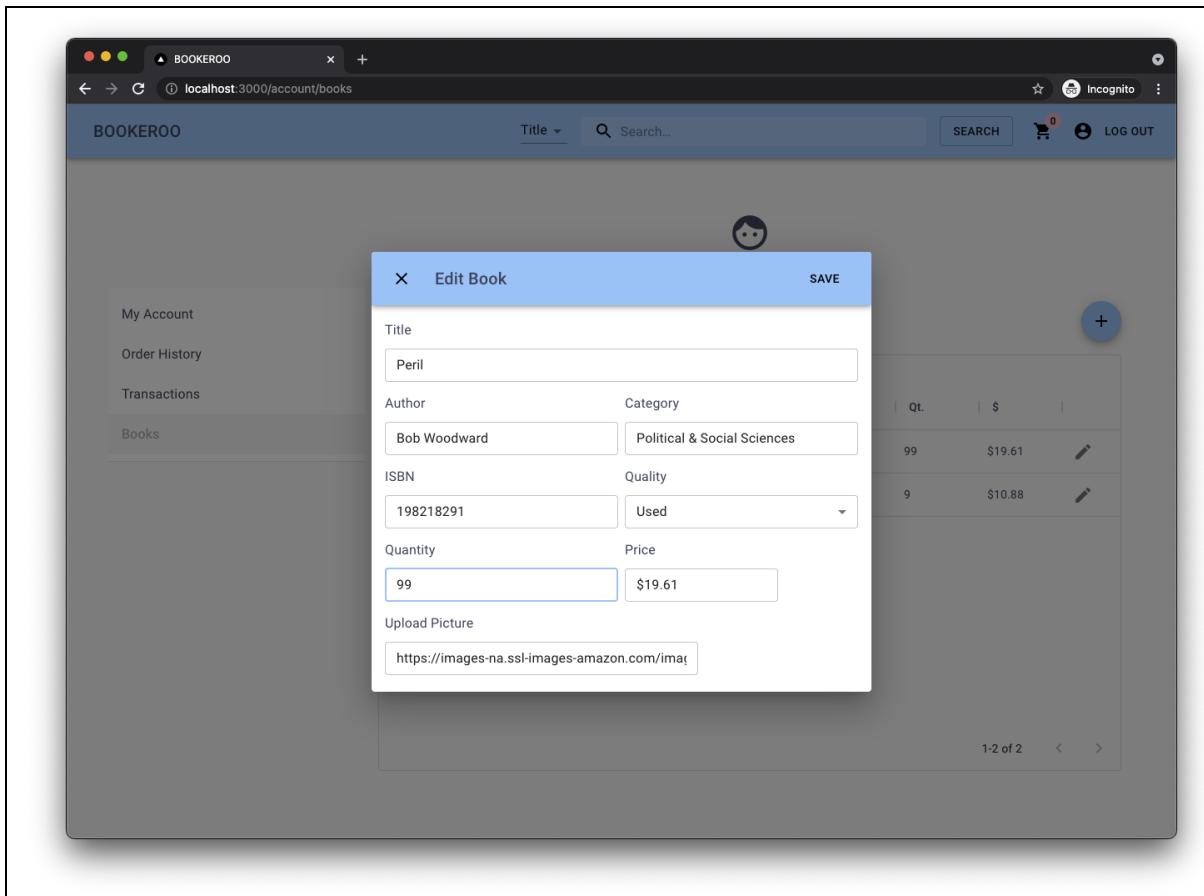
When:

The user clicks the edit button for a book

Expect:

A editing dialog with a changeable quantity field will be shown

Screenshot:



As a business user, I want to know how much inventory I have left on an item, so I can decide if I need to print more books.

Given:

A business user is logged in and on the book management page

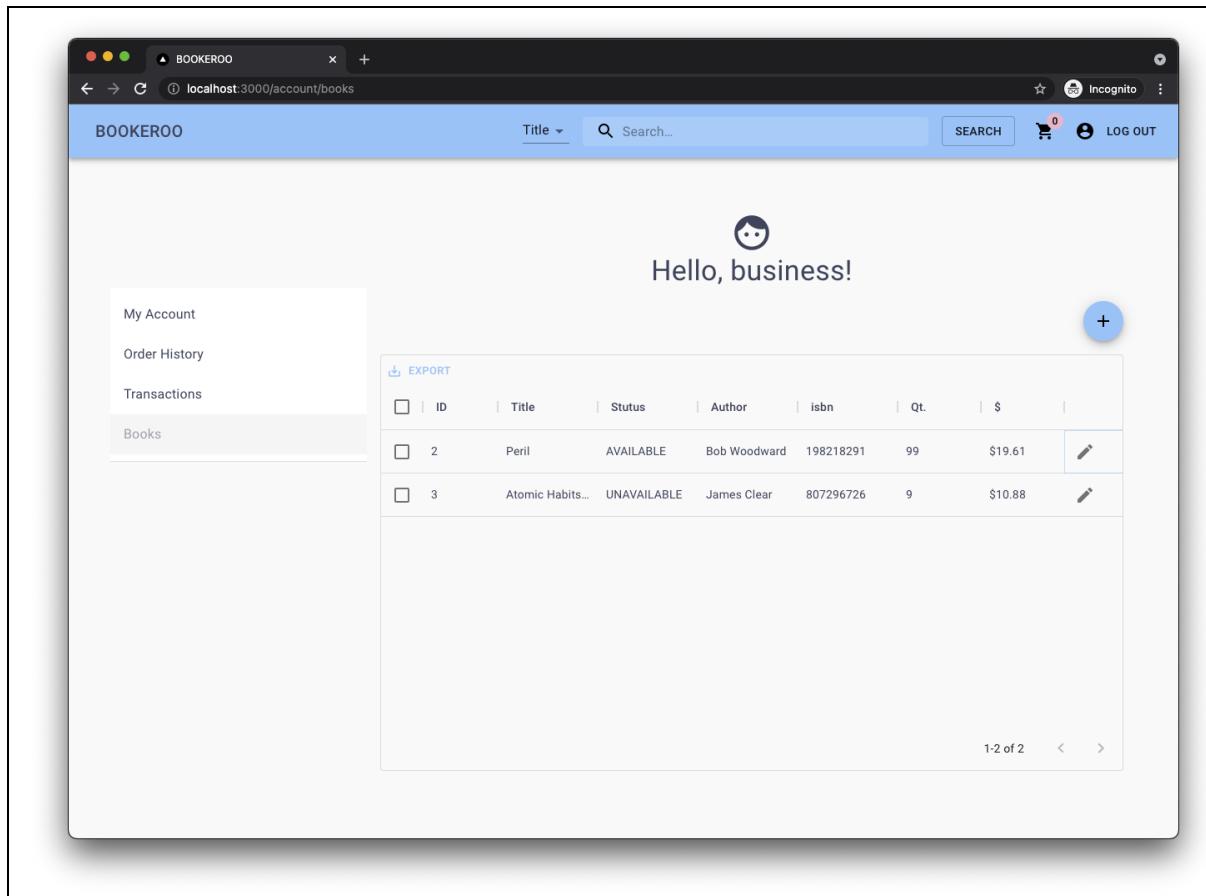
When:

The user checks the book table

Expect:

There is a quantity column for each book

Screenshot:



As a business user, I want to see the total amount of transactions, so I know if I am selling a large number of books.

Given:

A business user is logged in and on the transaction management page

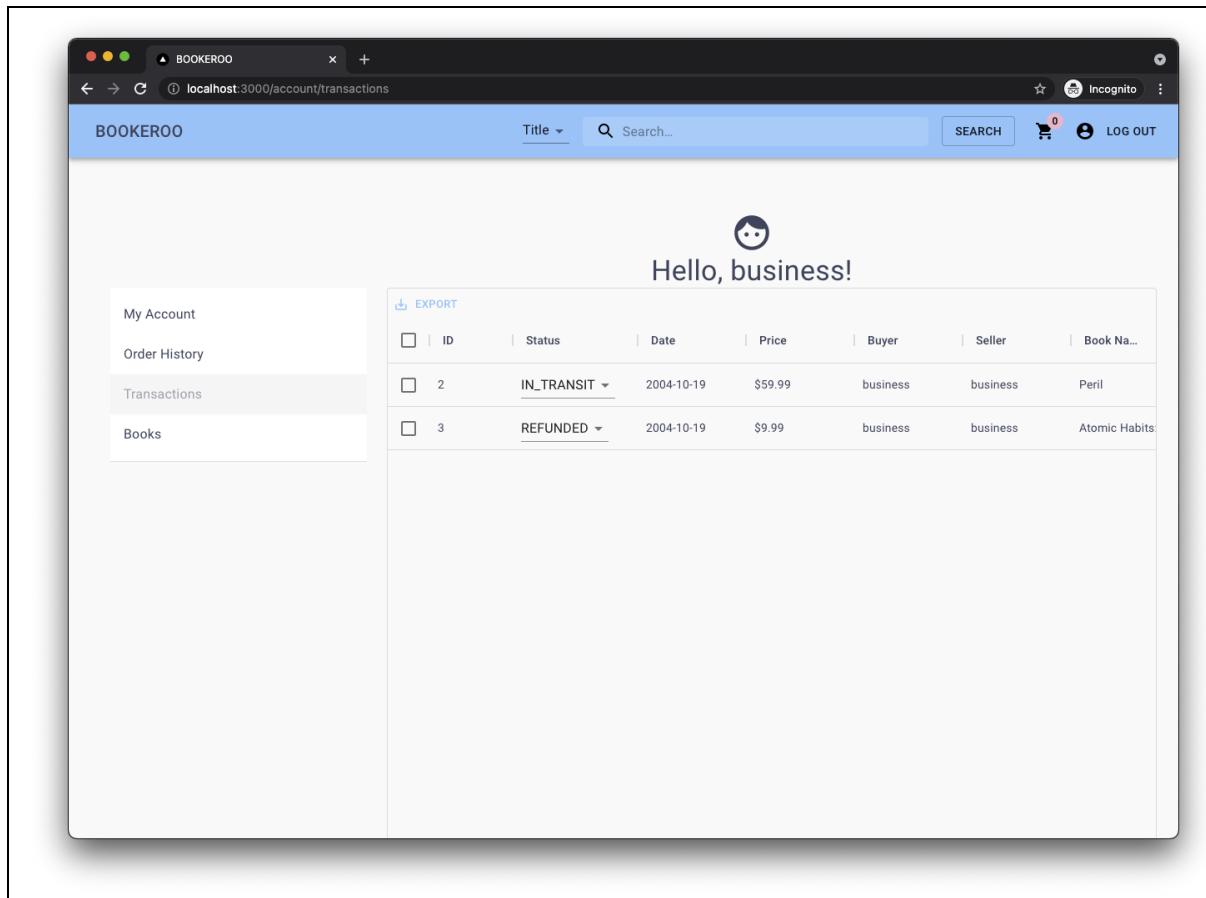
When:

The user checks the transaction table

Expect:

There is a price column for each transaction occurred

Screenshot:



As a customer, I want to be able to shut down my account, so that I cannot worry about my data being stolen

Given:

A user is logged in and on the account page

When:

The user clicks the close account button

Expect:

The account will be terminated.

Screenshot:

The screenshot shows a web application window titled 'BOOKEROO' with the URL 'localhost:3000/account'. The page contains several input fields and buttons:

- Company Name: testing11
- ABN: 1111111111
- SAVE button
- Reset your password section:
 - New Password: (empty input field)
 - Re-enter password: (empty input field)
- RESET button
- Change User Type section:

Submit request to change to Normal User

SUBMIT REQUEST button
- Close Account section:

CLOSE ACCOUNT button

At the bottom of the page, there is a footer note: "SEPT Team Project. Present by [Moving House](#)".

As a customer, I want to remove items from my shopping cart, so I can avoid buying items I don't want.

Given:

There is a book in the shopping cart

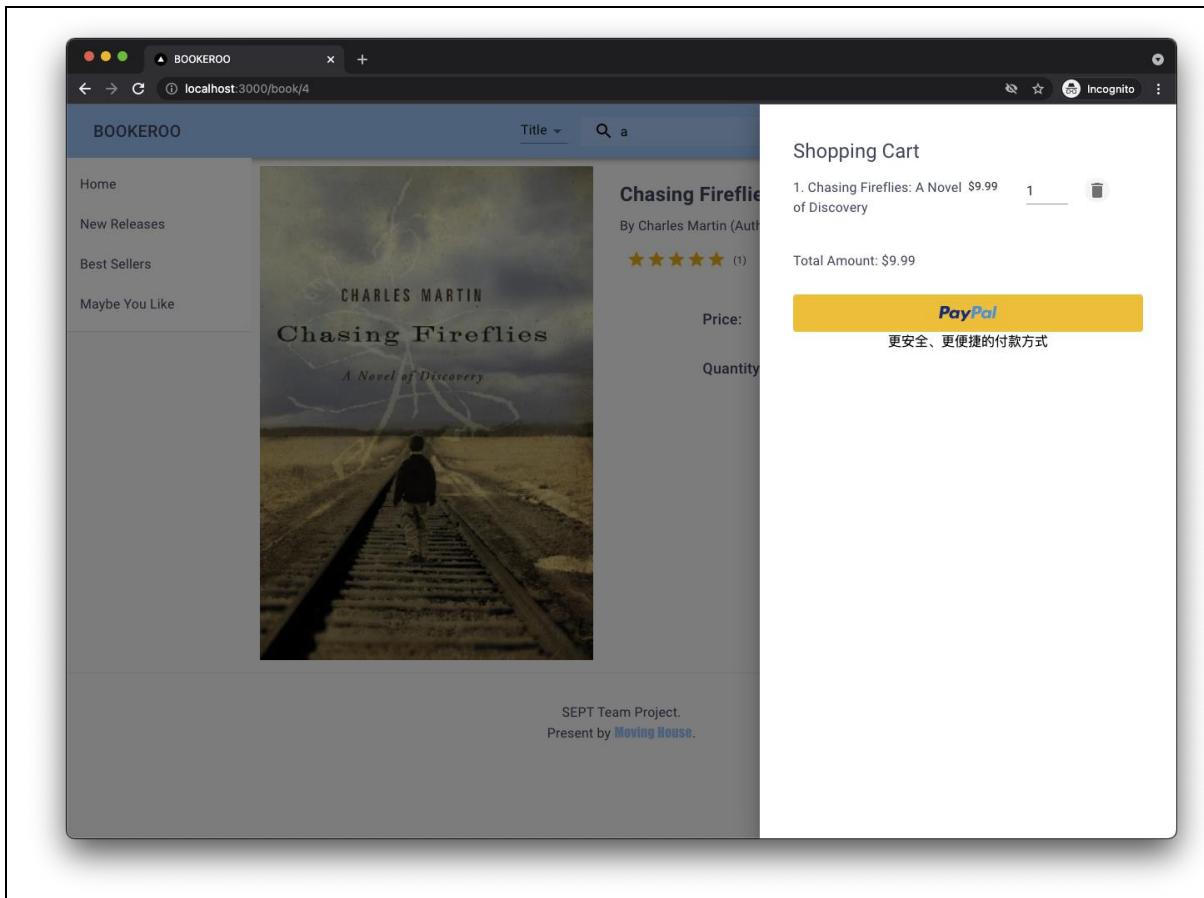
When:

The user clicks the delete button near the book item

Expect:

The item will be removed from the shopping cart

Screenshot:



As a customer, I want to put items to my shopping cart, so I can buy them later

Given:

A user is on a book page

When:

The user clicks the add to cart button with some number of quantity

Expect:

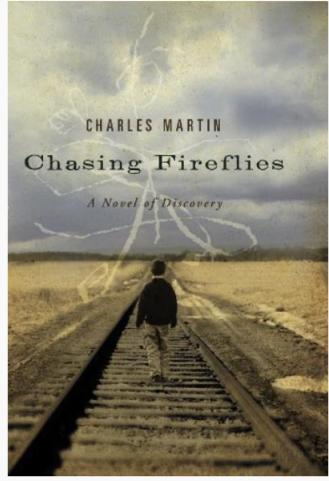
The item will be added to the shopping cart

Screenshot:

BOOKEROO

Title SEARCH 1 LOG OUT

Home New Releases Best Sellers Maybe You Like



Chasing Fireflies: A Novel of Discovery

By Charles Martin (Author) ISBN: 800612389

★★★★★ (1)

Price: \$9.99

Quantity: 2

ADD TO CART

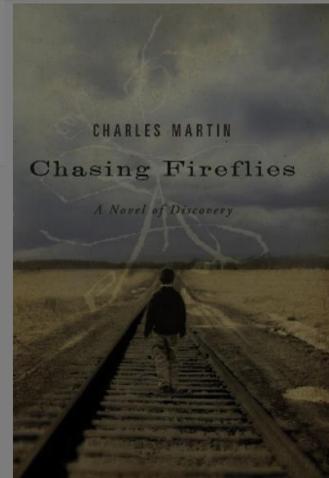
f t in

SEPT Team Project.
Present by [Moving House](#).

BOOKEROO

Title SEARCH 1 Incognito

Home New Releases Best Sellers Maybe You Like



Chasing Fireflies

By Charles Martin (Auth)

★★★★★ (1)

Price:

Quantity:

Shopping Cart

1. Chasing Fireflies: A Novel of Discovery \$9.99 2

Total Amount: \$19.98

PayPal
更安全、更便捷的付款方式

SEPT Team Project.
Present by [Moving House](#).

As a customer, I want to check out my shopping cart with PayPal, so I can finish my purchase

Given:

A user is logged in and on the shopping cart page

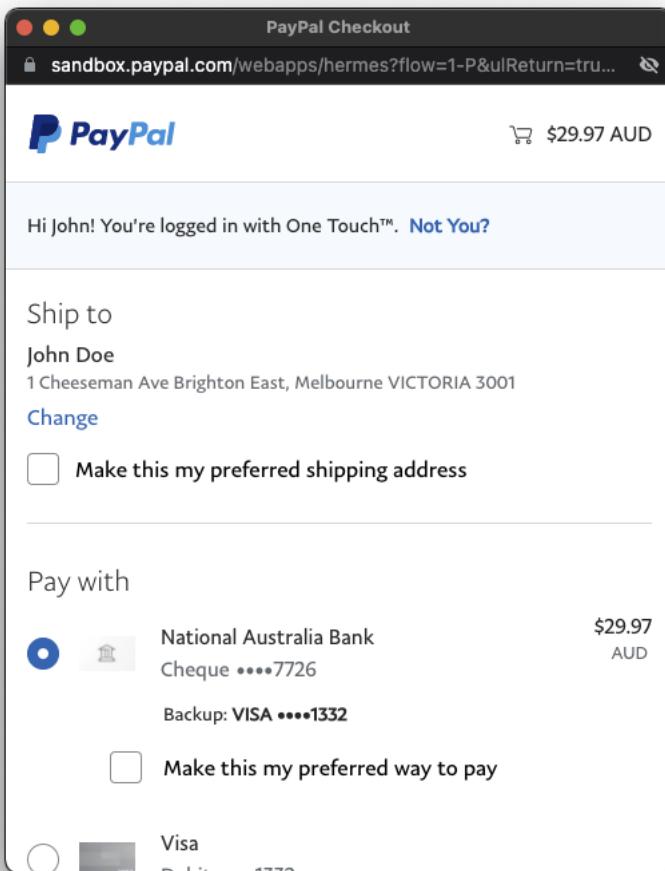
When:

The user clicks the PayPal button

Expect:

The user will take to PayPal to finish the purchase

Screenshot:



Given:

A user is not logged in

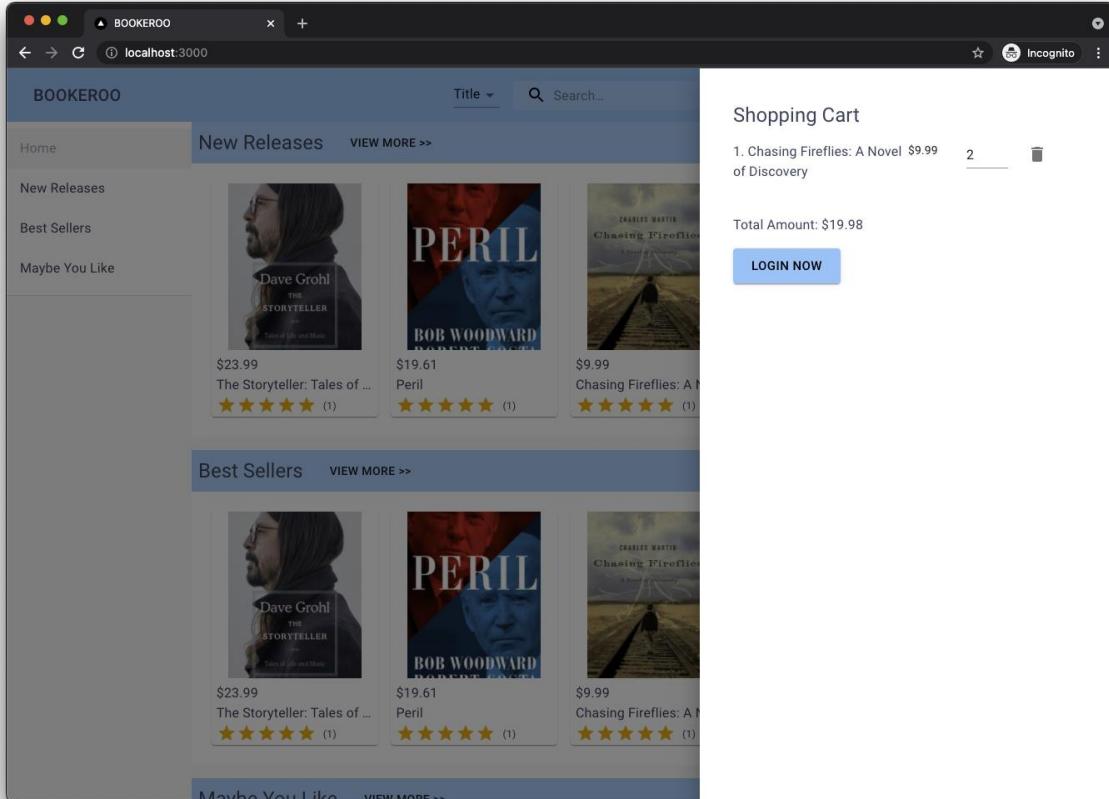
When:

The user opens the shopping cart page

Expect:

The user will see the login button

Screenshot:



As an Admin, I want to see the transaction history of an item, so I know if the right amount of money is being transferred

Given:

The admin is logged in

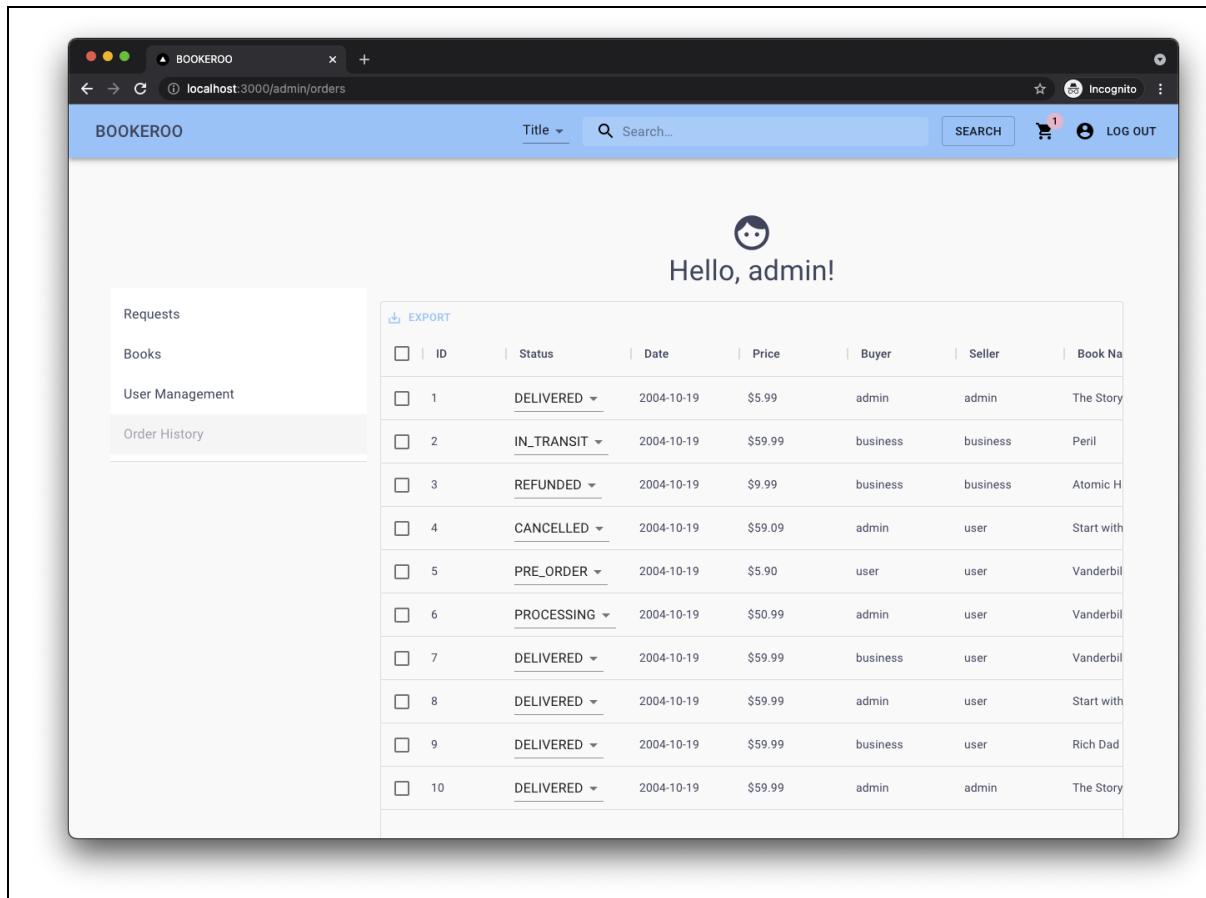
When:

The admin checks the order history page

Expect:

Admin will be able to see all the transactions in a table

Screenshot:



As an admin, I want to view the refund requests from users, so that I can manage all the refund requests.

Given:

The admin is logged in

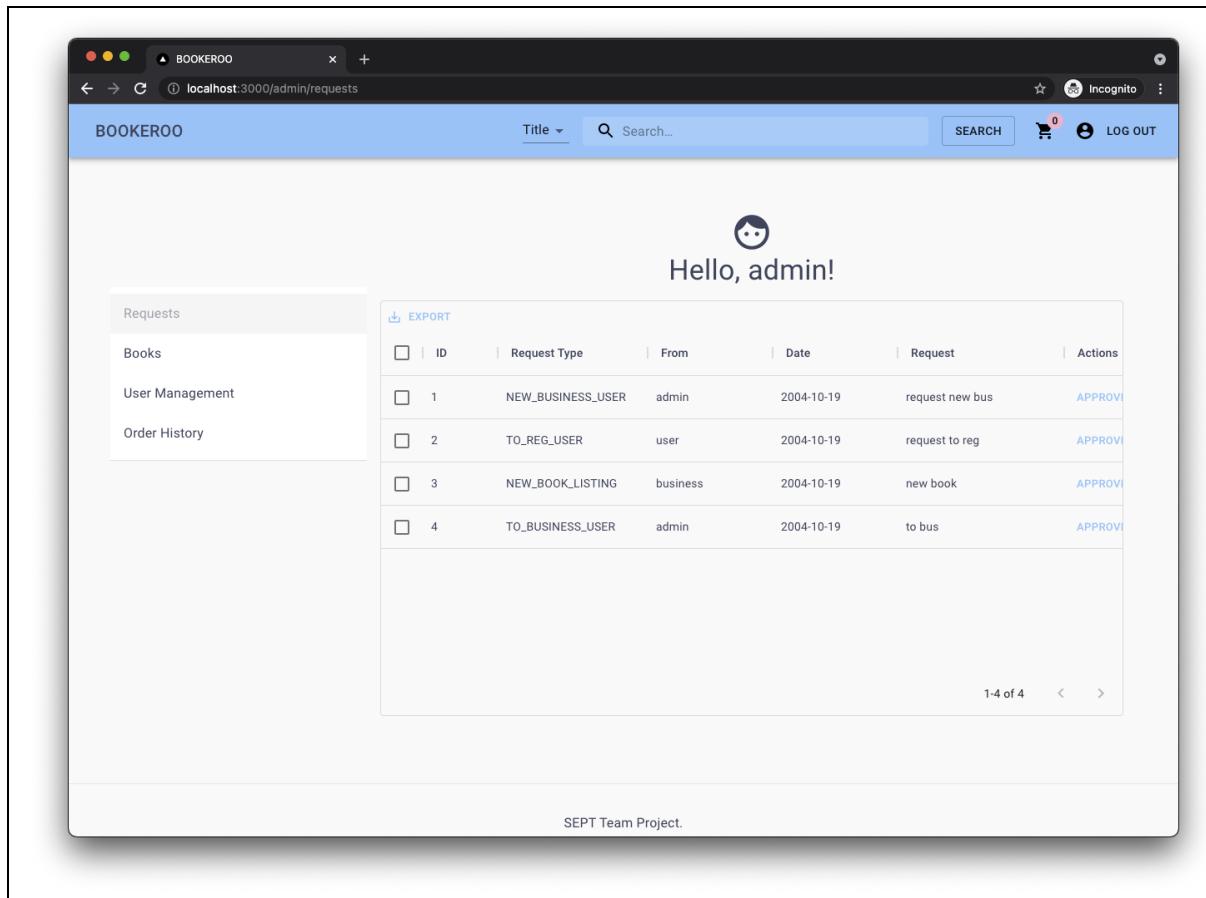
When:

The admin checks the request history page

Expect:

The refund request will be shown if any

Screenshot:



As an admin, I want to approve refund requests from users, so that I can allow the refund to the processed

Given:

The admin is logged in and on the request page

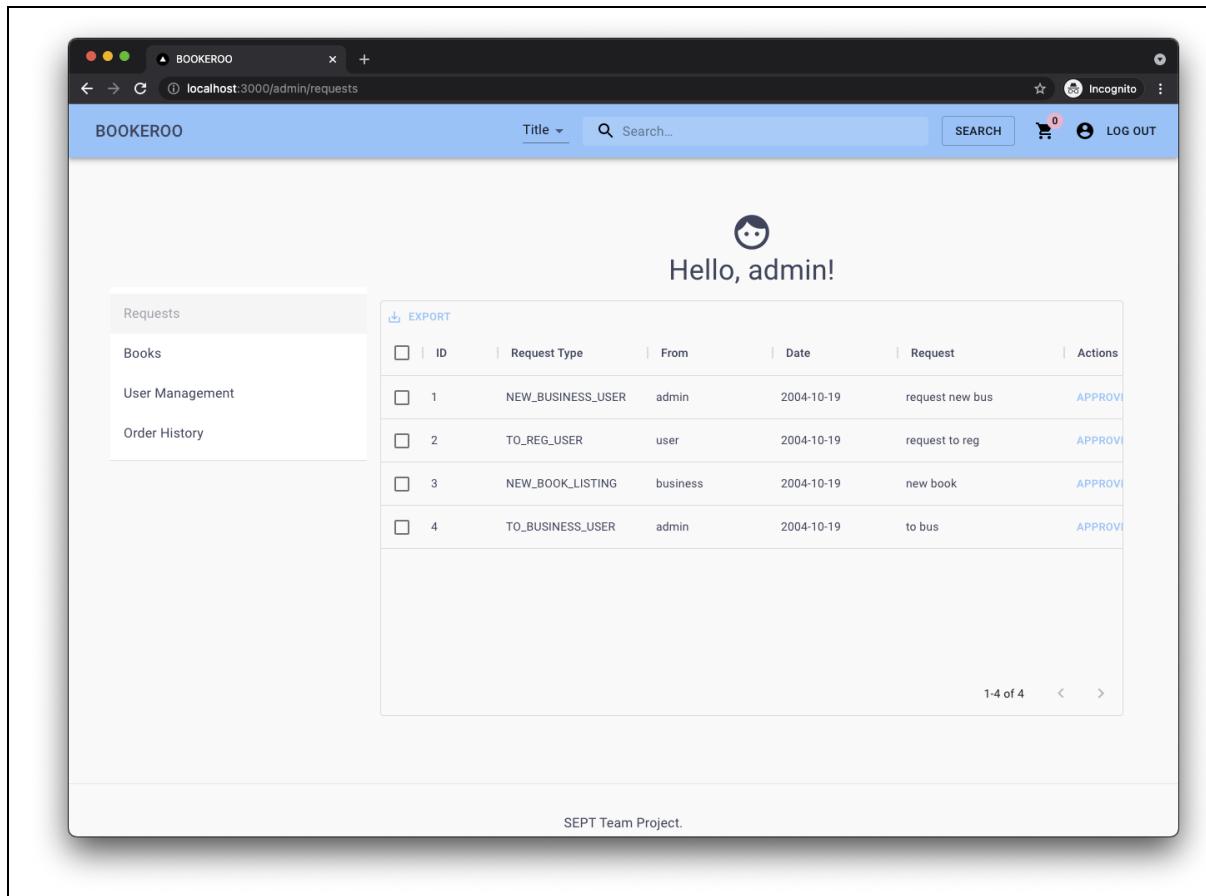
When:

The admin checks approve button for a refund request

Expect:

The refund request will be approved

Screenshot:



As an admin, I want to reject refund requests from users, so that I can deny the refund requests

Given:

The admin is logged in and on the request page

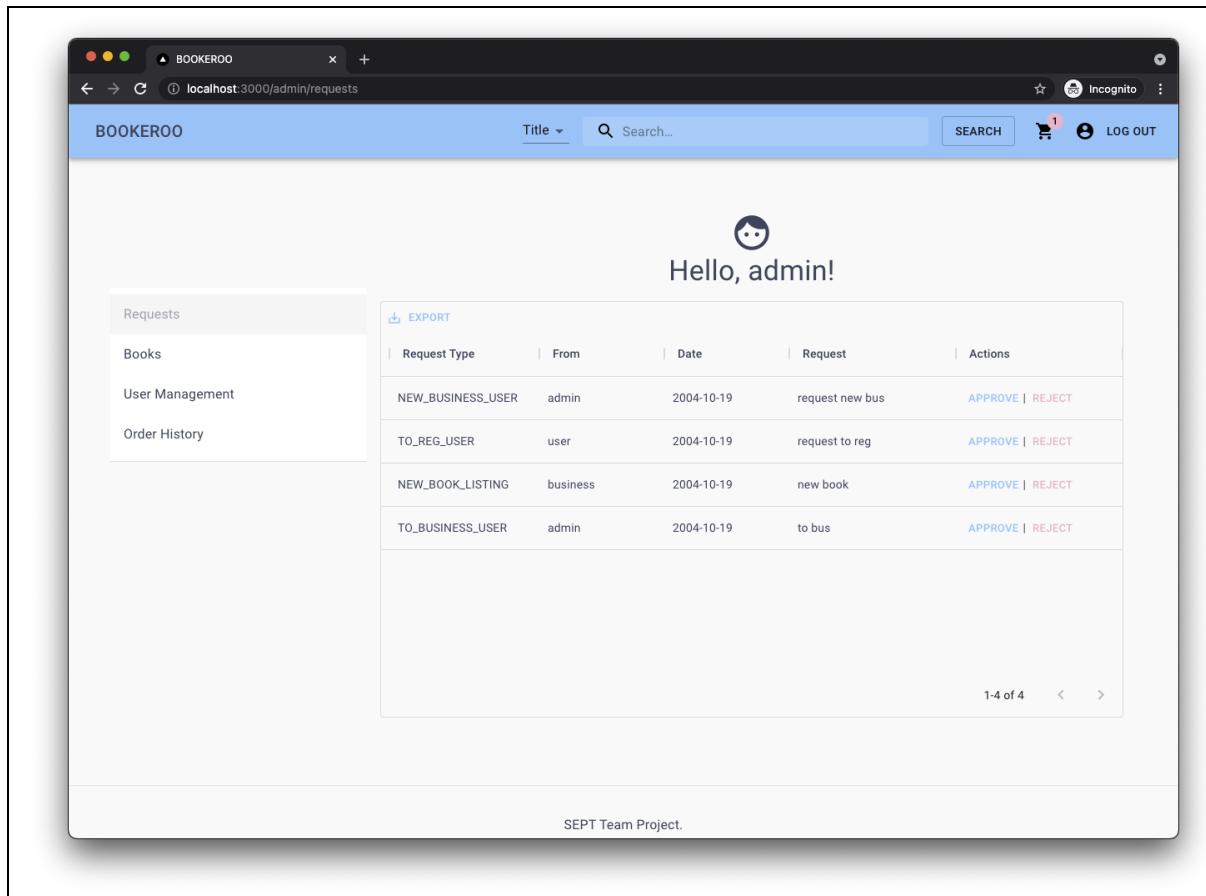
When:

The admin checks reject button for a refund request

Expect:

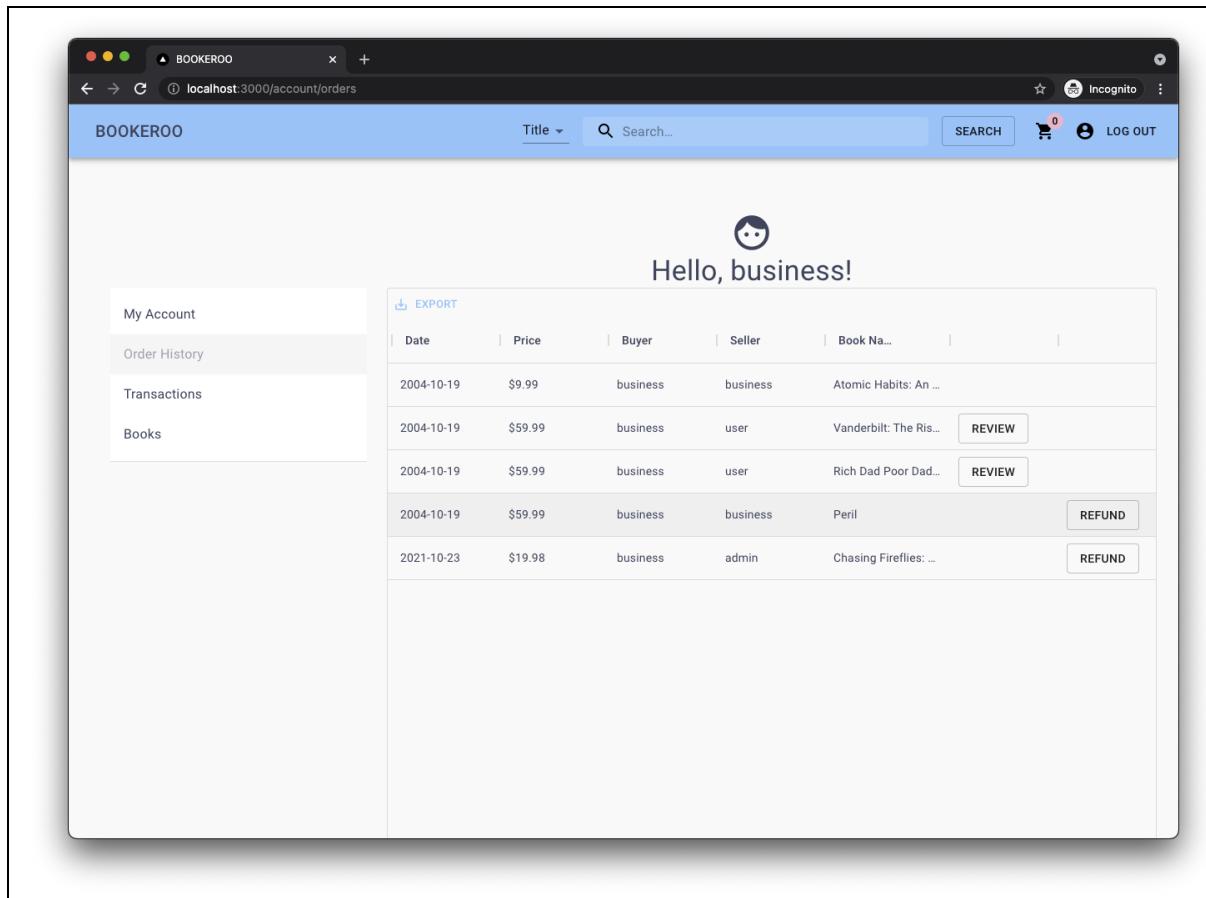
The refund request will be rejected

Screenshot:



As a customer, I want to cancel my order within 2 hours of purchasing, so I can get a full refund.

Given: The user is logged in and on the order page
When: User clicks the refund button for order within 2 hours after purchase
Expect: The transaction will be refunded



As a customer, I want to book items in advance, so I can get the book later when it is available.

Given:

Customer purchased a book which is not available

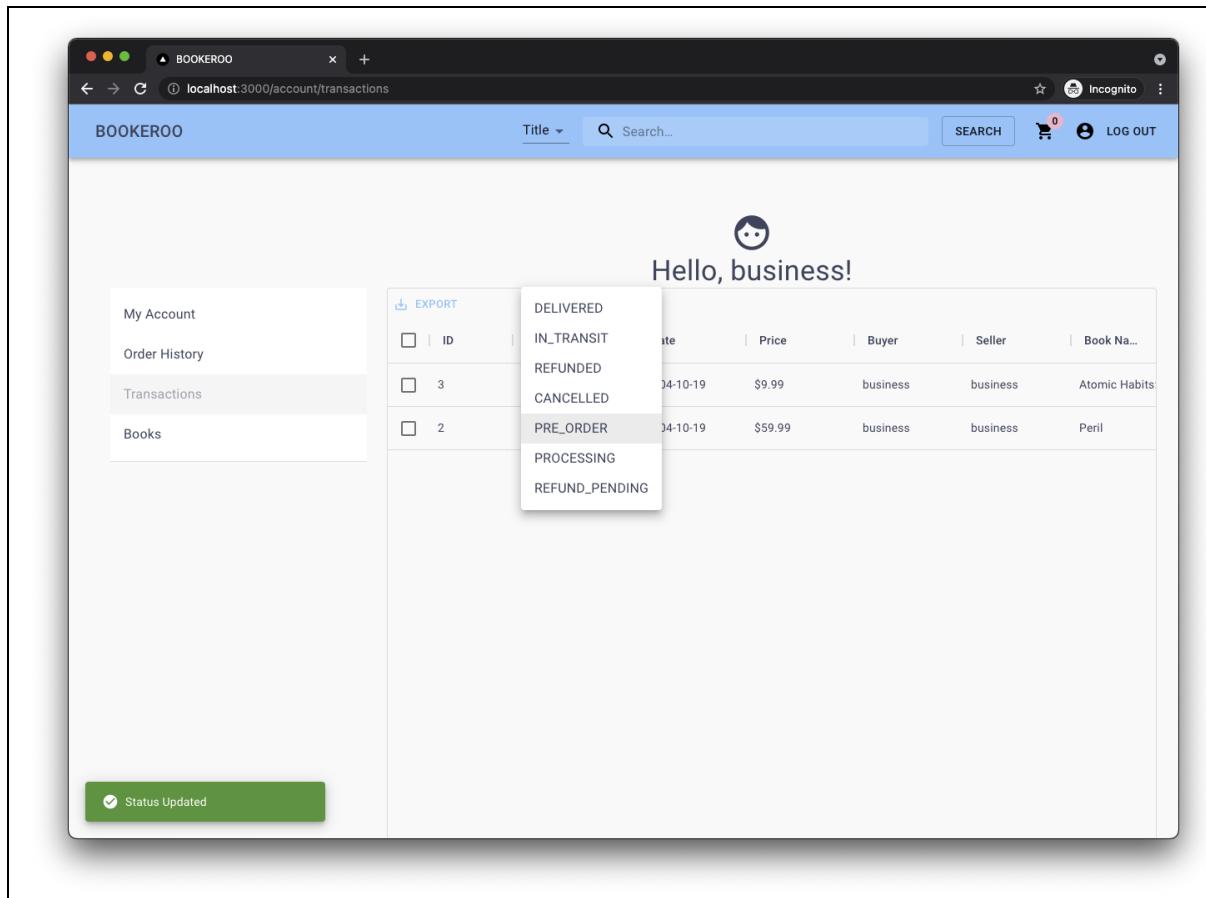
When:

The seller clicks the status on that order and change the status to pre-order

Expect:

The order will be a pre- order

Screenshot:



As a customer, I want to cancel my order after 2 hours of purchasing, so I can get a full refund.

Given:

The user is logged in and on the order page

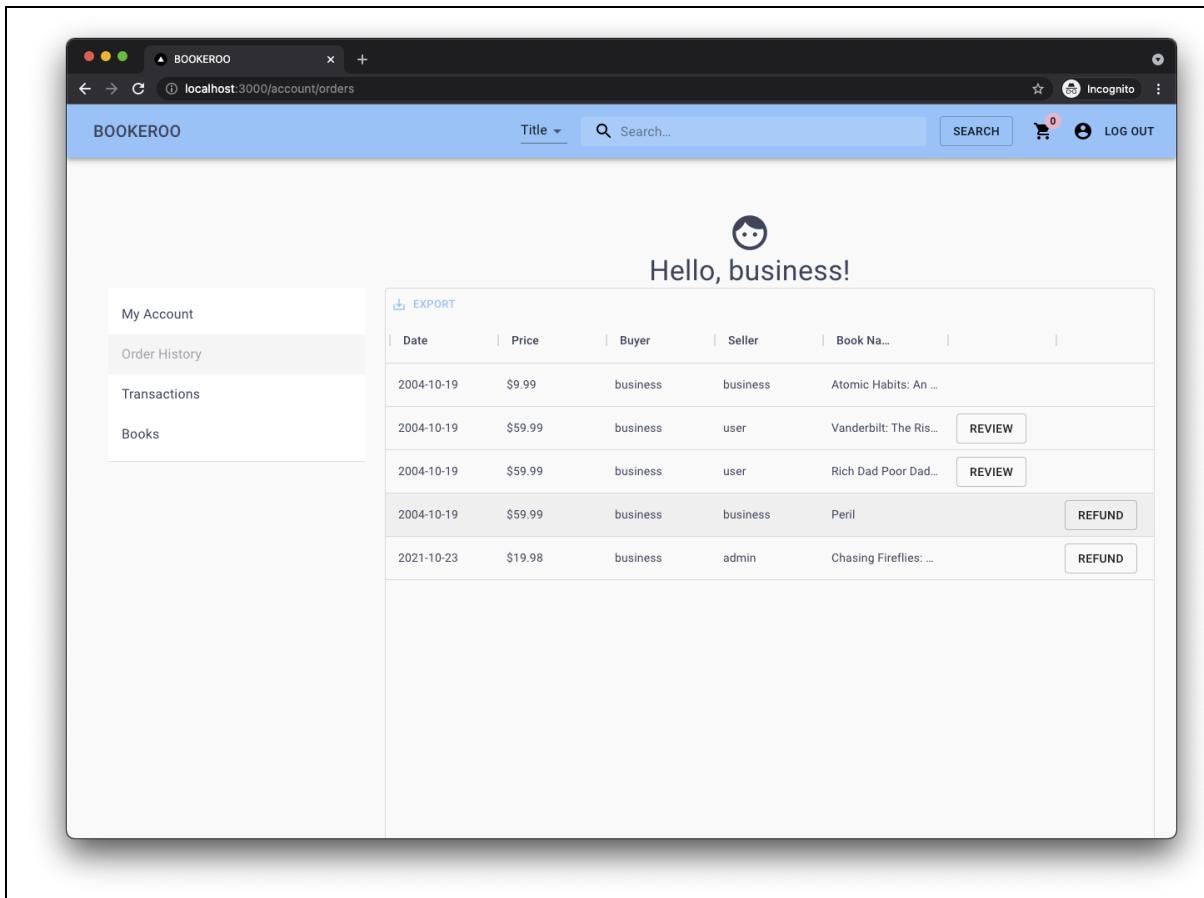
When:

User clicks the refund button for order after 2 hours after purchase

Expect:

A refund request will be created for approval.

Screenshot:



As a business user, I want to be able to upload images to my books, so that I can display previews for each book item.

Given:

The business user is logged in and on the book management page

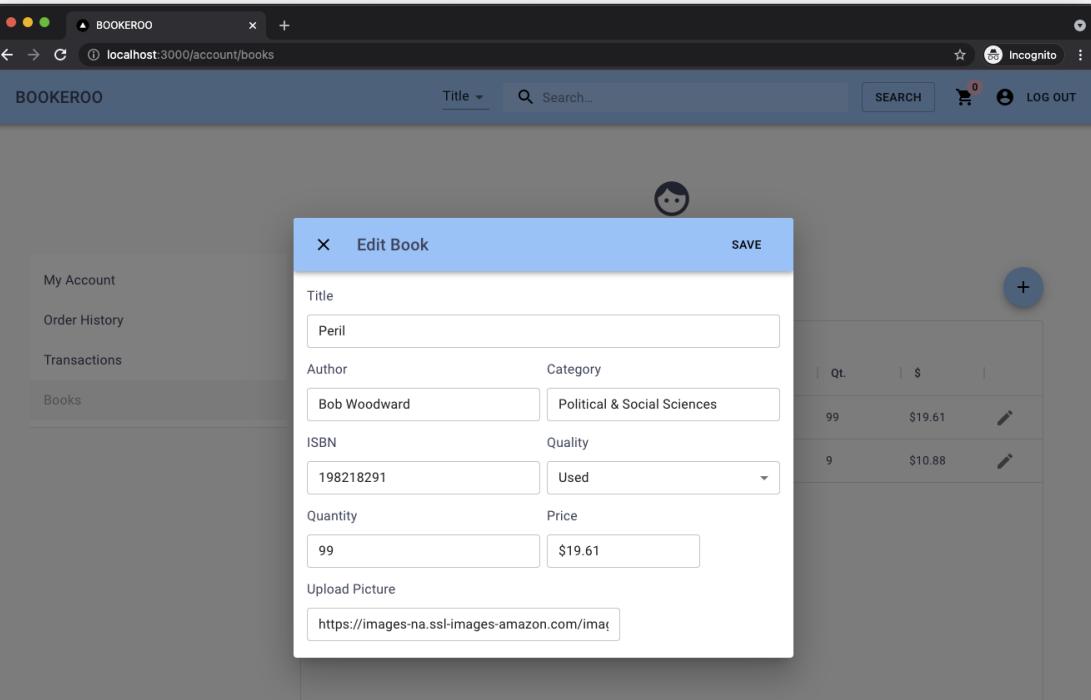
When:

User clicks the edit button for a book

Expect:

An edit dialog with an image URL field will pop up

Screenshot:



Given:

The business user is logged in and on the book management page

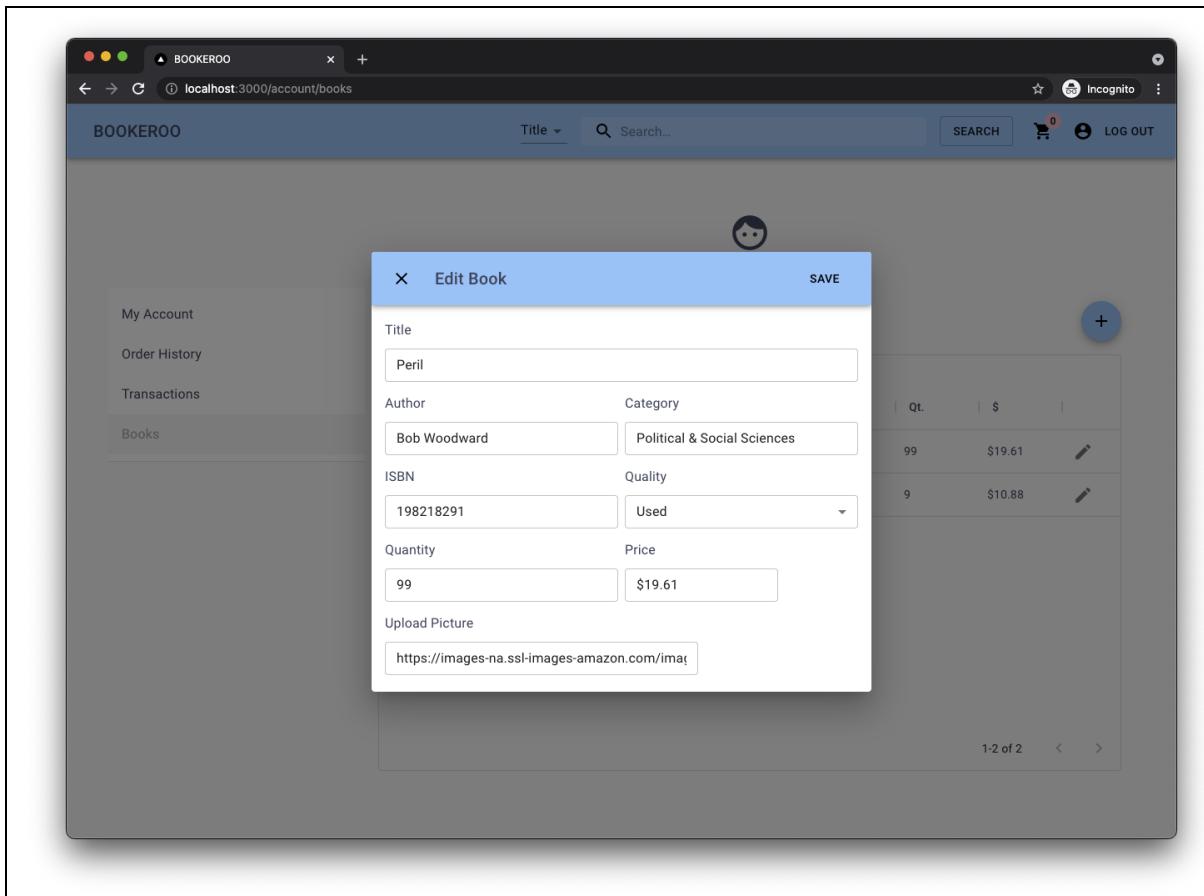
When:

User clicks the edit button for a book

Expect:

An edit dialog with an image URL field will pop up

Screenshot:

**Given:**

The business user is logged in and on the book management page

When:

User clicks the plus button to add a new book to sell

Expect:

An create dialog with an image URL field will pop up

Screenshot:

BOOKEROO ▾ +

localhost:3000/account/books

BOOKEROO

Title Search... SEARCH LOG OUT

+

Create Book

X SAVE

Title

Book Title

Author **Category**

Author Name Category

ISBN **Quality**

ISBN New

Quantity **Price**

Upload Picture

url

1-2 of 2 < >

Qt.	\$
99	\$19.61
9	\$10.88