



BOOKS AND REVIEWS

Saad Ahmed
22AprEnable_1

Introduction

- Create an OOP-based application with *CRUD* functionality.
- A relational database with 2 entities:
 - Books and Reviews
- User can interact through a locally hosted site.

Tools

- Tools:
 - Back-end:
 - Java(Spring Boot)(Maven)
 - Junit and Mockito for testing
 - Selenium for automated front-end tests
 - SonarQube
 - MySQL database
 - Front-end:
 - Html, CSS and JavaScript
 - Bootstrap
 - Jira
 - GitHub

Risk

Risk	Statement	Response	Objective	Likelihood	Impact	Risk Level
Repetitive strain injury (RSI)	Muscles can begin to ache	Ensure my posture is healthy and I move my muscles and stretch every so often.	To prevent pain/stiffness from RSI	Very Unlikely	Minor	Low
Github servers being down	I would not be able to upload my remote work.	Ensure that I regularly check github status and that I regularly push.	Ensure github repository is up to date with remote repository.	Very Unlikely	Minor	Low Medium
Losing my work	This will negatively affect the project, losing a lot of time to redo the work.	Ensure that I regularly save my work on my machine as well as push updates to github.	To make sure that my work is saved to multiple places	Very Unlikely	Catastrophic	Medium High
No internet	This would mean an inability to contact my trainer and push updates to github.	Ensure I have a reliable internet provider, I can also use mobile hotspot if it is down.	To make sure I am able to connect to the internet so that I can push to github and contact my trainer if needed.	Very Unlikely	Major	Low Medium
Trainer unavailable	Can be occupied helping others or away due to illness	Move on to something else whilst I am waiting, I can also check QA community resources, stack overflow and other places that may help me.	To ensure I have something to do whilst I wait for help or have the ability to find the answer myself.	Likely	Minor	Low Medium
Injury	An injury would lead me to being unable to work on the project.	Ensure I am engaging in safe activities and exercise responsibly.	To prevent an injury occurring by being more cautious and aware when I exercise.	Moderate	Major	Medium
Power outage	This could result in loss of work as well as time.	Ensure my laptop is charged, so that I can carry on working whilst the power is gone.	To reduce the amount of time that will be lost from a power outage	Very Unlikely	Hazardous	Medium
Fire	This could result in a loss of work/ equipment and injury	Ensure smoke detectors are working, a fire extinguisher and blanket are present. As well as a clear exit path.	To minimise damage to equipment and reduce the risk of injury.	Very Unlikely	Catastrophic	Medium High

Risk Matrices

Risk Matrices					
	Negligible	Minor	Major	Hazardous	Catastrophic
Very Unlikely	Low	Low	Low medium	Medium	Medium
Unlikely	Low	Low Medium	Low medium	Medium	Medium High
Moderate	Low	Low Medium	Medium	Medium High	Medium High
Likely	Low	Low medium	Medium	Medium high	High
Very Likely	Low medium	Medium	Medium high	High	High

MoSCoW

Must have:

- A working front end application that utilizes back-end API
- The ability to create/read/update/delete books in the system.
- The ability to create/read/update/delete reviews in the system.
- Data must be visible to the user in the front-end application.

Should have:

- books should have an id, title, description and author
- Reviews should have an id, first name, surname, rating, review and a book id
- Should have at least 80% test coverage
- Each book show the reviews associated with it

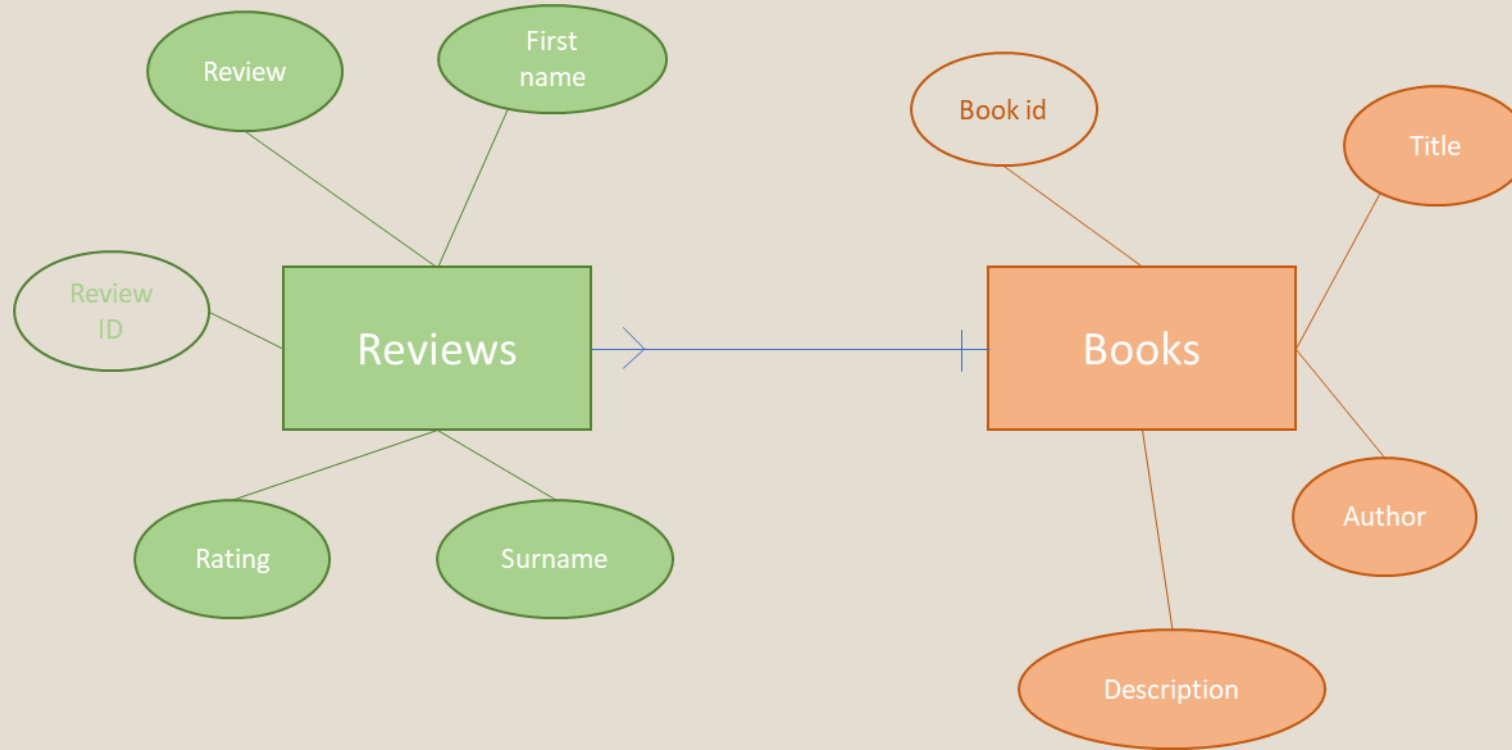
Could have:

- A login page for reviewers
- An average rating for books
- Could store reviewers emails/phone numbers

Won't have:

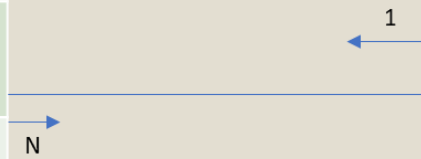
- Ads
- Sensitive reviewer information

Entity Relationship Diagram



UML

Reviews		
PK	review_id	int
FK	Book_id	int
	first_name	varchar(255)
	surname	Varchar(255)
	rating	int
	review	Varchar(255)



Books		
PK	book_id	int
	Title	varchar(255)
	Description	varchar(255)
	author	varchar(255)

Sprint(s)

Edit sprint: BOOK Sprint 1

Sprint name *

Start date *



End date *



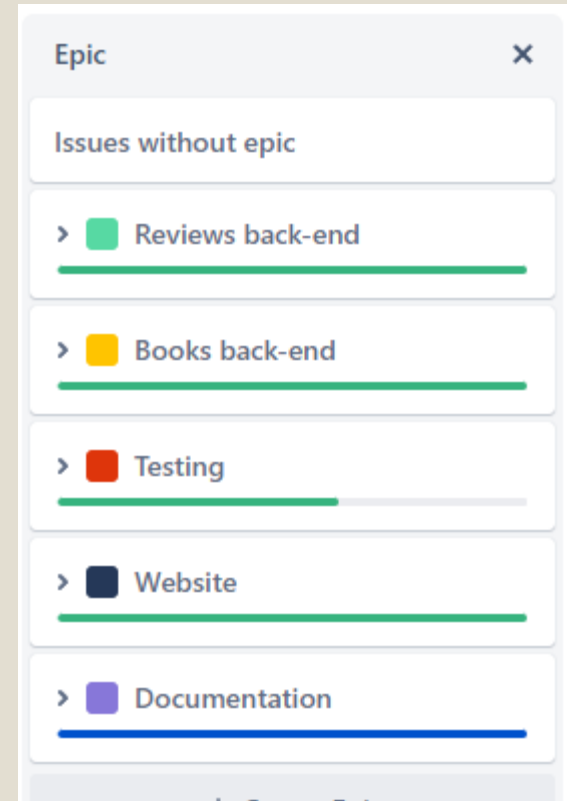
Sprint goal

Update

Cancel

Epics

- *Created 5 epics*
- *User stories*
- *Tasks*



User Stories and Tasks

BOOK-20	create book domain class	BOOKS BACK-END	1	DONE	
BOOK-59	use selenium to test website	TESTING	5	DONE	
BOOK-31	Manually test with postman	TESTING	3	DONE	
BOOK-21	create book dto class	BOOKS BACK-END	1	DONE	
BOOK-36	As a user i want to be able to select a book to write a review on, so that i can share my experience	REVIEWS BACK-END	3	DONE	
BOOK-35	As a user i want to be able to see the reviews on a certain book, so that i can see what people think of it.	BOOKS BACK-END	3	DONE	
BOOK-55	create tables to display data	WEBSITE	2	DONE	
BOOK-52	use javascript to connect to API	WEBSITE	3	DONE	
BOOK-51	use javascript to make functions for the buttons	WEBSITE	3	DONE	
BOOK-50	using html and css, make a page with buttons and input boxes	WEBSITE	3	DONE	
BOOK-23	create book exception class	BOOKS BACK-END	1	DONE	
BOOK-22	create book service class	BOOKS BACK-END	3	DONE	
BOOK-13	create review exception class	REVIEWS BACK-END	1	DONE	
BOOK-19	As a user, I would like to be able to delete books, so that I can remove books that are no longer available.	BOOKS BACK-END	2	DONE	
BOOK-0	As a user, I would like to be able to update reviews, so that any information can be changed.	REVIEWS BACK-END	2	DONE	
BOOK-16	As a user, I want to be able to add a books to the database, so that new books can be created	BOOKS BACK-END	2	DONE	
BOOK-10	create review domain class	REVIEWS BACK-END	1	DONE	

using html and css, make a page with buttons and input boxes



Done

✓ Done

Description

Add a description...

Child issues

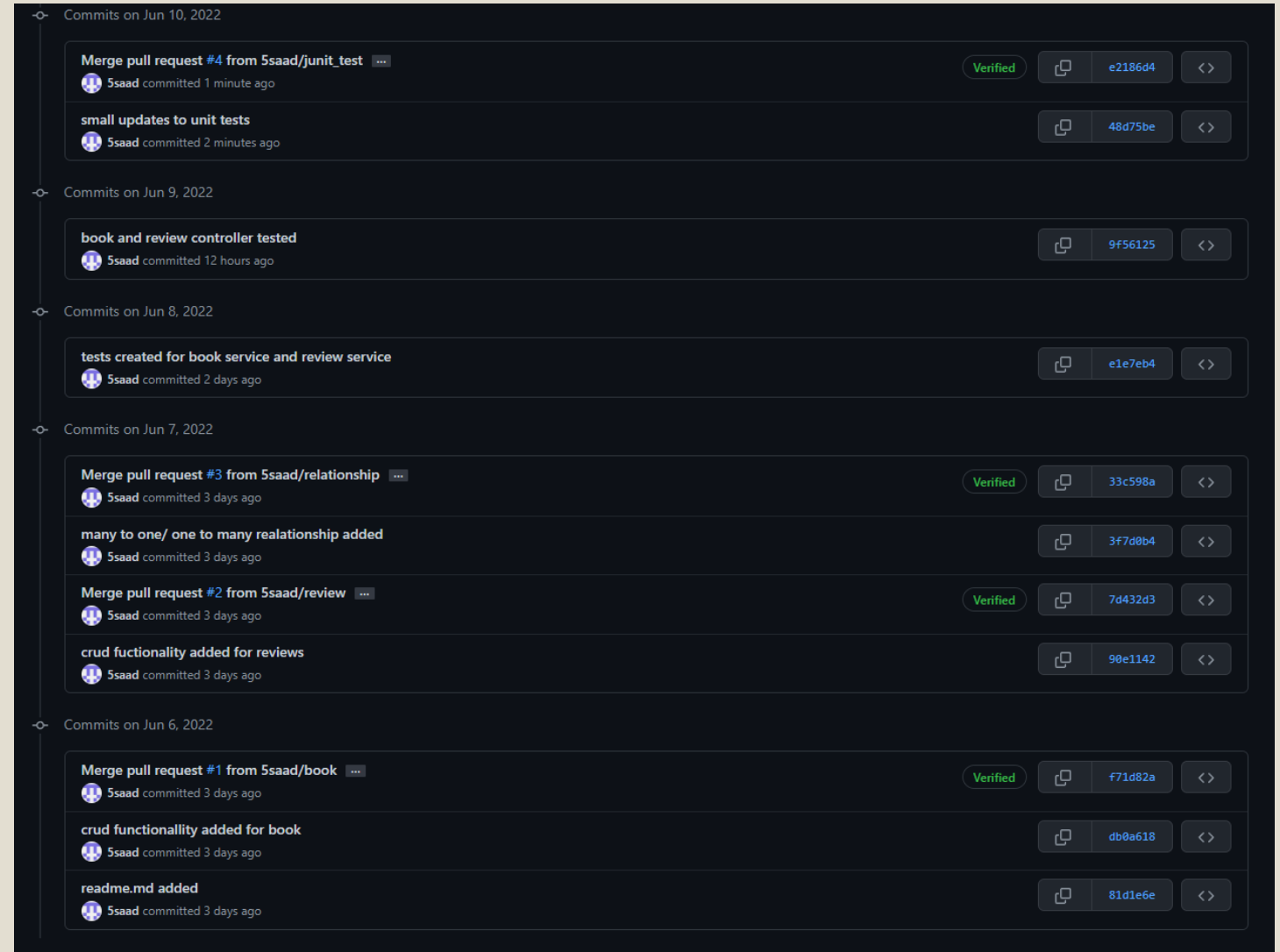
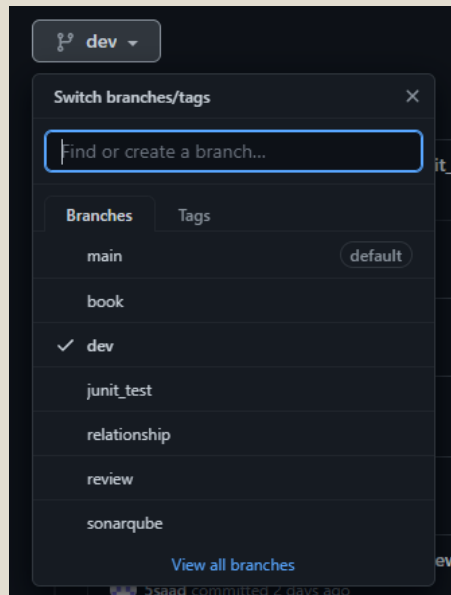
Order by ... +

0% Done

- BOOK-53 make inputs for book - TO DO
- BOOK-54 make inputs for revi... - TO DO
- BOOK-58 use bootstrap - TO DO

GitHub

- Dev branch was created
- Review Branch
- Book branch
- Relationship Branch
- Junit test branch
- Sonarqube branch



Version Control

- *Each epic = new feature*
- *Dev branch created > feature branch created*
- *After each feature is done:*
 - *Git add . > git commit -m "..."* > *git push*
 - *Merge with dev branch in GitHub*
 - *Git pull dev branch to local*
 - *Ready for new feature!*

H2-console

- *Used to locally host SQL database*
- *Manual testing*

The screenshot displays the H2 Console web interface. The top toolbar includes icons for help, connection, and settings, along with a status bar showing 'Auto commit' is checked, 'Max rows' is set to 1000, and 'Auto complete' is set to 'Off'. The left sidebar shows the database structure for 'jdbc:h2:mem:testdb', including tables 'BOOK' and 'REVIEW', a schema 'INFORMATION_SCHEMA', and users. The main area contains a 'SQL statement' input field and buttons for 'Run', 'Run Selected', 'Auto complete', and 'Clear'. Below this is a section titled 'Important Commands' with a table of shortcuts and their functions.

Icon	Shortcut	Action
?		Displays this Help Page
📜		Shows the Command History
▶	Ctrl+Enter	Executes the current SQL statement
▶	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
🔌		Disconnects from the database

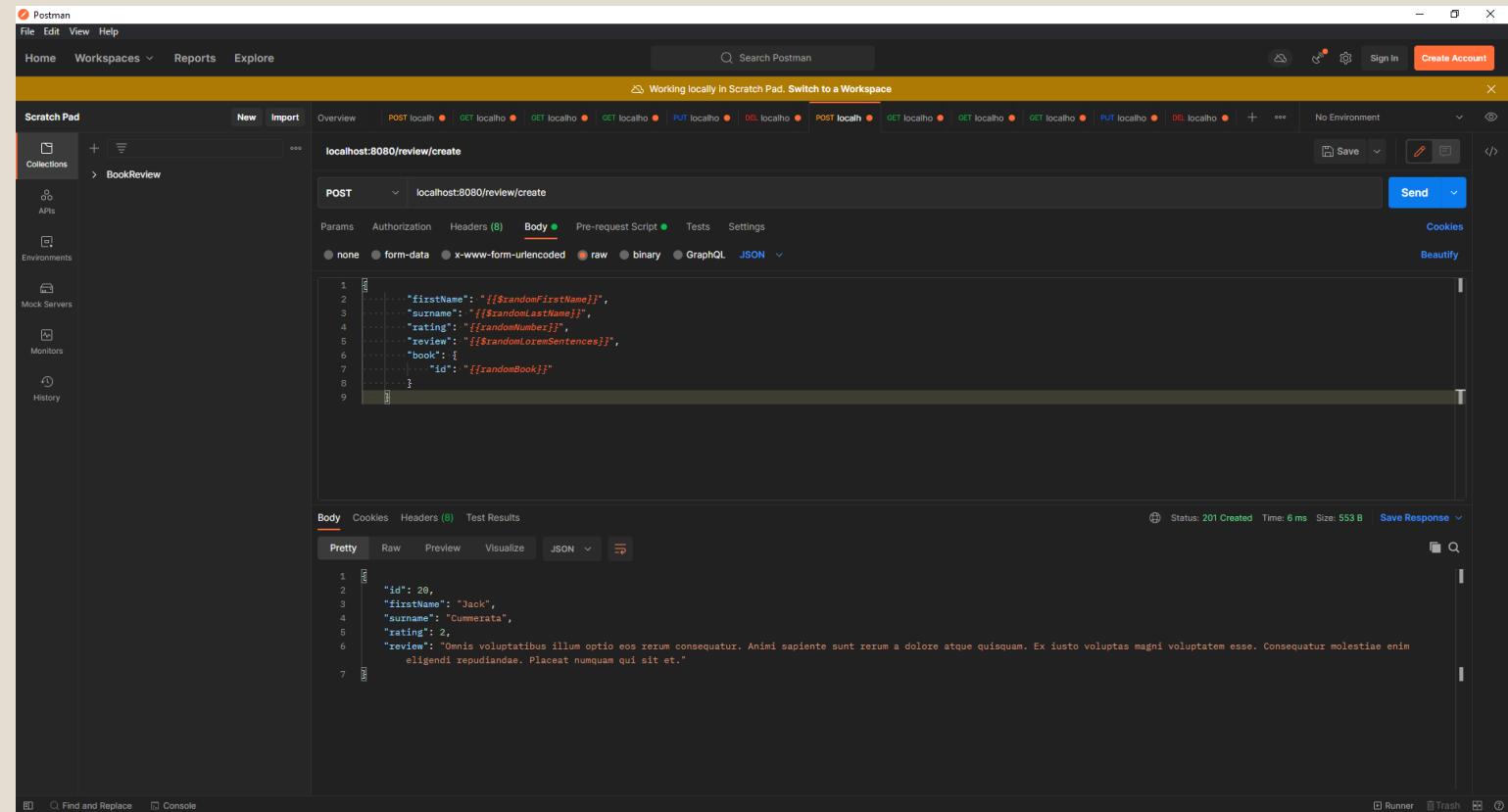
Below the commands table is a section titled 'Sample SQL Script' with a table of sample queries and their descriptions.

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

At the bottom, there is a section titled 'Adding Database Drivers' with the text: 'Additional database drivers can be registered by adding the Jar file location of the driver to the environment variables H2'.

Postman

- *Manual testing*
- *Random data*



Mockito testing

- Mockito used to test service Mocked the repo in the service class.
- CRUD functions were successfully tested.

```
@Test
public void updateTest() {
    final Optional<Review> book = Optional.of(new Review(1L, "jeff", "smith", 3, "great book"));
    final Review updated = new Review(1L, "changed", "changed", 1, "changed");
    final ReviewDto updatedReviewDto = new ReviewDto(1L, "changed", "changed", 1, "changed");
    final long id = 1L;

    Mockito.when(this.repo.findById(id)).thenReturn(book);
    Mockito.when(this.repo.saveAndFlush(updated)).thenReturn(updated);

    assertThat(this.service.update(id, updated)).isEqualTo(updatedReviewDto);

    Mockito.verify(this.repo, Mockito.times(1)).findById(id);
    Mockito.verify(this.repo, Mockito.times(1)).saveAndFlush(updated);
}

@Test
public void deleteTest() {
    final long id = 1L;
    Optional<Review> foundReview = Optional.of(new Review(1L, "jeff", "smith", 3, "great book"));
    Mockito.when(this.repo.findById(id)).thenReturn(foundReview);
    Mockito.when(this.repo.existsById(id)).thenReturn(false);

    assertThat(this.service.delete(id)).isTrue();

    Mockito.verify(this.repo, Mockito.times(1)).findById(id);
    Mockito.verify(this.repo, Mockito.times(1)).existsById(id);
}

@Test
public void findByRatingTest() {
    List<Review> expectedReviews = List.of(new Review(2L, "sam", "thomas", 2, "meh book"));
    List<ReviewDto> expectedReviewsDto = List.of(new ReviewDto(2L, "sam", "thomas", 2, "meh book"));
    final int rating = 2;

    Mockito.when(this.repo.findByRating(rating)).thenReturn(expectedReviews);
    assertThat(this.service.findByRating(rating)).isEqualTo(expectedReviewsDto);

    Mockito.verify(this.repo, Mockito.times(1)).findByRating(rating);
}
```

```
@SpringBootTest
public class BookServiceTest {

    @Autowired
    private BookService service;

    @MockBean
    private BookRepo repo;

    @Test
    public void readAllTest() {
        List<BookDto> expectedBooksDto = List.of(new BookDto(1L, "cool title", "random description", "an author"),
            new BookDto(2L, "cool title II", "random description", "an author"));
        List<Book> expectedBooks = List.of(new Book(1L, "cool title", "random description", "an author"),
            new Book(2L, "cool title II", "random description", "an author"));

        Mockito.when(this.repo.findAll()).thenReturn(expectedBooks);
        assertThat(this.service.readAll()).isEqualTo(expectedBooksDto);

        Mockito.verify(this.repo, Mockito.times(1)).findAll();
    }

    @Test
    public void readByIdTest() {
        final Optional<Book> book = Optional.of(new Book(1L, "title", "great book", "John Doe"));
        final BookDto bookDto = new BookDto(1L, "title", "great book", "John Doe");
        final long id = 1L;

        Mockito.when(this.repo.findById(id)).thenReturn(book);
        assertThat(this.service.readId(id)).isEqualTo(bookDto);
    }

    @Test
    public void createTest() {
        final Book created = new Book(1L, "title", "great book", "John Doe");
        final BookDto createdDto = new BookDto(1L, "title", "great book", "John Doe");

        Mockito.when(this.repo.save(created)).thenReturn(created);
        assertThat(this.service.create(created)).isEqualTo(createdDto);

        Mockito.verify(this.repo, Mockito.times(1)).save(created);
    }

    @Test
    public void updateTest() {
        final Optional<Book> book = Optional.of(new Book(1L, "title", "great book", "John Doe"));
    }
```

Runs: 6/6 ✖ Errors: 0

BookServiceTest [Runner: JUnit 5] (0.199 s)

- ✓ readAllTest() (0.154 s)
- ✓ updateTest() (0.014 s)
- ✓ readByIdTest() (0.007 s)
- ✓ createTest() (0.006 s)
- ✓ deleteTest() (0.007 s)
- ✓ findByAuthorTest() (0.005 s)

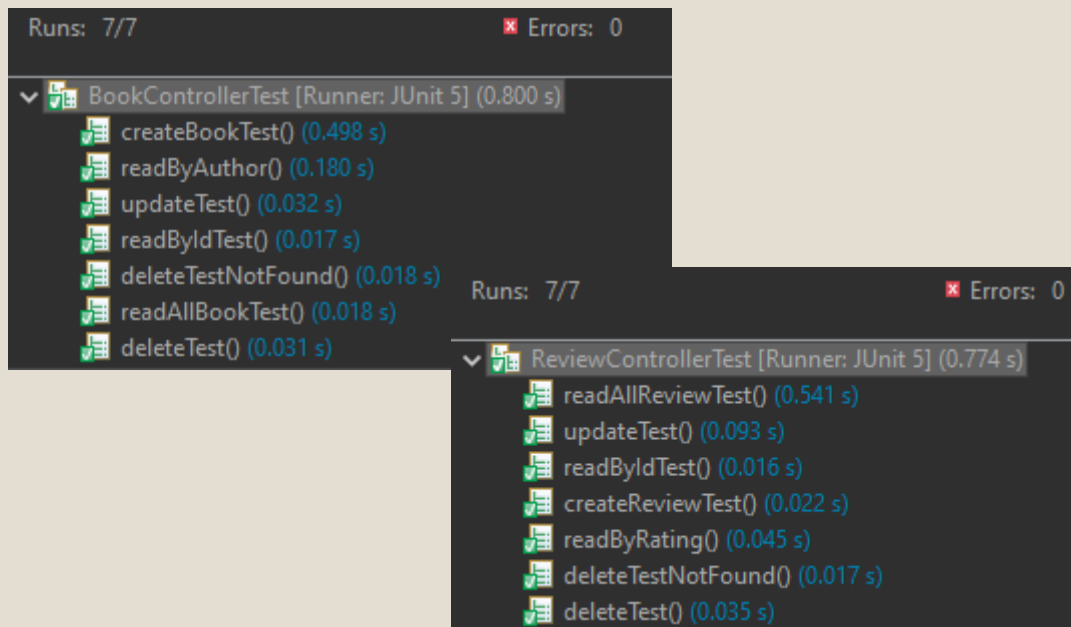
Runs: 6/6 ✖ Errors: 0

ReviewServiceTest [Runner: JUnit 5] (0.218 s)

- ✓ readAllTest() (0.175 s)
- ✓ updateTest() (0.012 s)
- ✓ readByIdTest() (0.006 s)
- ✓ findByRatingTest() (0.006 s)
- ✓ createTest() (0.006 s)
- ✓ deleteTest() (0.006 s)

MockMvc testing

- MockMvc used to test the controller classes.
- Correct HTTP response
- Correct content



The image shows two screenshots of JUnit test results. The top screenshot shows the results for `BookControllerTest` with 7/7 tests passing and 0 errors. The tests listed are `createBookTest()`, `readByAuthor()`, `updateTest()`, `readByIdTest()`, `deleteTestNotFound()`, `readAllBookTest()`, and `deleteTest()`. The bottom screenshot shows the results for `ReviewControllerTest` with 7/7 tests passing and 0 errors. The tests listed are `readAllReviewTest()`, `updateTest()`, `readByIdTest()`, `createReviewTest()`, `readByRating()`, `deleteTestNotFound()`, and `deleteTest()`.

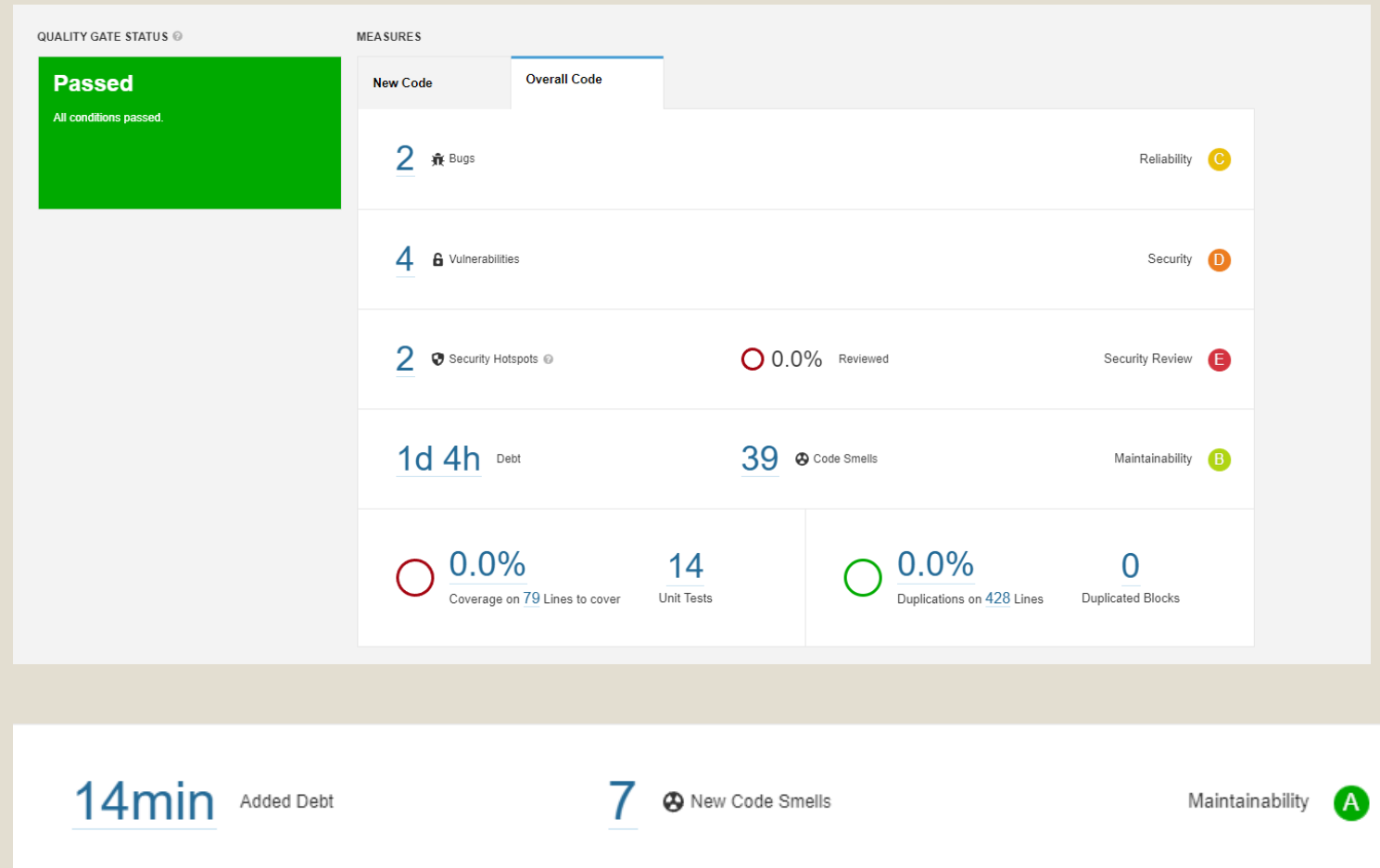
```
Runs: 7/7 Errors: 0
BookControllerTest [Runner: JUnit 5] (0.800 s)
  ✓ createBookTest() (0.498 s)
  ✓ readByAuthor() (0.180 s)
  ✓ updateTest() (0.032 s)
  ✓ readByIdTest() (0.017 s)
  ✓ deleteTestNotFound() (0.018 s)
  ✓ readAllBookTest() (0.018 s)
  ✓ deleteTest() (0.031 s)

Runs: 7/7 Errors: 0
ReviewControllerTest [Runner: JUnit 5] (0.774 s)
  ✓ readAllReviewTest() (0.541 s)
  ✓ updateTest() (0.093 s)
  ✓ readByIdTest() (0.016 s)
  ✓ createReviewTest() (0.022 s)
  ✓ readByRating() (0.045 s)
  ✓ deleteTestNotFound() (0.017 s)
  ✓ deleteTest() (0.035 s)
```

```
29 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
30 @ActiveProfiles("dev")
31 @Sql(scripts = { "classpath:testschema.sql",
32                 "classpath:testdata.sql" }, executionPhase = ExecutionPhase.BEFORE_TEST_METHOD)
33 @AutoConfigureMockMvc
34 public class BookControllerTest {
35
36     @Autowired
37     private MockMvc mockMvc;
38
39     @Autowired
40     private ObjectMapper mapper;
41
42     private final Long testId = 1L;
43     private final String testAuthor = "prof. jeff";
44     private final Book testBook = new Book(1L, "the title", "nice book", "prof. jeff");
45
46     private final List<Review> expectedReviews = List.of(new Review(1L, "john", "doe", 4, "good"));
47     private final Book testBookRev = new Book(1L, "the title", "nice book", "prof. jeff", expectedReviews);
48     private final ArrayList<Book> bookArray = new ArrayList<Book>();
49
50     @Test
51     void createBookTest() throws Exception {
52
53         RequestBuilder request = post("/book/create").contentType(MediaType.APPLICATION_JSON)
54             .content(this.mapper.writeValueAsString(testBook));
55         ResultMatcher responseStatus = status().isCreated();
56         ResultMatcher responseContent = content().json(this.mapper.writeValueAsString(testBook));
57         this.mockMvc.perform(request).andExpect(responseStatus).andExpect(responseContent);
58     }
59
60     @Test
61     void readAllBookTest() throws Exception {
62         bookArray.add(testBookRev);
63
64         RequestBuilder request = get("/book/readAll").contentType(MediaType.APPLICATION_JSON)
65             .accept(MediaType.APPLICATION_JSON);
66         ResultMatcher responseStatus = status().isOk();
67         ResultMatcher responseContent = content().json(this.mapper.writeValueAsString(bookArray));
68         this.mockMvc.perform(request).andExpect(responseStatus).andExpect(responseContent);
69     }
70
71     @Test
72     void readByIdTest() throws Exception {
73         RequestBuilder request = get("/book/read/" + testId).contentType(MediaType.APPLICATION_JSON)
74             .accept(MediaType.APPLICATION_JSON);
75         ResultMatcher responseStatus = status().isOk();
76         ResultMatcher responseContent = content().json(this.mapper.writeValueAsString(testBookRev));
77         this.mockMvc.perform(request).andExpect(responseStatus).andExpect(responseContent);
78     }
79
80     @Test
```

SonarQube

- Refactoring code
- Reduce Smells
- Reduce Bugs
- Reduce Vulnerabilities



Front-end

- Design:
 - HTML
 - Bootstrap
- JavaScript:
 - Fetch API
 - Retrieve inputs
 - Functions for buttons

The screenshot shows the top section of the 'Book Reviews' application. It features two main input sections: 'Books' and 'Reviews'. The 'Books' section has input fields for ID, Title, Description, and Author, followed by buttons for 'Create Book', 'Find By ID', 'Find By Author', 'Read All Books', 'Update Book', and 'Delete Book'. The 'Reviews' section has input fields for ID, First name, Surname, Rating (a dropdown menu), Review, and Book ID, followed by buttons for 'Create Review', 'Find By ID', 'Find By Rating', 'Read All Reviews', 'Update Review', and 'Delete Review'.

The screenshot shows the bottom section of the 'Book Reviews' application, displaying two tables. The 'Books' table has columns for ID, Title, Description, Author, and Review. The 'Reviews' table has columns for ID, First Name, Surname, Rating, and Review.

ID	Title	Description	Author	Review
1	Chair Steel Salad connecting Cambridgeshire	You can't transmit the card without calculating the wireless PCI feed!	Samuel Jacobson	0 Reviews
2	blue platforms	You can't navigate the interface without bypassing the redundant SSL panel!	Simon Nicolas	<ul style="list-style-type: none">Review Id: 8 Rating: 1 Review: Rem laborum unde eos omnis enim. Laborum id dolores. Necessitatibus et quisquam eos at. Nam ut quibusdam voluptatem necessitatibus. Itaque aliquam quo iste. Consequatur ut repellendus delectus accusantium facilis.Review Id: 11 Rating: 2 Review: Sit praesentium perspiciatis alias omnis delentit. Ut nisi sit voluptas error. Molestias quia fugiat nam molestiae quo nesciunt est in.Review Id: 14 Rating: 5 Review: Quae ut rerum vel. Commodi ex mollitia et consequatur.Review Id: 16

ID	First Name	Surname	Rating	Review
1	Durward	Zemlak	4	Natus ullam numquam veniam vel praesentium voluptas iste ipsum. Vel molestiae nam quos non aut consequuntur nobis. Quos aliquid voluptas ducimus et. Sed dolores aut. Ipsum iusto fugit qui accusamus voluptas. Nulla quasi atque rerum ipsam.
2	Gaylord	Schneider	1	Est quibusdam explicabo iure officis. Voluptatem in maiores. Magni id odio consequuntur. Et ipsum totam similique rerum. Provident beatae eveniet at quibusdam ut.
3	Maximilla	Witting	4	Sit eligendi quas ut fugit cumque alias. Molestias dolore consequatur et ipsum qui error velit. Cumque eaque illum suscipit magnam debitis commodi. Ut vitae laborum modi.
4	Keyon	Tromp	5	Ipsam adipisci quia repudiandae consequatur ipsum culpa fugiat itaque eligendi. Accusantium perspiciatis ea ipsum. Omnis aperiam blanditibus hic possimus neque quis cumque molestiae at. Molestias saepe optio earum delectus. Minus quo quis.
5	Theodore	Legros	3	Occaecati hic fugit qui est qui cum. Molestiae sint voluptas quos a aspernatur omnis. Harum excepturi nesciunt. Sapiente qui id recusandae quaerat. Consequuntur molestiae ratione porro nam debitis beatae velit vero.
6	Piper	Becker	4	Quo rerum molestiae. Illum fuga et ad illum. Autem atque non rerum enim. Repudiandae alias maiores laudantium. Hic blanditibus recusandae et voluptatum vel voluptatem ad.

Selenium

- *Used for front-end automated testing*
- *Successfully tested all of the functions on the website*

Books

ID Title Description Author

Create Book

Find By ID

Find By Author

Read All Books

Update Book

Delete Book

Reviews

ID First name Surname Rating Review Book ID

Create Review

Find By ID

Find By Rating

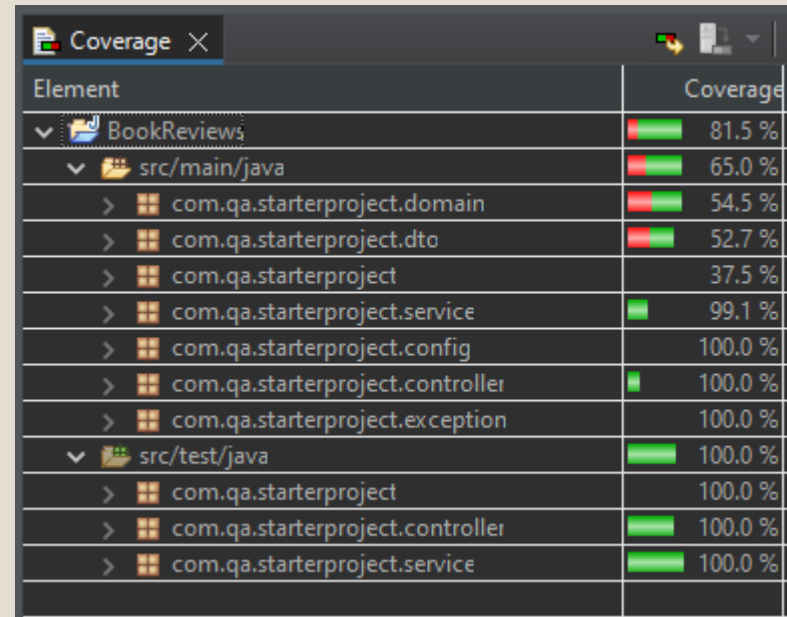
Read All Reviews

Update Review

Delete Review

Test Coverage

- 81.5% Test coverage!



The screenshot shows a 'Coverage' window from an IDE. It displays a tree view of the project structure with corresponding coverage percentages. The 'BookReviews' project has an overall coverage of 81.5%. The 'src/main/java' directory has 65.0% coverage, while the 'src/test/java' directory has 100.0% coverage. Individual classes in the main source are also listed with their coverage percentages.

Element	Coverage
BookReviews	81.5 %
src/main/java	65.0 %
com.qa.starterproject.domain	54.5 %
com.qa.starterproject.dto	52.7 %
com.qa.starterproject	37.5 %
com.qa.starterproject.service	99.1 %
com.qa.starterproject.config	100.0 %
com.qa.starterproject.controller	100.0 %
com.qa.starterproject.exception	100.0 %
src/test/java	100.0 %
com.qa.starterproject	100.0 %
com.qa.starterproject.controller	100.0 %
com.qa.starterproject.service	100.0 %

LIVE DEMO!

Sprint Review

- 36 out of 36 issue completed.
- MVP achieved
- Must haves/some should haves achieved
- Back-end tested
- Industry standard test coverage achieved
- Front-end tested
- User friendly application with working CRUD functions

Projects / BookReview

BOOK Sprint 1

Complete front-end and back-end for application

SA [User Icon] Epic ▾

TO DO

IN PROGRESS

DONE 36 ISSUES ✓

Powerpoint

DOCUMENTATION

BOOK-48 ✓ 4

create book repo class

BOOKS BACK-END

BOOK-25 ✓ 2

use sonarqube to minimise smells and errors

TESTING

BOOK-44 ✓ 3

Use JUnit to test the back-end code

TESTING

BOOK-39 ✓ 5

Risk assessment

DOCUMENTATION

BOOK-49 ✓ 1

Developer Journey

10 weeks ago:

- *No experience in:*
 - *SQL*
 - *Java*
 - *JavaScript*
 - *HTML, CSS*
 - *Git*

Today:

- *Successfully completed the MVP for my Books and reviews application using knowledge acquired from the past couple of months*

Thanks QA and trainers ☺

Thank You

Questions?