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 reguraly check github status and that I regurarly 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 reguralry 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 inablity to contact my trainer and push updates to github.	Ensure I have a reliable interent provider, I can also usbe mobile hotspot if it is down.	To make sure I am able to connect to the interenet 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 aslo check QA community rescources, 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 excersice resposibly.	To prevent an injury occuring by being more cautious and aware when I excerise.	Moderate	Major	Medium
Power outtage	This could result in loss of work as well as time.	Enure 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 outtage	Very Unlikely	Hazourdous	Medium
Fire	This could result in a loss of work/ equiptment and injury	Enusre smoke detectors are working, a fire extingusher and blanket are present. As well as a clear exit path.	To minimise damage to equiptment 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*

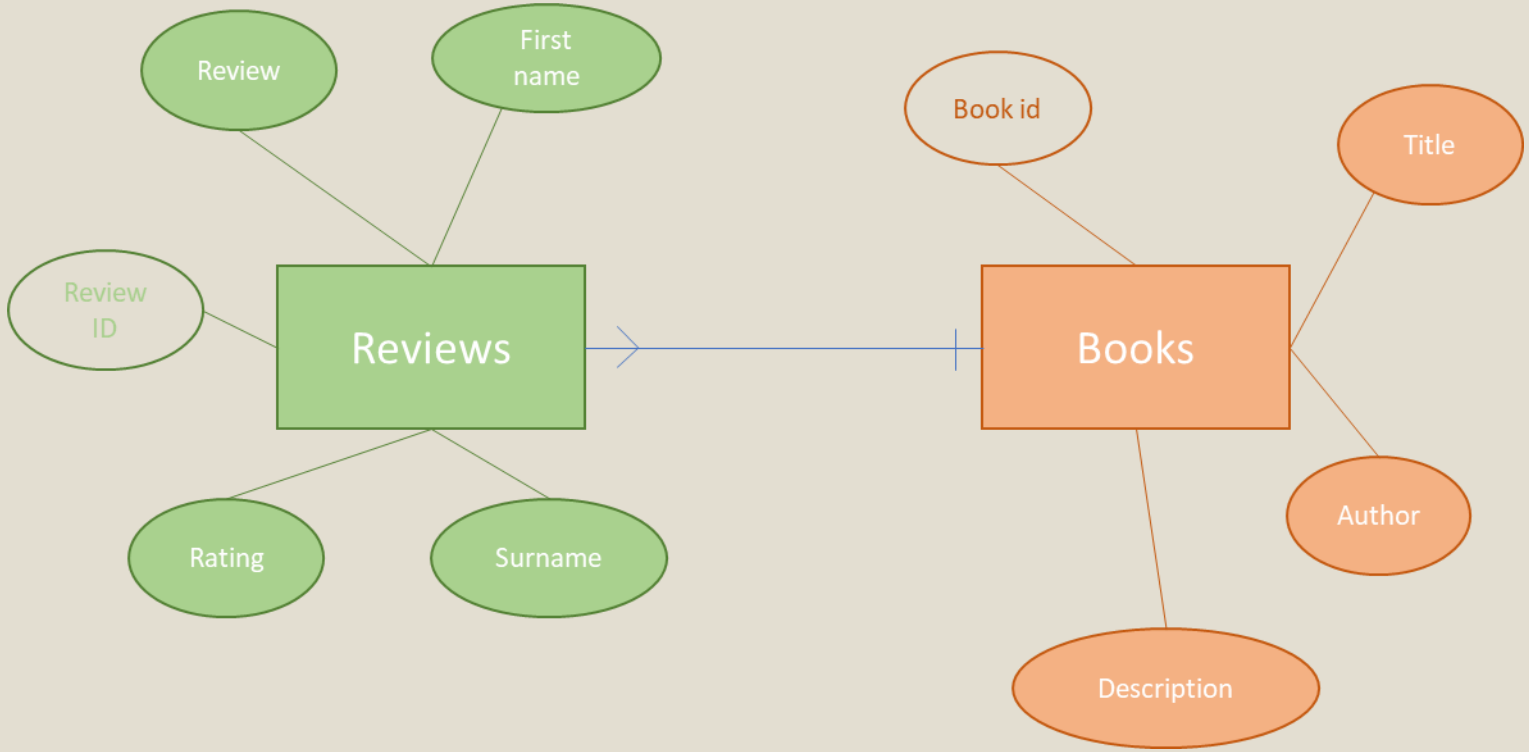
Could have:

- *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

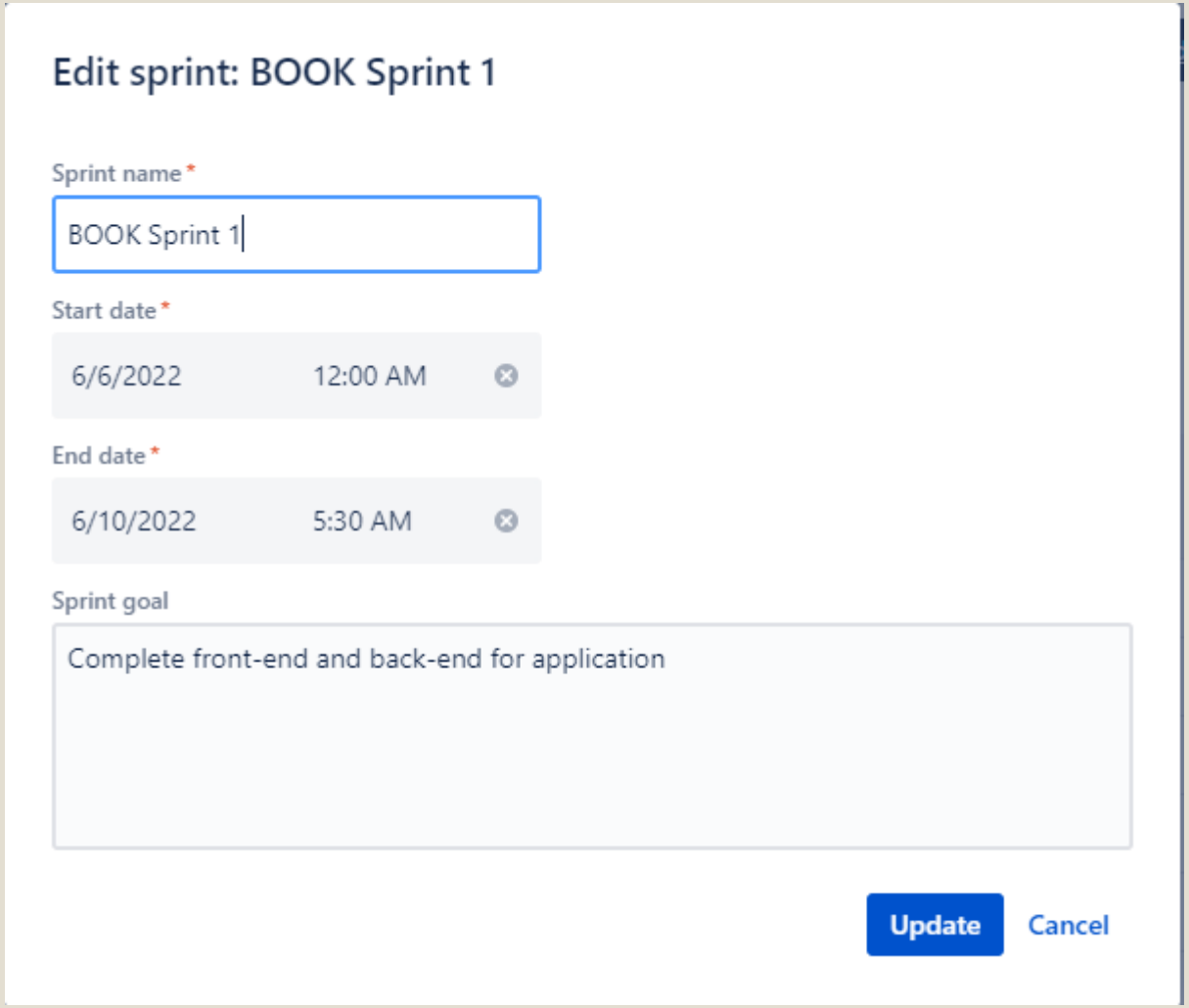
Reviews		
PK	review_id	int
FK	Book_id	int
	first_name	varchar(255)
	surname	Varchar(255)
	rating	int
	review	Varchar(255)

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

1

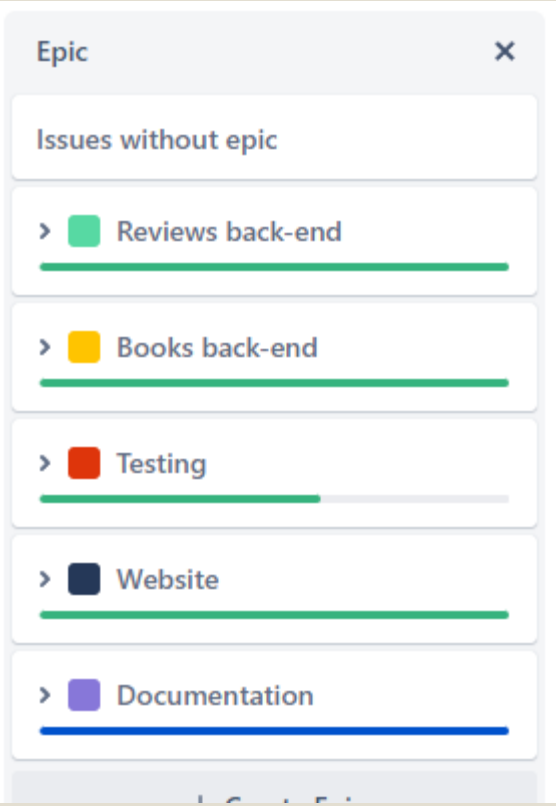
N

Sprint(s)


Edit sprint: BOOK Sprint 1
Sprint name *
BOOK Sprint 1
Start date *
6/6/2022
12:00 AM
End date *
6/10/2022
5:30 AM
Sprint goal
Complete front-end and back-end for application
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-8~~	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...

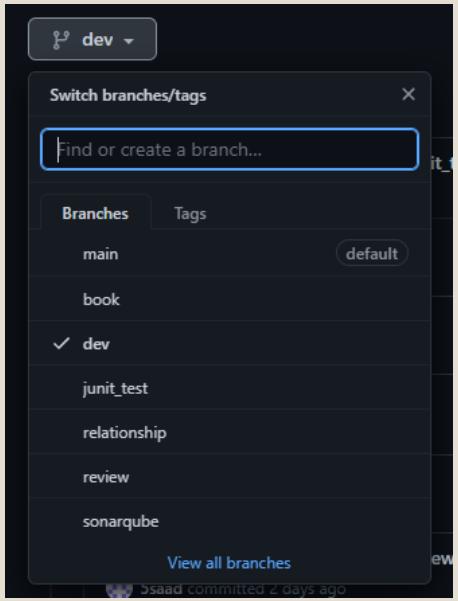
Child issues Order by

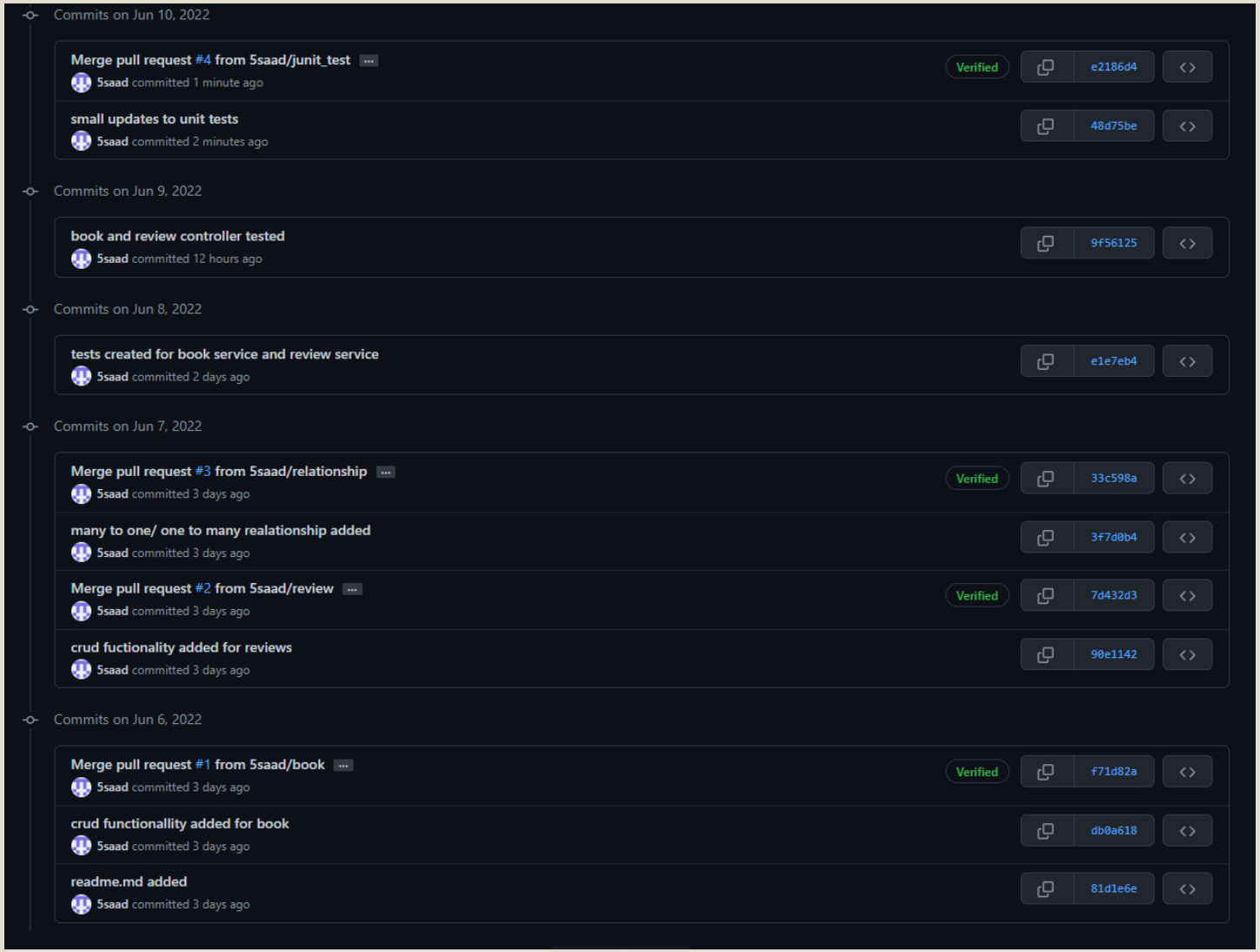
0% Done

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*



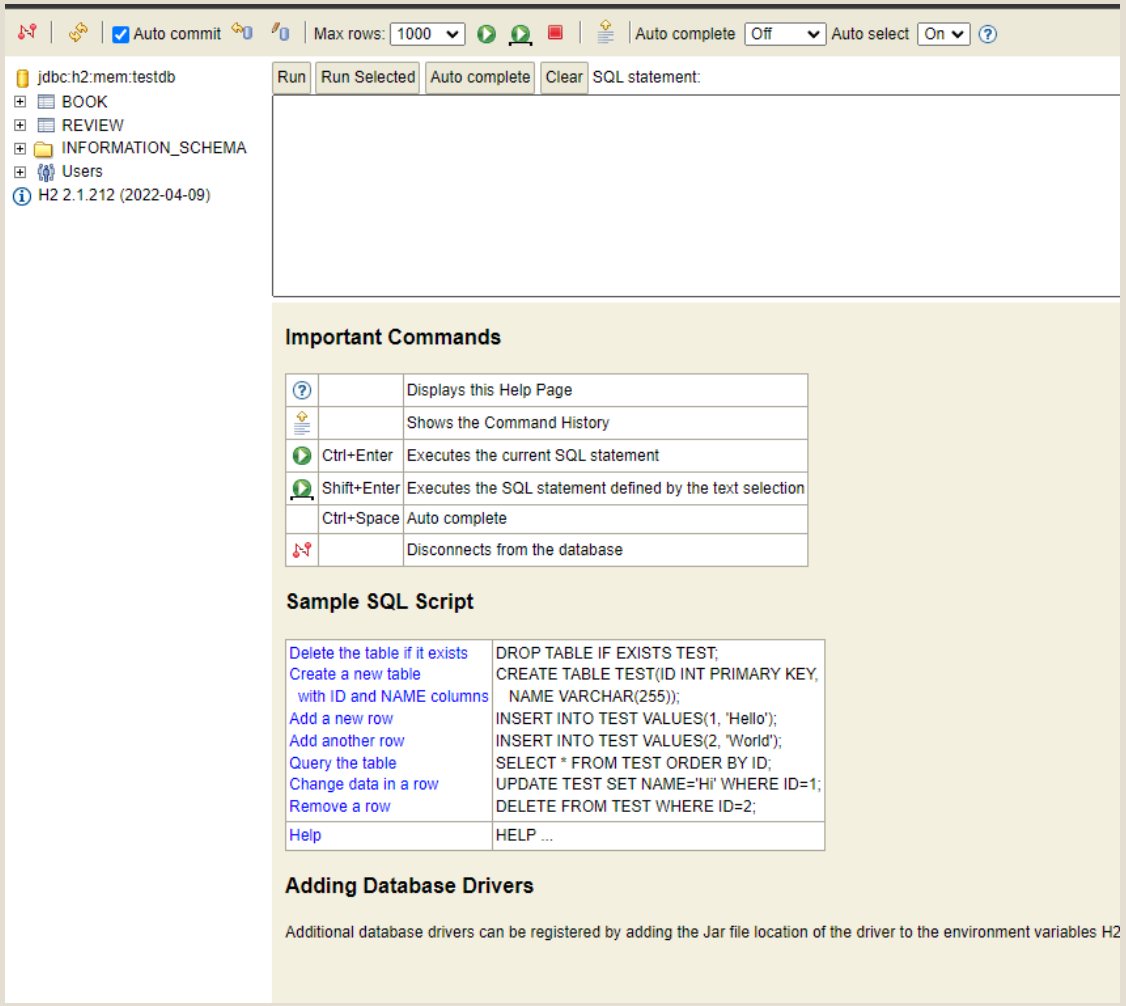


Version Control

- *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!*

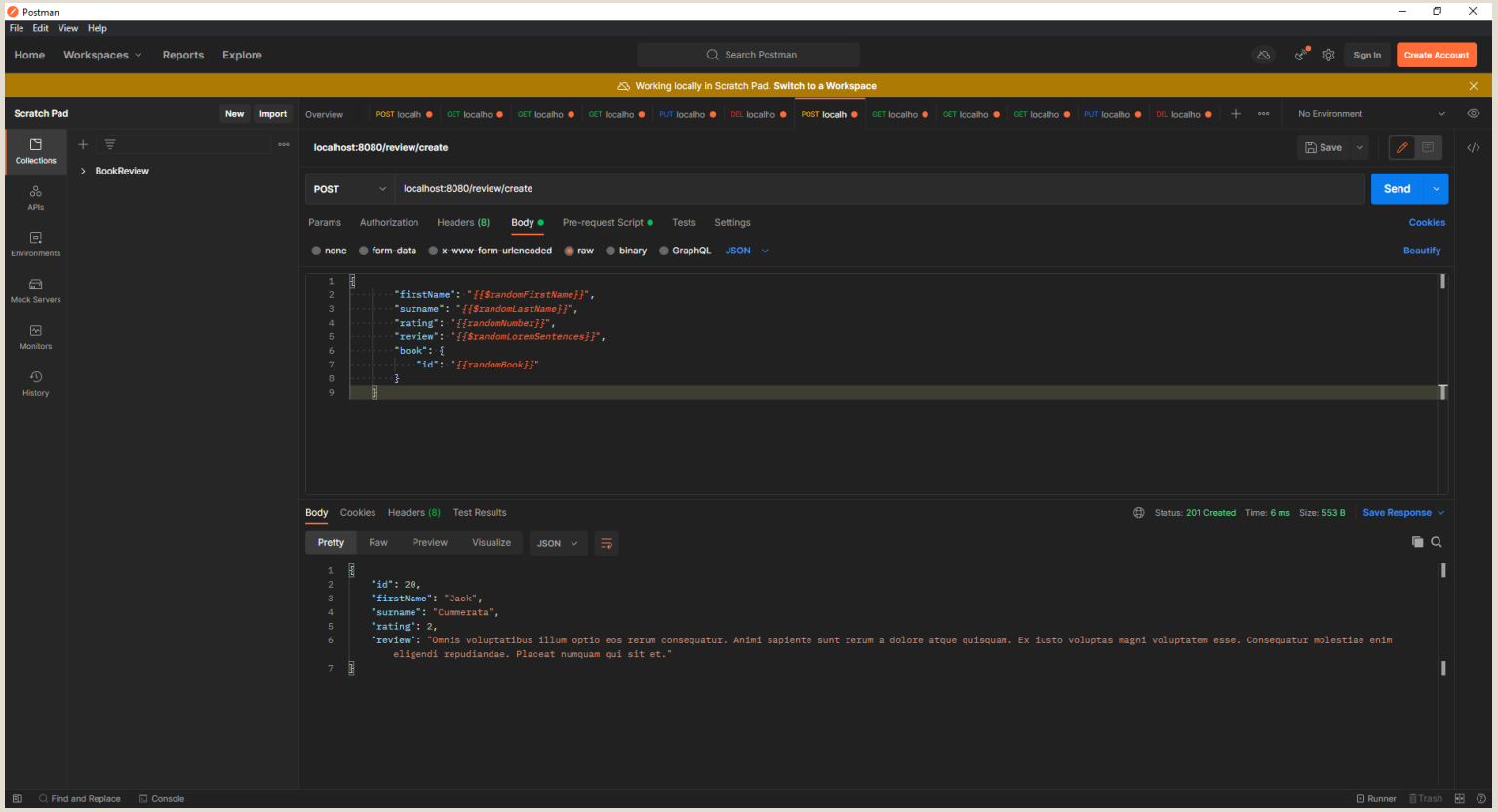
H2-console

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



Postman

- *Manual testing*
- *Random data*



Mockito testing

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

```java
@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"));
```

```java
@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);
}
```

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*

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)

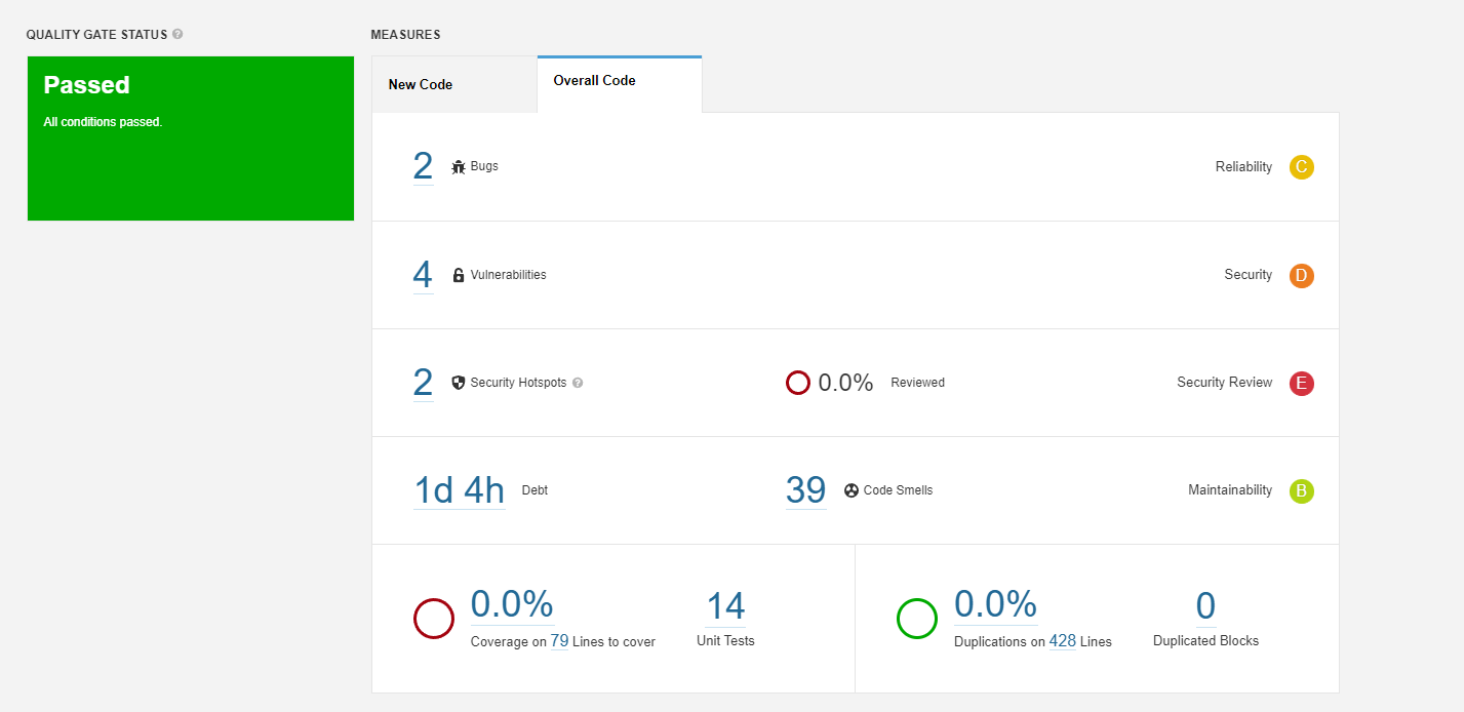
Runs: 7/7 Errors: 0

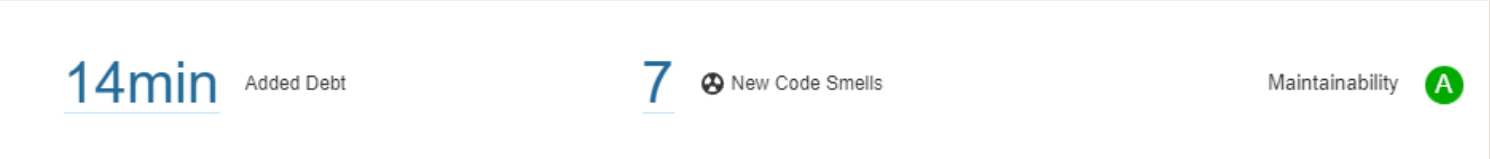
ReviewControllerTest [Runner: JUnit 5] (0.774 s)
- readAllReviewTest() (0.541 s)
- updateTest() (0.093 s)
- readByIdTest() (0.016 s)
- createReviewTest() (0.022 s)
- readByRating() (0.045 s)
- deleteTestNotFound() (0.017 s)
- deleteTest() (0.035 s)

```java
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@ActiveProfiles("dev")
@Sql(scripts = { "classpath:testschema.sql",
        "classpath:testdata.sql" }, executionPhase = ExecutionPhase.BEFORE_TEST_METHOD)
@AutoConfigureMockMvc
public class BookControllerTest {

    @Autowired
    private MockMvc mock;

    @Autowired
    private ObjectMapper mapper;

    private final Long testId = 1L;
    private final String testAuthor = "prof. jeff";
    private final Book testBook = new Book(1L, "the title", "nice book", "prof. jeff");

    private final List<Review> expectedReviews = List.of(new Review(1L, "john", "doe", 4, "good"));
    private final Book testBookRev = new Book(1L, "the title", "nice book", "prof. jeff", expectedReviews);
    private final ArrayList<Book> bookArray = new ArrayList<Book>();

    @Test
    void createBookTest() throws Exception {

        RequestBuilder request = post("/book/create").contentType(MediaType.APPLICATION_JSON)
                .content(this.mapper.writeValueAsString(testBook));
        ResultMatcher responseStatus = status().isCreated();
        ResultMatcher responseContent = content().json(this.mapper.writeValueAsString(testBook));
        this.mock.perform(request).andExpect(responseStatus).andExpect(responseContent);
    }

    @Test
    void readAllBookTest() throws Exception {
        bookArray.add(testBookRev);

        RequestBuilder request = get("/book/readAll").contentType(MediaType.APPLICATION_JSON)
                .accept(MediaType.APPLICATION_JSON);
        ResultMatcher responseStatus = status().isOk();
        ResultMatcher responseContent = content().json(this.mapper.writeValueAsString(bookArray));
        this.mock.perform(request).andExpect(responseStatus).andExpect(responseContent);
    }

    @Test
    void readByIdTest() throws Exception {
        RequestBuilder request = get("/book/read/" + testId).contentType(MediaType.APPLICATION_JSON)
                .accept(MediaType.APPLICATION_JSON);
        ResultMatcher responseStatus = status().isOk();
        ResultMatcher responseContent = content().json(this.mapper.writeValueAsString(testBookRev));
        this.mock.perform(request).andExpect(responseStatus).andExpect(responseContent);
    }
```

SonarQube

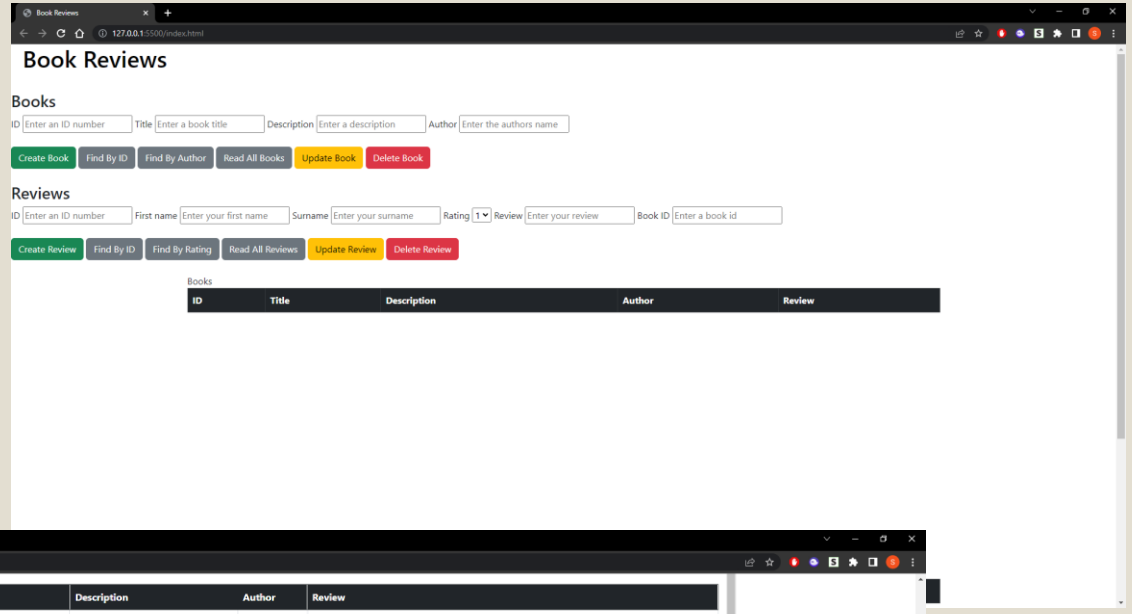
- *Refactoring code*
- *Reduce Smells*
- *Reduce Bugs*
- *Reduce Vulnerabilities*

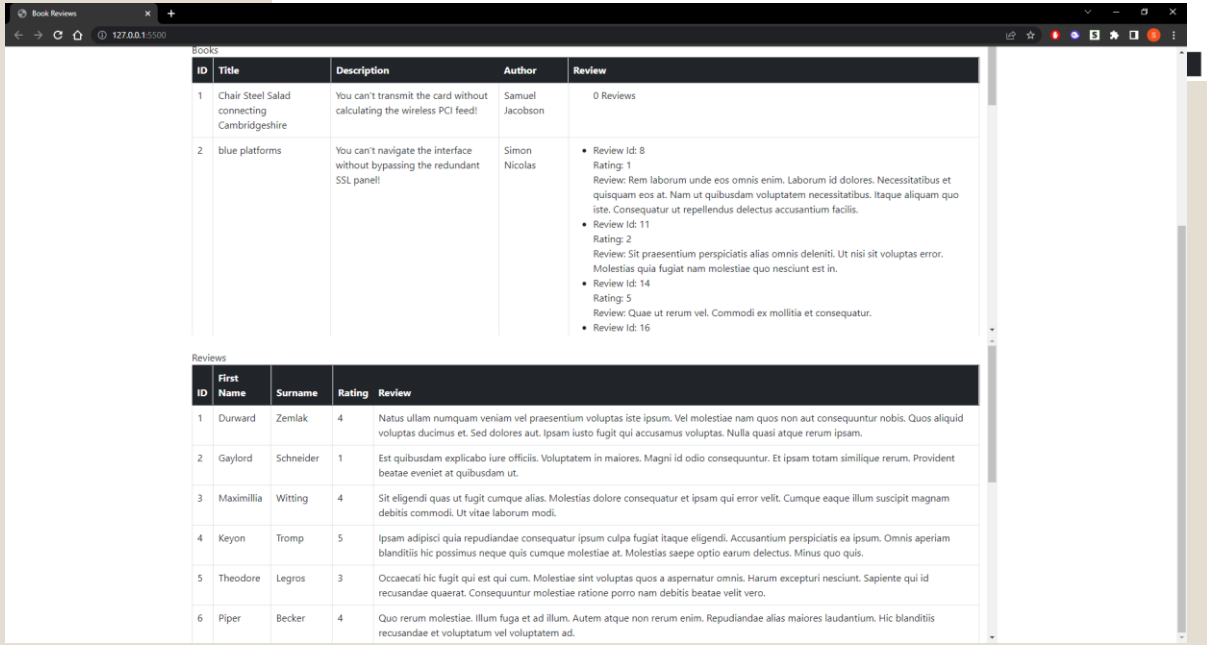




Front-end

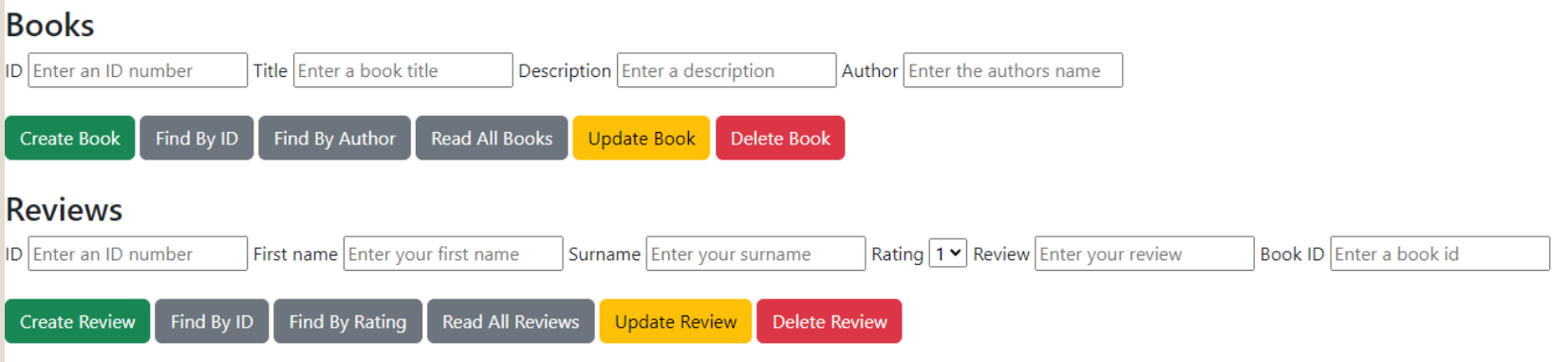
- *Design:*
  - *HTML*
  - *Bootstrap*
- *JavaScript:*
  - *Fetch API*
  - *Retrieve inputs*
  - *Functions for buttons*





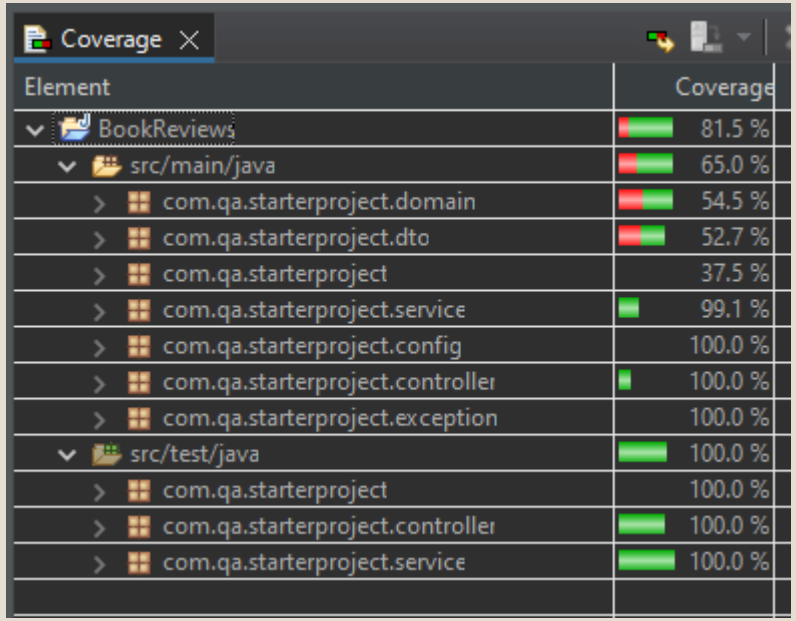
Selenium

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



Test Coverage

- *81.5% Test coverage!*



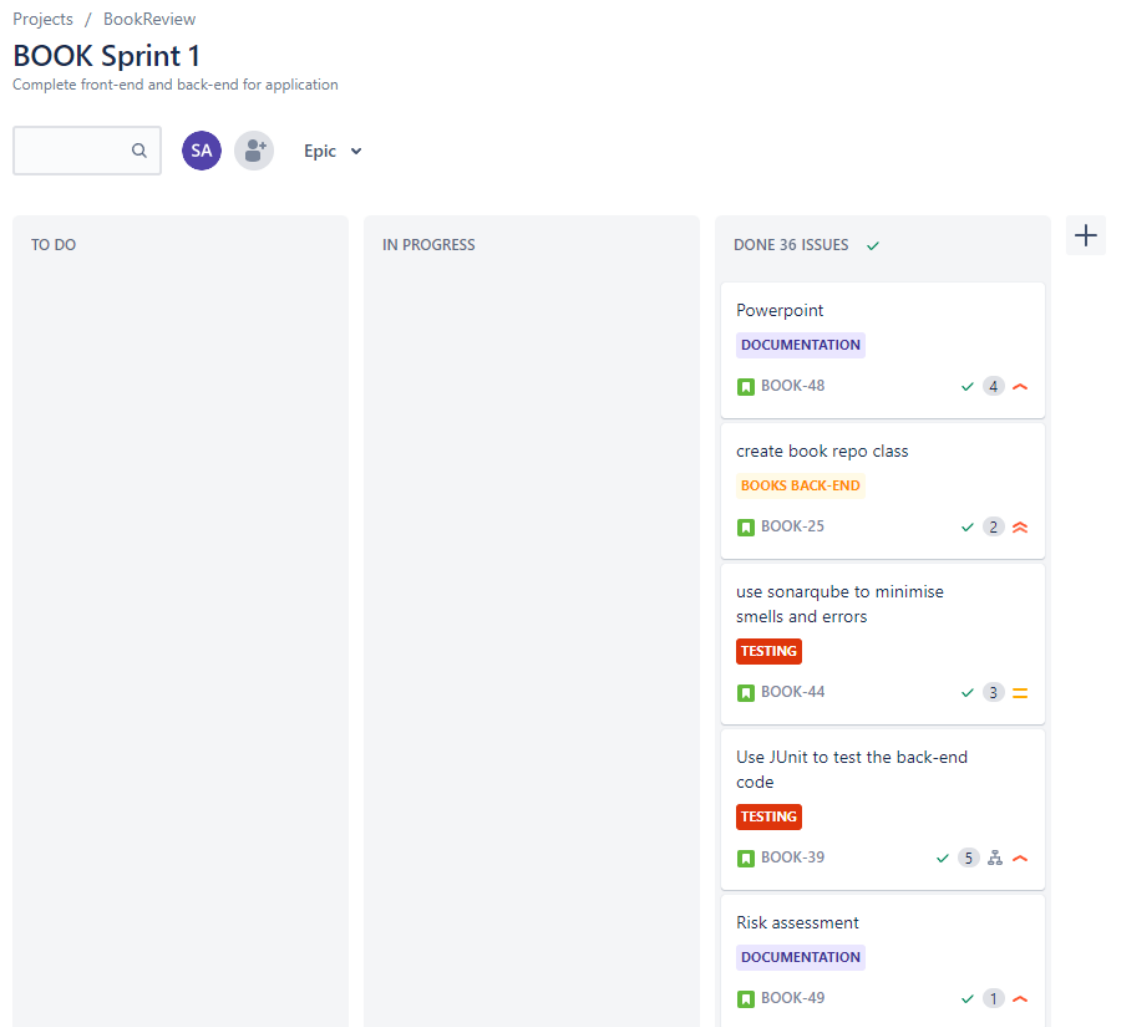
Coverage

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*



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?