



# READING LIST APP PROJECT DOCUMENTATION

Asawer Khan

BAE14



## Table Of Contents

### Contents

Table Of Contents .....	1
1. Risk Assessment .....	4
2. Risk Assessment Matrix .....	4
3. Setting up a Spring Project and GitHub Repository .....	5
4. Creating a Book Entity Class.....	6
4.1 Properties.....	6
4.2 Constructors.....	6
4.3 Getters and Setters .....	7
4.4 toString, hashCodes and equals.....	7
5. Connecting to databases.....	8
5.1 File to Switch Between Database Connections.....	8
5.2 H2 Database Connection.....	8
5.3 MySQL Database Connection.....	8
6. Rest API - Controller.....	9
6.1 Constructor .....	9
6.2 Get Methods .....	9
6.3 Create, Update and Delete Methods.....	9
7. Rest API – Service.....	10
7.1 Constructor .....	10
7.2 Get Methods .....	10
7.3 Create, Update and Delete Methods.....	10
8. Rest API – Repo .....	11
8.1 Customer Queries .....	11
9. Postman .....	11
10. Integration Test.....	12
10.1 Test for Create .....	12
10.2 Test for getAll.....	12
10.3 Test for getByID.....	12
10.4 Test for Update .....	12
10.5 Test for Delete .....	12

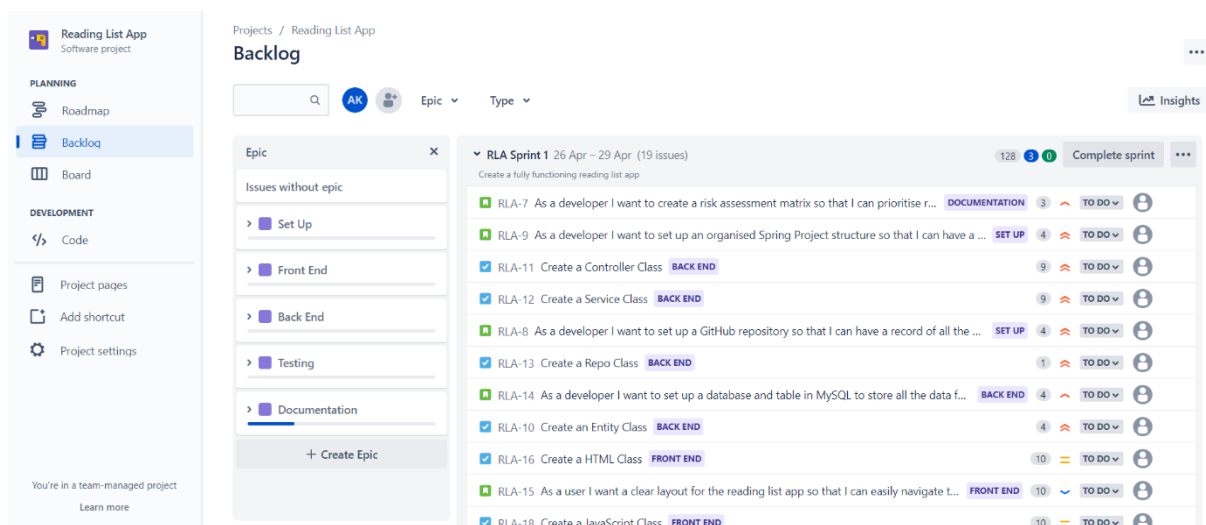
11. Unit Test – Controller.....	13
11.1 Test for getAll.....	13
11.2 Test for getByID.....	13
11.3 Test for Create .....	13
11.4 Test for Update .....	13
11.5 Test for Delete .....	13
12. Unit Test – Service .....	14
12.1 Test for getAll.....	14
12.2 Test for getByID.....	14
12.3 Test for Create .....	14
12.4 Test for Update .....	14
12.5 Test for Delete .....	14
13. Test Coverage .....	15
13.1 Integration Test.....	15
13.2 Unit Test – Controller.....	15
13.3 Unit Test – Service .....	15
14. Front-End .....	16
14.1 Planning.....	16

## Setup Jira Board

Created a Jira Board which consisted of a Sprint made up of 5 Epics:

- Setup
- Documentation
- Frontend
- Backend
- Testing

Each Epic included stories, mostly from the perspective of myself as the developer, but also some as the user. The Epics also included tasks. Each story and task were given a story point and a priority label. This will be revisited and updated as stories and tasks are completed or to create new ones.



## 1. Risk Assessment

Risk Assessment						
Description	Evaluation	Likelihood	Impact Level	Responsibility	Response	Control Measures
The developer falls ill	The project will not be completed on time	Low	High	N/A	Request an extension	Increase level of hygiene practices
The JUnit tests fail	The API will not work as desired	High	Medium	Developer	The code in the program and the test code need to be checked for mistakes and corrected	Ensure test code is correctly written whilst it is being written and test the functionality of new methods as they are written to identify mistakes in code before testing
Unit and integration tests don't reach 80% test coverage	Lower project mark	Medium	Medium	Developer	Contact trainers for advice to increase test coverage	Carry out self study and research to increase knowledge of testing
The project is not completed in time	Potential to fail the project	Low	High	Developer	Request an extension	Manage time taken on tasks appropriately and keep well organised
The full requirements of the project brief have not been achieved	Lower project mark	Low	Medium	Developer	Read project brief to identify missed requirement and include it in your project submission ASAP	Create stories and tasks on the project jiraboard based on each project brief requirement and set these to highest priority
Connection to the database can't be established	The CRUD requests will fail	Low	High	Developer	View recorded lectures to follow step-by-step instructions to identify mistake, or contact trainers for guidance	Solidify knowledge on connecting the database or follow step-by-step tutorial in recorded lectures if unsure
Frontend and Backend are not linked correctly	The user will not be able to access the backend functionality	High	High	Developer	View recorded lectures to follow step-by-step instructions to identify mistake, or contact trainers for guidance	Solidify knowledge on linking the Frontend and Backend or follow step-by-step tutorial in recorded lectures if unsure

## 2. Risk Assessment Matrix



### 3. Setting up a Spring Project and GitHub Repository

```
MINGW64:/c/Users/User/BAE NSAC/Reading List Project

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (master)
$ git branch
* master

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (master)
$ git branch -M main

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (main)
$ git branch
* main

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (main)
$ git remote add origin https://github.com/AsawerKhan/ReadingListProject.git

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (main)
$ git remote -v
origin https://github.com/AsawerKhan/ReadingListProject.git (fetch)
origin https://github.com/AsawerKhan/ReadingListProject.git (push)

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (main)
$ git push -u origin main
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 8 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (25/25), 38.90 KiB | 5.35 MiB/s, done.
Total 25 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/AsawerKhan/ReadingListProject.git
 * [new branch] main -> main
branch 'main' set up to track 'origin/main'.

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (main)
$ git branch
* main

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (main)
$ git checkout -b dev
Switched to a new branch 'dev'

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (dev)
$ git branch
* dev
  main

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (dev)
$ git push -u origin dev
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:   https://github.com/AsawerKhan/ReadingListProject/pull/new/dev
remote:
To https://github.com/AsawerKhan/ReadingListProject.git
 * [new branch] dev -> dev
branch 'dev' set up to track 'origin/dev'.

User@DESKTOP-S581BQ0 MINGW64 ~/BAE NSAC/Reading List Project (dev)
$ |
```

AsawerKhan / ReadingListProject Public

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

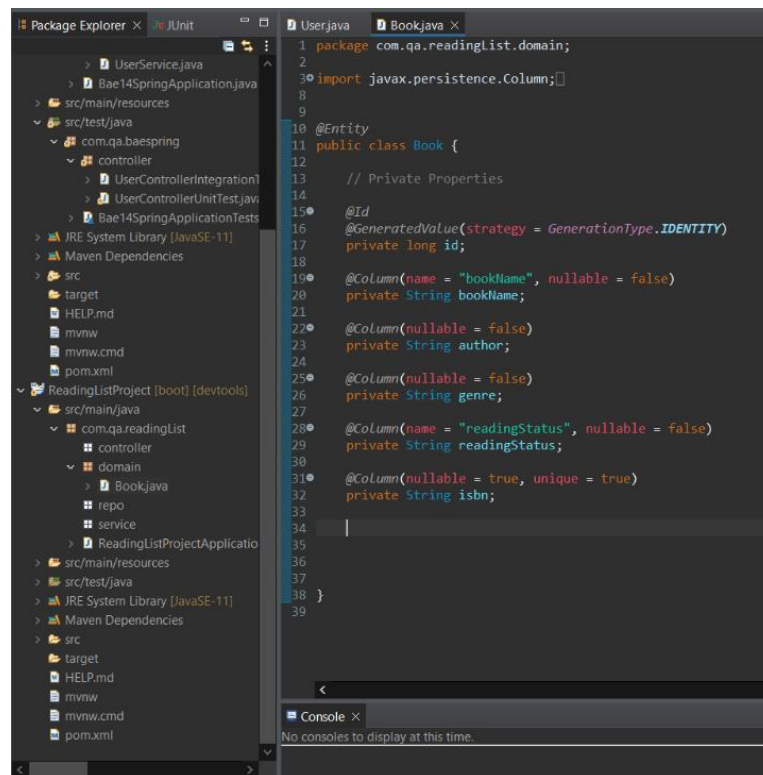
main 2 branches 0 tags

Go to file Add file Code

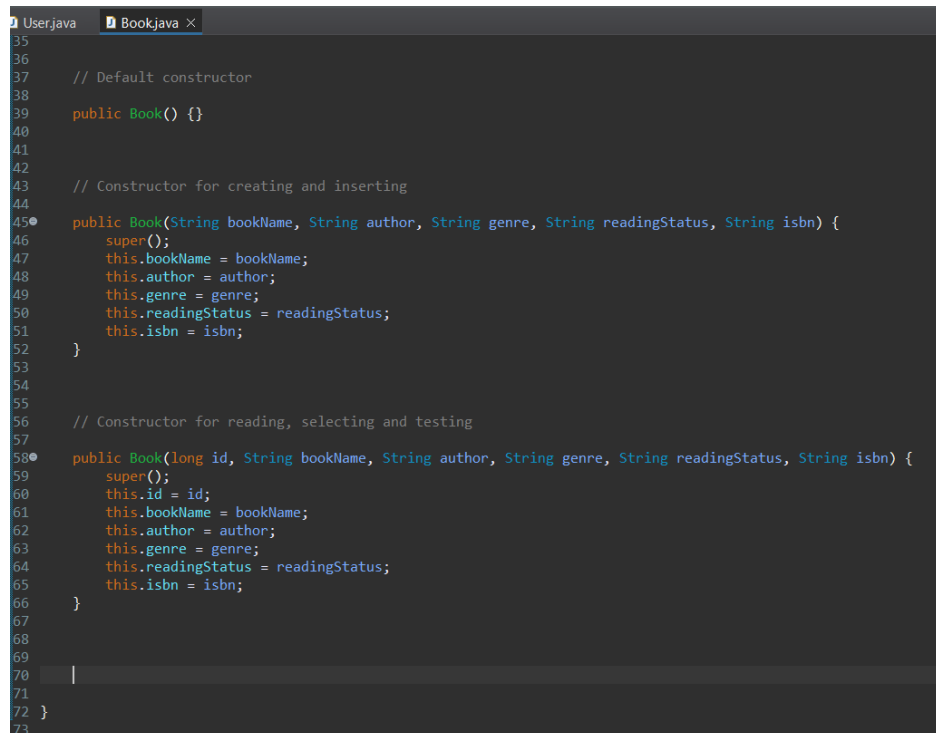
Asawer Khan AK: init commit		4b77ee0 1 hour ago 1 commit
📁 .mvn/wrapper	AK: init commit	1 hour ago
📁 src	AK: init commit	1 hour ago
📄 .gitignore	AK: init commit	1 hour ago
📄 mvnw	AK: init commit	1 hour ago
📄 mvnw.cmd	AK: init commit	1 hour ago
📄 pom.xml	AK: init commit	1 hour ago

## 4. Creating a Book Entity Class

### 4.1 Properties



### 4.2 Constructors



## 4.3 Getters and Setters

```
User.java  Book.java ×
69 // Getters and setters
70 public long getId() {
71     return id;
72 }
73 public void setId(long id) {
74     this.id = id;
75 }
76
77 public String getBookName() {
78     return bookName;
79 }
80 public void setBookName(String bookName) {
81     this.bookName = bookName;
82 }
83
84 public String getAuthor() {
85     return author;
86 }
87 public void setAuthor(String author) {
88     this.author = author;
89 }
90
91 public String getGenre() {
92     return genre;
93 }
94 public void setGenre(String genre) {
95     this.genre = genre;
96 }
97
98 public String getReadingStatus() {
99     return readingStatus;
100 }
101 public void setReadingStatus(String readingStatus) {
102     this.readingStatus = readingStatus;
103 }
104
105 public String getIsbn() {
```

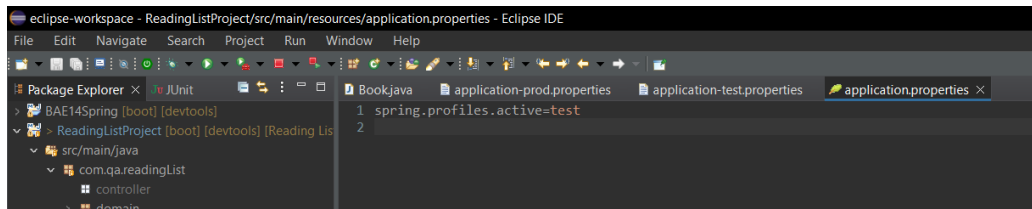
## 4.4 toString, hashCode and equals

```
User.java  Book.java ×
113 }
114
115 // toString
116
117 @Override
118 public String toString() {
119     return "Book [id=" + id + ", bookName=" + bookName + ", author=" + author + ", genre=" + genre
120         + ", readingStatus=" + readingStatus + ", isbn=" + isbn + "]";
121 }
122
123 // hashCode and equals
124 // hashCode
125
126 @Override
127 public int hashCode() {
128     return Objects.hash(author, bookName, genre, id, isbn, readingStatus);
129 }
130
131 // equals
132
133 @Override
134 public boolean equals(Object obj) {
135     if (this == obj)
136         return true;
137     if (obj == null)
138         return false;
139     if (getClass() != obj.getClass())
140         return false;
141     Book other = (Book) obj;
142     return Objects.equals(author, other.author) && Objects.equals(bookName, other.bookName)
143         && Objects.equals(genre, other.genre) && id == other.id && Objects.equals(isbn, other.isbn)
144         && Objects.equals(readingStatus, other.readingStatus);
145 }
146
147 }
```

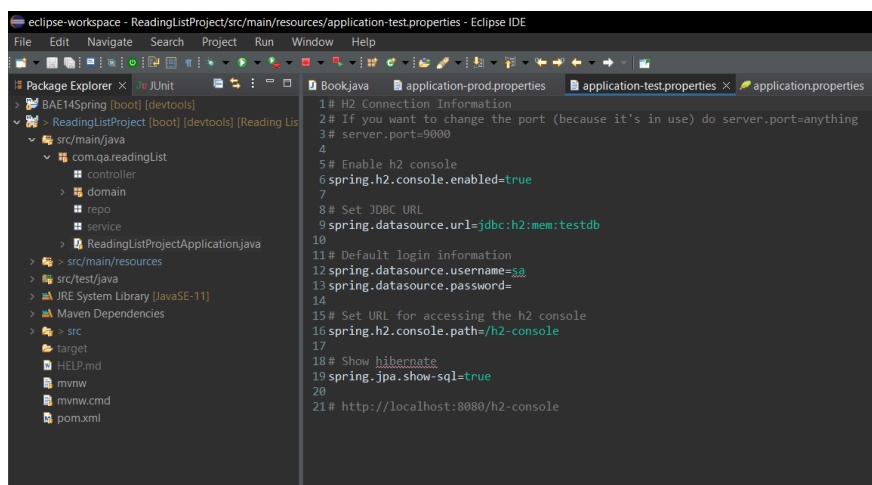


## 5. Connecting to databases

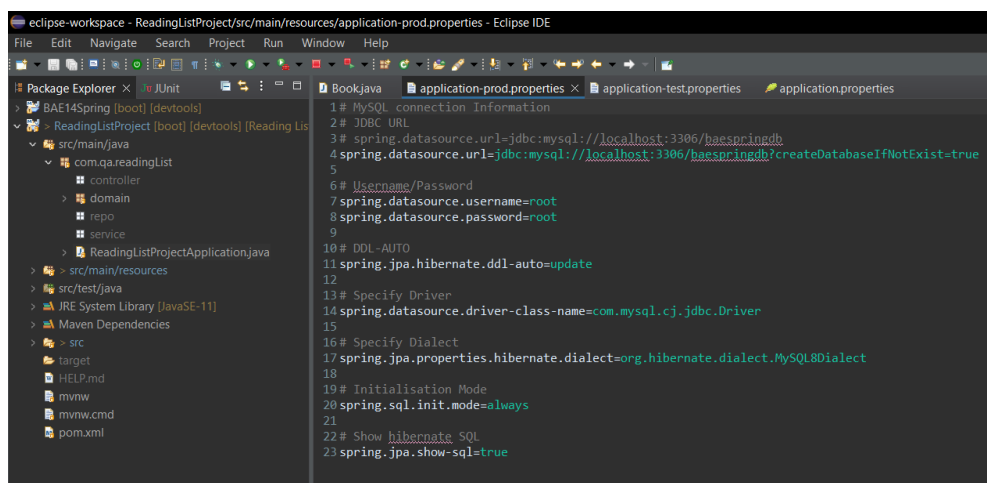
### 5.1 File to Switch Between Database Connections



### 5.2 H2 Database Connection

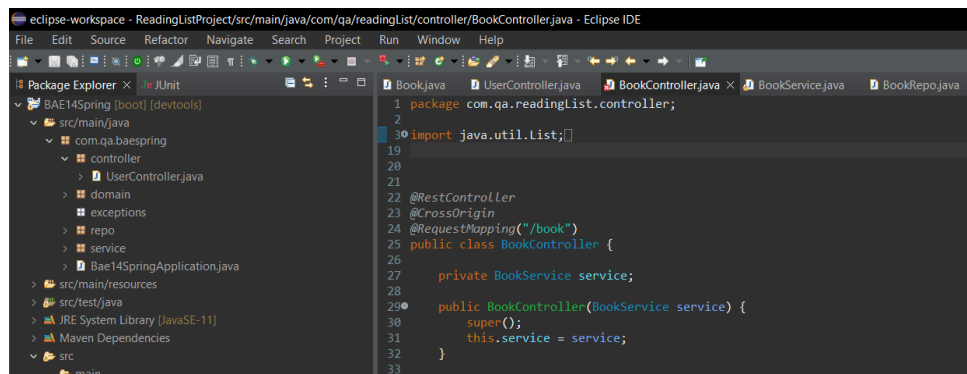


### 5.3 MySQL Database Connection



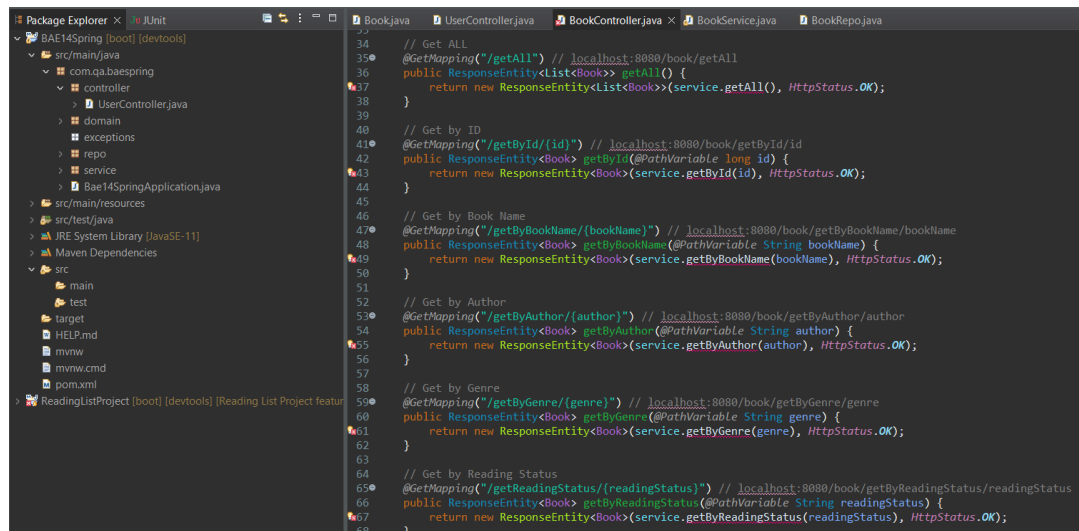
## 6. Rest API - Controller

### 6.1 Constructor



```
1 package com.qa.readingList.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 @RestController
23 @CrossOrigin
24 @RequestMapping("/book")
25 public class BookController {
26
27     private BookService service;
28
29     public BookController(BookService service) {
30         super();
31         this.service = service;
32     }
33 }
```

### 6.2 Get Methods



```
34 // Get ALL
35 @GetMapping("/getAll") // localhost:8080/book/getAll
36 public ResponseEntity<List<Book>> getAll() {
37     return new ResponseEntity<List<Book>>(service.getAll(), HttpStatus.OK);
38 }
39
40 // Get by ID
41 @GetMapping("/getId/{id}") // localhost:8080/book/getId/{id}
42 public ResponseEntity<Book> getId(@PathVariable long id) {
43     return new ResponseEntity<Book>(service.getId(id), HttpStatus.OK);
44 }
45
46 // Get by Book Name
47 @GetMapping("/getByBookName/{bookName}") // localhost:8080/book/getByBookName/{bookName}
48 public ResponseEntity<Book> getByBookName(@PathVariable String bookName) {
49     return new ResponseEntity<Book>(service.getByBookName(bookName), HttpStatus.OK);
50 }
51
52 // Get by Author
53 @GetMapping("/getByAuthor/{author}") // localhost:8080/book/getByAuthor/{author}
54 public ResponseEntity<Book> getByAuthor(@PathVariable String author) {
55     return new ResponseEntity<Book>(service.getByAuthor(author), HttpStatus.OK);
56 }
57
58 // Get by Genre
59 @GetMapping("/getByGenre/{genre}") // localhost:8080/book/getByGenre/{genre}
60 public ResponseEntity<Book> getByGenre(@PathVariable String genre) {
61     return new ResponseEntity<Book>(service.getByGenre(genre), HttpStatus.OK);
62 }
63
64 // Get by Reading Status
65 @GetMapping("/getReadingStatus/{readingStatus}") // localhost:8080/book/getByReadingStatus/{readingStatus}
66 public ResponseEntity<Book> getByReadingStatus(@PathVariable String readingStatus) {
67     return new ResponseEntity<Book>(service.getByReadingStatus(readingStatus), HttpStatus.OK);
68 }
69 }
```

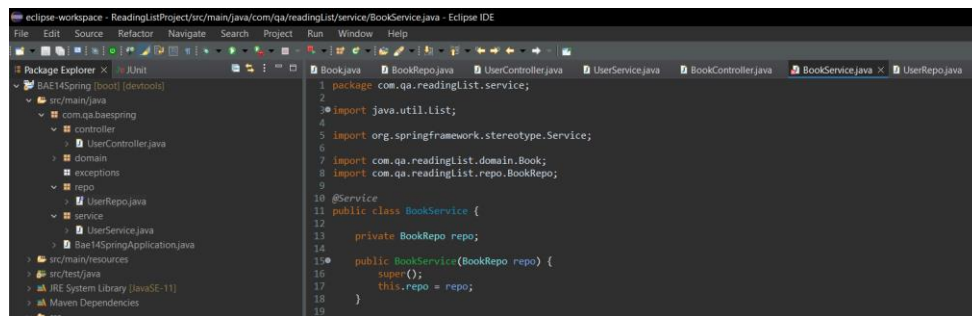
### 6.3 Create, Update and Delete Methods



```
75
76 // Create a Book (Post)
77 @PostMapping("/create") // localhost:8080/book/create
78 public ResponseEntity<Book> create(@RequestBody Book book) {
79     return new ResponseEntity<Book>(service.create(book), HttpStatus.CREATED);
80 }
81
82 // Update a Book (Put)
83 @PutMapping("/update/{id}") // localhost:8080/book/update/{id}
84 public ResponseEntity<Book> update(@PathVariable long id, @RequestBody Book book) {
85     return new ResponseEntity<Book>(service.update(id, book), HttpStatus.ACCEPTED);
86 }
87
88 // Delete a Book
89 @DeleteMapping("/delete/{id}") // localhost:8080/book/delete/{id}
90 public ResponseEntity<Boolean> delete(@PathVariable long id) {
91     return (service.delete(id)) ? new ResponseEntity<Boolean>(HttpStatus.NO_CONTENT) :
92         new ResponseEntity<Boolean>(HttpStatus.INTERNAL_SERVER_ERROR);
93 }
94 }
95
96
```

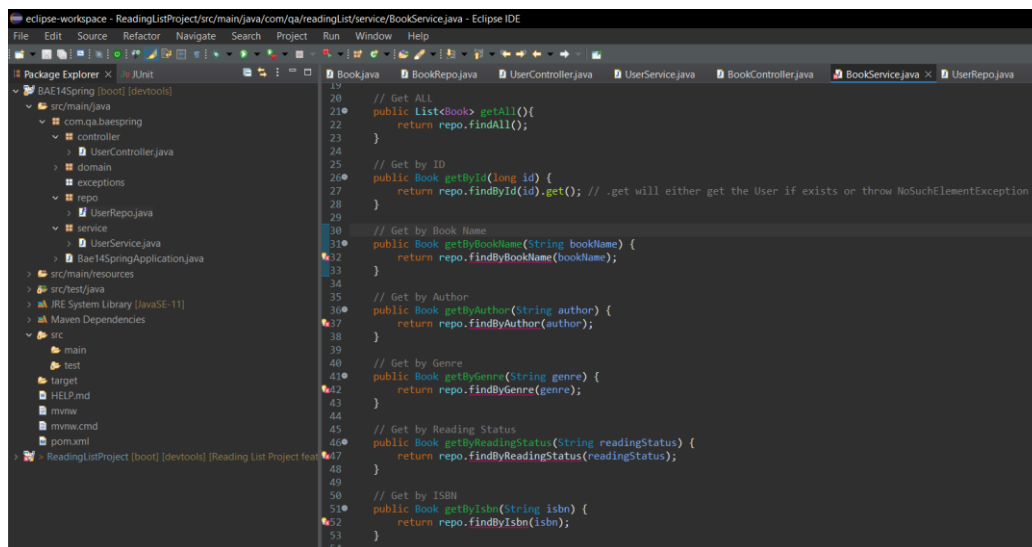
## 7. Rest API – Service

### 7.1 Constructor



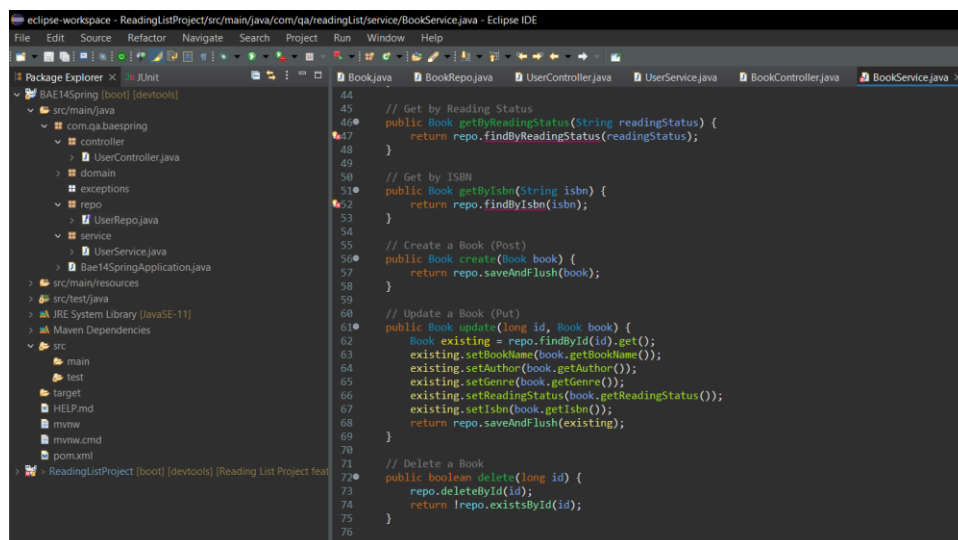
```
1 package com.qa.readinglist.service;
2
3 import java.util.List;
4
5 import org.springframework.stereotype.Service;
6
7 import com.qa.readinglist.domain.Book;
8 import com.qa.readinglist.repo.BookRepo;
9
10 @Service
11 public class BookService {
12
13     private BookRepo repo;
14
15     public BookService(BookRepo repo) {
16         super();
17         this.repo = repo;
18     }
19 }
```

### 7.2 Get Methods



```
19
20 // Get ALL
21 public List<Book> getAll(){
22     return repo.findAll();
23 }
24
25 // Get by ID
26 public Book getById(long id) {
27     return repo.findById(id).get(); // .get will either get the User if exists or throw NoSuchElementException
28 }
29
30 // Get by Book Name
31 public Book getByBookName(String bookName) {
32     return repo.findByName(bookName);
33 }
34
35 // Get by Author
36 public Book getByAuthor(String author) {
37     return repo.findByAuthor(author);
38 }
39
40 // Get by Genre
41 public Book getByGenre(String genre) {
42     return repo.findByGenre(genre);
43 }
44
45 // Get by Reading Status
46 public Book getByReadingStatus(String readingStatus) {
47     return repo.findByReadingStatus(readingStatus);
48 }
49
50 // Get by ISBN
51 public Book getByIsbn(String isbn) {
52     return repo.findByIsbn(isbn);
53 }
54 }
```

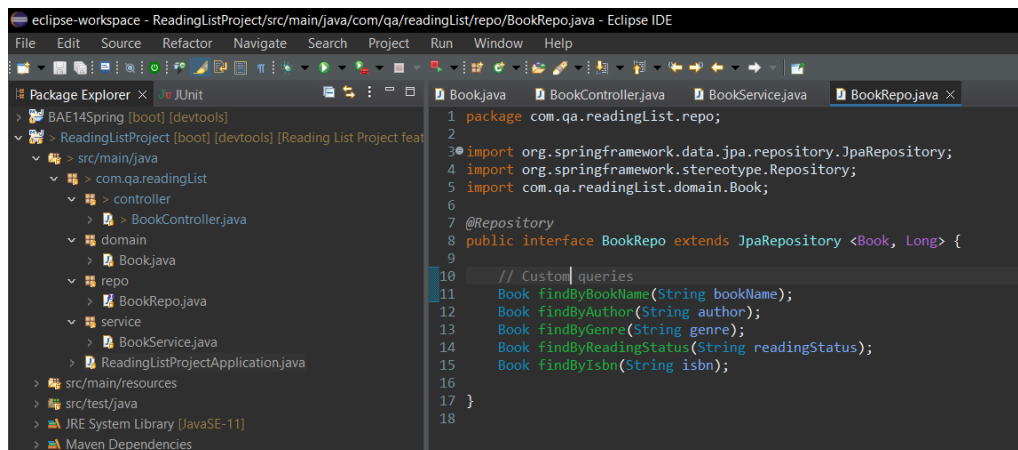
### 7.3 Create, Update and Delete Methods



```
44
45 // Get by Reading Status
46 public Book getByReadingStatus(String readingStatus) {
47     return repo.findByReadingStatus(readingStatus);
48 }
49
50 // Get by ISBN
51 public Book getByIsbn(String isbn) {
52     return repo.findByIsbn(isbn);
53 }
54
55 // Create a Book (Post)
56 public Book create(Book book) {
57     return repo.saveAndFlush(book);
58 }
59
60 // Update a Book (Put)
61 public Book update(long id, Book book) {
62     Book existing = repo.findById(id).get();
63     existing.setBookName(book.getBookName());
64     existing.setAuthor(book.getAuthor());
65     existing.setGenre(book.getGenre());
66     existing.setReadingStatus(book.getReadingStatus());
67     existing.setIsbn(book.getIsbn());
68     return repo.saveAndFlush(existing);
69 }
70
71 // Delete a Book
72 public boolean delete(long id) {
73     repo.deleteById(id);
74     return !repo.existsById(id);
75 }
76
77 }
```

## 8. Rest API – Repo

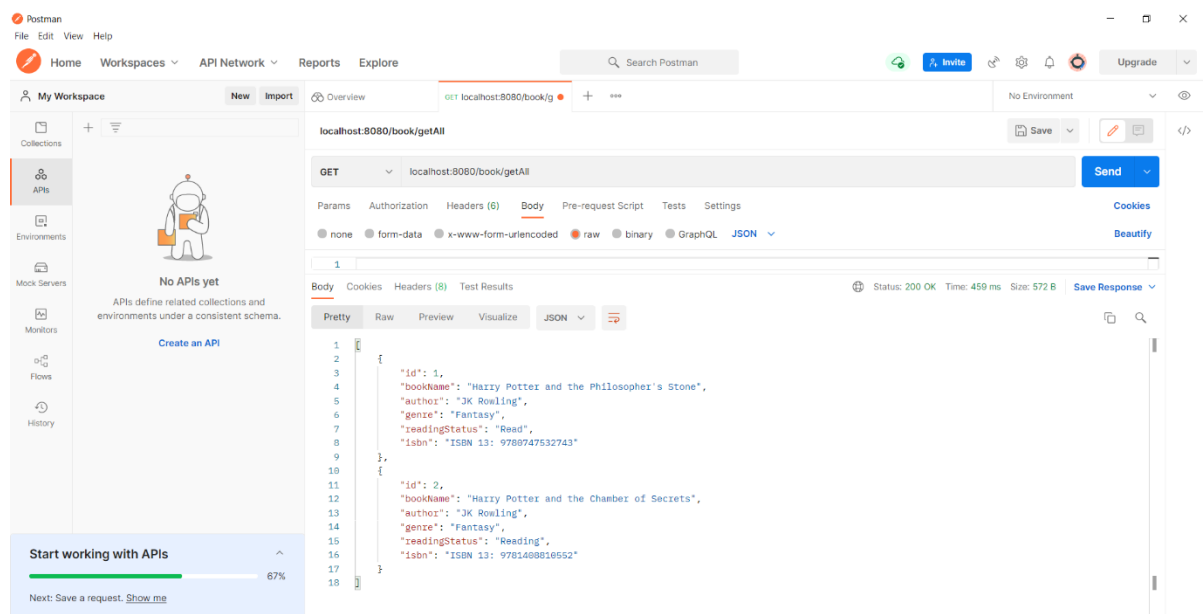
### 8.1 Customer Queries



```
1 package com.qa.readingList.repo;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5 import com.qa.readingList.domain.Book;
6
7 @Repository
8 public interface BookRepo extends JpaRepository<Book, Long> {
9
10     // Custom queries
11     Book findByBookName(String bookName);
12     Book findByAuthor(String author);
13     Book findByGenre(String genre);
14     Book findByReadingStatus(String readingStatus);
15     Book findByIsbn(String isbn);
16 }
17
18
```

## 9. Postman

Checked all queries using Postman to ensure they worked.



## 10. Integration Test

The following tests were created, were ran as a Junit test, failed initially due to error in casing of a word not matching with that in testdata.sql file. This error was amended and the test passed.

### 10.1 Test for Create

```
gList/controller/BookControllerIntegrationTest.java - Eclipse IDE
Run Window Help
BookController.java Book.java BookControllerUnitTest.java UserControllerUnitTest.java BookControllerIntegrationTest.java X
1 package com.qa.readingList.controller;
2
3 import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;
4
5
6 @SpringBootTest
7 @AutoConfigureMockMvc
8 @Sql(scripts = {"classpath:testschema.sql", "classpath:testdata.sql"}, executionPhase = ExecutionPhase.BEFORE_TEST_METHOD)
9 @ActiveProfiles("test")
10 public class BookControllerIntegrationTest {
11
12     @Autowired
13     private MockMvc mvc;
14
15     @Autowired
16     private ObjectMapper mapper;
17
18     // Create
19     @Test
20     public void createTest() throws Exception {
21         Book entry = new Book(1L, "The Book Thief", "Markus Zusak", "Historical Fiction", "To Be Read", "ISBN: 9781784162122");
22         String entryAsJSON = mapper.writeValueAsString(entry);
23
24         Book result = new Book(2L, "The Book Thief", "Markus Zusak", "Historical Fiction", "To Be Read", "ISBN: 9781784162122");
25         String resultAsJSON = mapper.writeValueAsString(result);
26
27         mvc.perform(post("/book/create")
28             .contentType(MediaType.APPLICATION_JSON)
29             .content(entryAsJSON))
30             .andExpect(status().isCreated())
31             .andExpect(content().json(resultAsJSON));
32     }
33 }
```

### 10.2 Test for getAll

```
54 // getAll
55 @Test
56 public void getAllTest() throws Exception {
57     Book book = new Book(1L, "Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323");
58     List<Book> output = new ArrayList<>();
59     output.add(book);
60     String outputAsJSON = mapper.writeValueAsString(output);
61
62     mvc.perform(get("/book/getAll")
63         .contentType(MediaType.APPLICATION_JSON))
64         .andExpect(status().isOk())
65         .andExpect(content().json(outputAsJSON));
66 }
67 }
```

### 10.3 Test for getByID

```
67 // getByID
68 @Test
69 public void getByIdTest() throws Exception {
70     Book entry = new Book(1L, "Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323");
71     String entryAsJSON = this.mapper.writeValueAsString(entry);
72
73     mvc.perform(get("/book/getById/1")
74         .contentType(MediaType.APPLICATION_JSON))
75         .andExpect(status().isOk())
76         .andExpect(content().json(entryAsJSON));
77 }
78 }
79 }
```

### 10.4 Test for Update

```
80 // Update
81 @Test
82 public void updateTest() throws Exception {
83     Book entry = new Book("The Book Thief", "Markus Zusak", "Historical Fiction", "To Be Read", "ISBN: 9781784162122");
84     Book result = new Book(1L, "The Book Thief", "Markus Zusak", "Historical Fiction", "To Be Read", "ISBN: 9781784162122");
85     String entryAsJSON = this.mapper.writeValueAsString(entry);
86     String resultAsJSON = this.mapper.writeValueAsString(result);
87
88     mvc.perform(put("/book/update/1")
89         .contentType(MediaType.APPLICATION_JSON)
90         .content(entryAsJSON))
91         .andExpect(status().isAccepted())
92         .andExpect(content().json(resultAsJSON));
93 }
94 }
```

### 10.5 Test for Delete

```
94 // Delete
95 @Test
96 public void deleteTest() throws Exception {
97     mvc.perform(delete("/book/delete/1")
98         .contentType(MediaType.APPLICATION_JSON))
99         .andExpect(status().isNoContent());
100 }
101 }
102 }
103 }
```

## 11. Unit Test – Controller

### 11.1 Test for getAll

```
37 // getAll
38 @Test
39 public void getAllTest() throws Exception {
40     Book book = new Book(1L, "Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323");
41     List<Book> output = new ArrayList<>();
42     output.add(book);
43     String outputAsString = mapper.writeValueAsString(output);
44
45     Mockito.when(this.service.getAll()).thenReturn(output);
46
47     mvc.perform(get("/book/getAll")
48         .contentType(MediaType.APPLICATION_JSON))
49         .andExpect(status().isOk())
50         .andExpect(content().json(outputAsString));
51 }
52
```

### 11.2 Test for getById

```
53 // getById
54 @Test
55 public void getByIdTest() throws Exception {
56     Book entry = new Book("The Book Thief", "Markus Zusak", "Historical Fiction", "To Be Read", "ISBN: 9781784162122");
57     String entryAsString = this.mapper.writeValueAsString(entry);
58
59     Mockito.when(this.service.getById(1L)).thenReturn(entry);
60
61     mvc.perform(get("/book/getById/1")
62         .contentType(MediaType.APPLICATION_JSON))
63         .andExpect(status().isOk())
64         .andExpect(content().json(entryAsString));
65 }
66
```

### 11.3 Test for Create

```
67 // Create
68 @Test
69 void createTest() throws Exception {
70     Book entry = new Book("The Book Thief", "Markus Zusak", "Historical Fiction", "To Be Read", "ISBN: 9781784162122");
71     String entryAsString = this.mapper.writeValueAsString(entry);
72
73     Mockito.when(this.service.create(entry)).thenReturn(entry);
74
75     mvc.perform(post("/book/create")
76         .contentType(MediaType.APPLICATION_JSON)
77         .content(entryAsString))
78         .andExpect(status().isCreated())
79         .andExpect(content().json(entryAsString));
80 }
81
```

### 11.4 Test for Update

```
82 // Update
83 @Test
84 public void updateTest() throws Exception {
85     Book entry = new Book("The Book Thief", "Markus Zusak", "Historical Fiction", "To Be Read", "ISBN: 9781784162122");
86     String entryAsString = this.mapper.writeValueAsString(entry);
87
88     Mockito.when(this.service.update(1L, entry)).thenReturn(entry);
89
90     mvc.perform(put("/book/update/1")
91         .contentType(MediaType.APPLICATION_JSON)
92         .content(entryAsString))
93         .andExpect(status().isAccepted())
94         .andExpect(content().json(entryAsString));
95 }
96
```

### 11.5 Test for Delete

```
97 // Delete
98 @Test
99 public void deleteTest() throws Exception {
100     Mockito.when(this.service.delete(1L)).thenReturn(true);
101
102     mvc.perform(delete("/book/delete/1")
103         .contentType(MediaType.APPLICATION_JSON))
104         .andExpect(status().isNoContent());
105 }
106
107
108 @Test
109 public void deleteFailTest() throws Exception {
110     Mockito.when(this.service.delete(2L)).thenReturn(false);
111
112     mvc.perform(delete("/book/delete/5")
113         .contentType(MediaType.APPLICATION_JSON))
114         .andExpect(status().isInternalServerError());
115 }
116 }
```

## 12. Unit Test – Service

### 12.1 Test for getAll

```
adingList/service/BookServiceUnitTest.java - Eclipse IDE
Run Window Help
BookController.java Book.java BookControllerUnitTest.java BookControllerIntegrationTest.java BookServiceUnitTest.java X
25 @Test
26 public void getAllTest() {
27     List<Book> output = new ArrayList<>();
28     output.add(new Book("Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323"));
29
30     Mockito.when(this.repo.findAll()).thenReturn(output);
31
32     assertEquals(output, this.service.getAll());
33
34     Mockito.verify(this.repo, Mockito.times(1)).findAll();
35 }
36
```

### 12.2 Test for getById

```
37 @Test
38 public void getByIdTest() {
39     Optional<Book> OptionalOutput = Optional.of(new Book(1L, "Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323"));
40     Book output = new Book(1L, "Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323");
41
42     Mockito.when(this.repo.findById(1L)).thenReturn(OptionalOutput);
43
44     assertEquals(output, this.service.getById(1L));
45
46     Mockito.verify(this.repo, Mockito.times(1)).findById(1L);
47 }
48
```

### 12.3 Test for Create

```
49 @Test
50 public void createTest() {
51     Book input = new Book("Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323");
52     Book output = new Book(1L, "Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323");
53
54     Mockito.when(this.repo.saveAndFlush(input)).thenReturn(output);
55
56     assertEquals(output, this.service.create(input));
57
58     Mockito.verify(this.repo, Mockito.times(1)).saveAndFlush(input);
59 }
60
```

### 12.4 Test for Update

```
61 @Test
62 public void updateTest() {
63     Book input = new Book("Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323");
64     Optional<Book> existing = Optional.of(new Book(1L, "Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323"));
65     Book output = new Book(1L, "Coding for Dummies", "Nikhil Abraham", "Educational", "To Be Read", "ISBN: 9781119293323");
66
67     Mockito.when(this.repo.findById(1L)).thenReturn(existing);
68     Mockito.when(this.repo.saveAndFlush(output)).thenReturn(output);
69
70     assertEquals(output, this.service.update(1L, input));
71
72     Mockito.verify(this.repo, Mockito.times(1)).findById(1L);
73     Mockito.verify(this.repo, Mockito.times(1)).saveAndFlush(output);
74 }
75
```

### 12.5 Test for Delete

```
76 @Test
77 public void deleteTest() {
78     Mockito.when(this.repo.existsById(1L)).thenReturn(false);
79
80     assertTrue(this.service.delete(1L));
81
82     Mockito.verify(this.repo, Mockito.times(1)).deleteById(1L);
83     Mockito.verify(this.repo, Mockito.times(1)).existsById(1L);
84 }
85
```

## 13. Test Coverage

### 13.1 Integration Test

The integration test coverage for the project is at 47.6%, below the industry standard of 80%.

The screenshot shows the Eclipse IDE with the `BookControllerIntegrationTest.java` file open. The test is run using JUnit 5, and the coverage report is displayed in the Console window. The coverage is 47.6%.

Element	Coverage	Vered Instructions	Issed Instructions	Total Instructions
ReadingListProject	47.6 %	400	440	840

### 13.2 Unit Test – Controller

The unit test coverage for the controller is at 50%, below the industry standard of 80%.

The screenshot shows the Eclipse IDE with the `BookControllerUnitTest.java` file open. The test is run using JUnit 5, and the coverage report is displayed in the Console window. The coverage is 50.0%.

Element	Coverage	Vered Instructions	Issed Instructions	Total Instructions
ReadingListProject	50.0 %	420	420	840

### 13.3 Unit Test – Service

The unit test coverage for the service is at 41.7%, below the industry standard of 80%.

The screenshot shows the Eclipse IDE with the `BookServiceUnitTest.java` file open. The test is run using JUnit 5, and the coverage report is displayed in the Console window. The coverage is 41.7%.

Element	Coverage	Vered Instructions	Issed Instructions	Total Instructions
ReadingListProject	41.7 %	451	630	1,081



## 14. Front-End

### 14.1 Planning

#### Front End Planning

28 April 2022 22:41

