Reading List – Hobby Web Application

Syed Zaidi

Introduction

Concept

- An application with some functional use
- A reading list application to keep track of the books you have read/are reading/plan to read

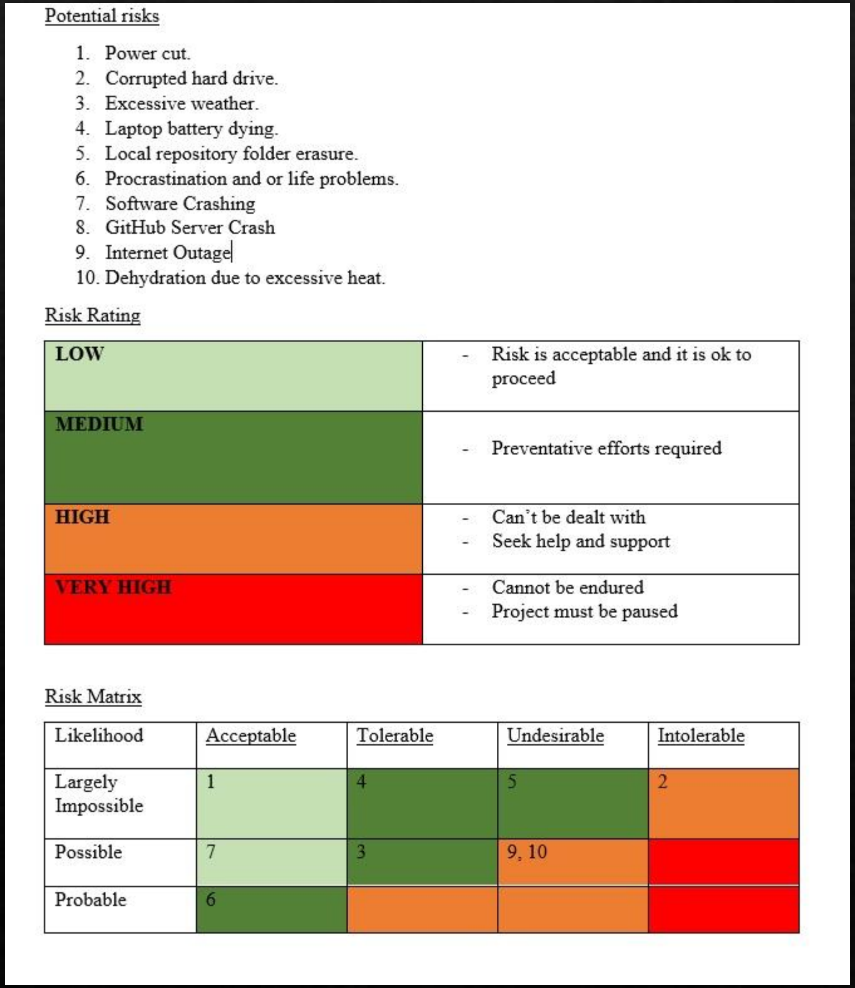
MVP

- Full CRUD functionality
- SQL local database
- Intuitive/Visually pleasing Front End
- Aiming for 80% coverage

Consultant Journey -Technologies Used

- Git/Github
- MySQL/H2
- Java/Springboot (API)
- HTML/CSS/JavaScript
- Maven, Junit, Mockito
- Kanban Board - Jirra

Risk Assessment



Potential risks

1. Power cut.
2. Corrupted hard drive.
3. Excessive weather.
4. Laptop battery dying.
5. Local repository folder erasure.
6. Procrastination and or life problems.
7. Software Crashing
8. GitHub Server Crash
9. Internet Outage
10. Dehydration due to excessive heat.

Risk Rating

LOW	- Risk is acceptable and it is ok to proceed
MEDIUM	- Preventative efforts required
HIGH	- Can't be dealt with - Seek help and support
VERY HIGH	- Cannot be endured - Project must be paused

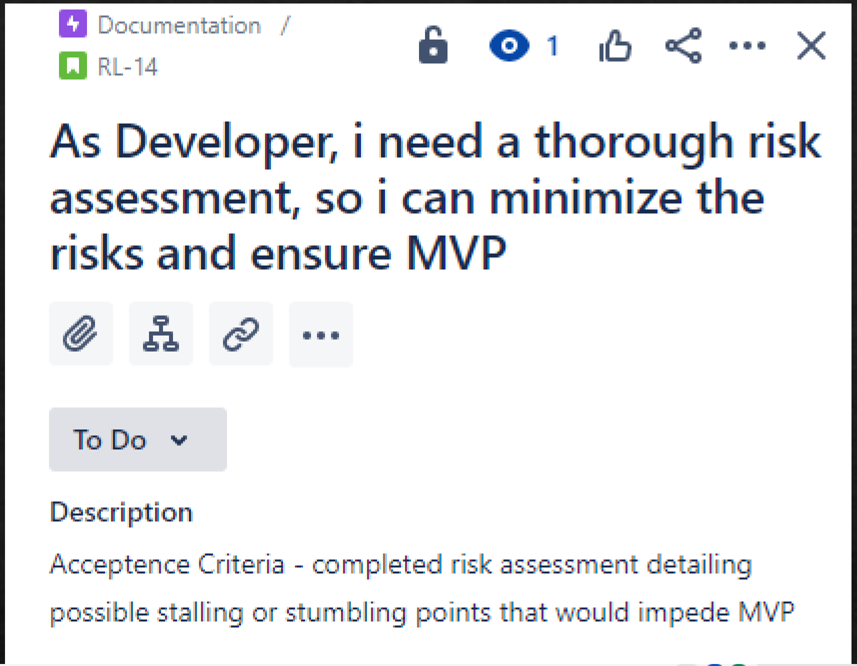
Risk Matrix

Likelihood	Acceptable	Tolerable	Undesirable	Intolerable
Largely Impossible	1	4	5	2
Possible	7	3	9, 10	🟥
Probable	6	🟧	🟧	🟥

- Internet Outage
  - Call ISP to get back online ASAP
- Slow Computer
- Excessive weather/Heat

Sprint Plan

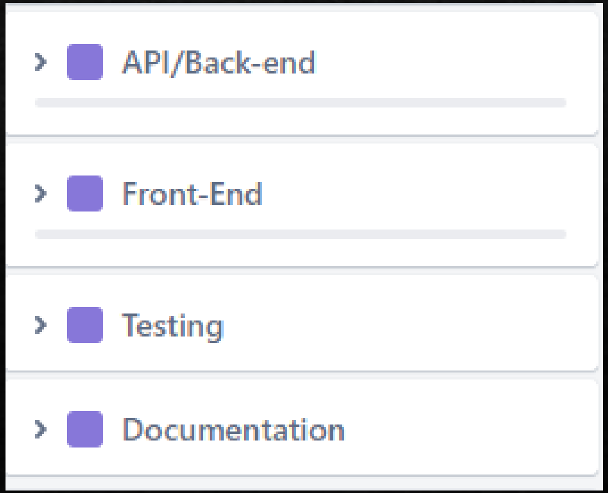
- 4 Epics
- Single sprint – 3 days
- Acceptance criteria/prioritisation

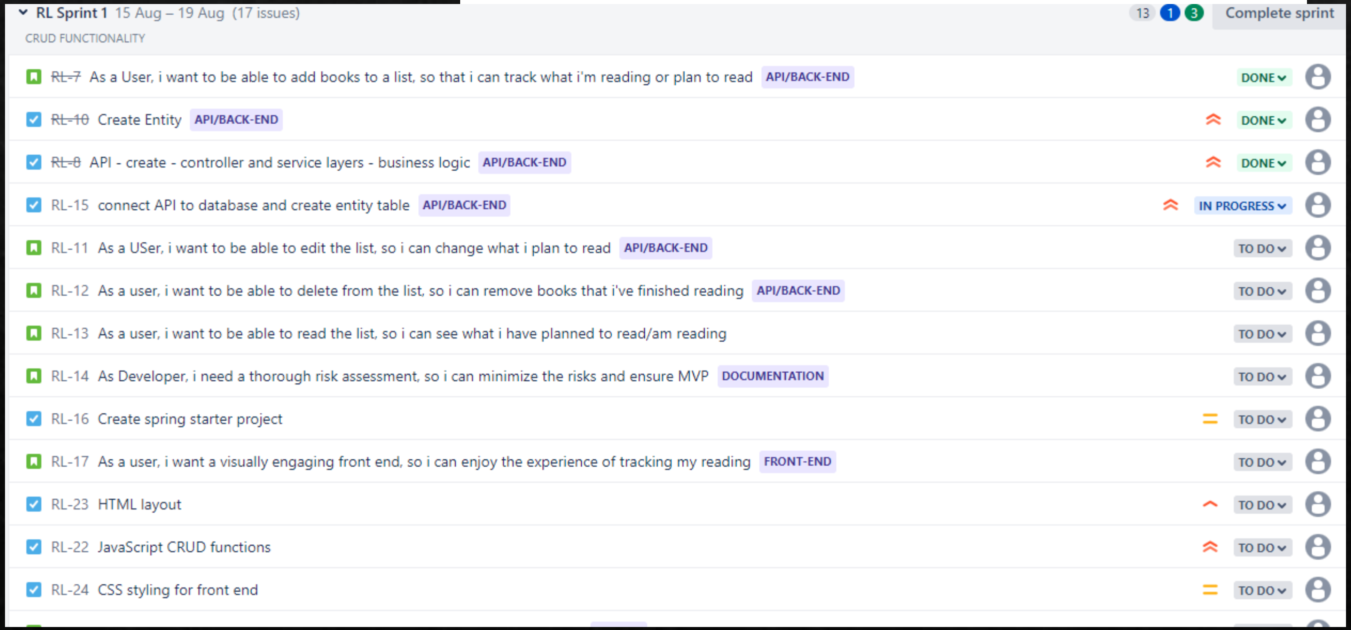

Documentation / RL-14

As Developer, i need a thorough risk assessment, so i can minimize the risks and ensure MVP

Description

Acceptance Criteria - completed risk assessment detailing possible stalling or stumbling points that would impede MVP


- API/Back-end
- Front-End
- Testing
- Documentation


RL Sprint 1 15 Aug – 19 Aug (17 issues)

CRUD FUNCTIONALITY

~~RL-7~~	As a User, i want to be able to add books to a list, so that i can track what i'm reading or plan to read	API/BACK-END	DONE
~~RL-10~~	Create Entity	API/BACK-END	DONE
~~RL-8~~	API - create - controller and service layers - business logic	API/BACK-END	DONE
RL-15	connect API to database and create entity table	API/BACK-END	IN PROGRESS
RL-11	As a USer, i want to be able to edit the list, so i can change what i plan to read	API/BACK-END	TO DO
RL-12	As a user, i want to be able to delete from the list, so i can remove books that i've finished reading	API/BACK-END	TO DO
RL-13	As a user, i want to be able to read the list, so i can see what i have planned to read/am reading		TO DO
RL-14	As Developer, i need a thorough risk assessment, so i can minimize the risks and ensure MVP	DOCUMENTATION	TO DO
RL-16	Create spring starter project		TO DO
RL-17	As a user, i want a visually engaging front end, so i can enjoy the experience of tracking my reading	FRONT-END	TO DO
RL-23	HTML layout		TO DO
RL-22	JavaScript CRUD functions		TO DO
RL-24	CSS styling for front end		TO DO

FBD - Continuous Integration

- Git and Github – Version control
- Main Branch
- Development Branch
- Feature branches

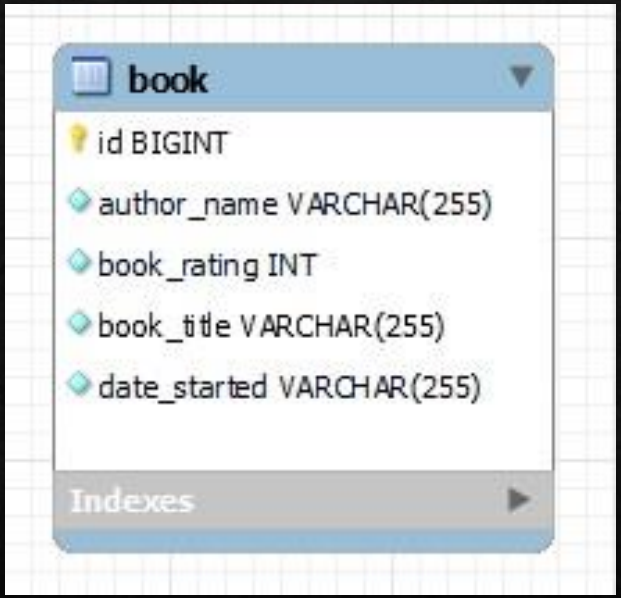
  - API

  - Testing

  - Front End

  -Documentation

ERD



```
id BIGINT
author_name VARCHAR(255)
book_rating INT
book_title VARCHAR(255)
date_started VARCHAR(255)
```

- Book

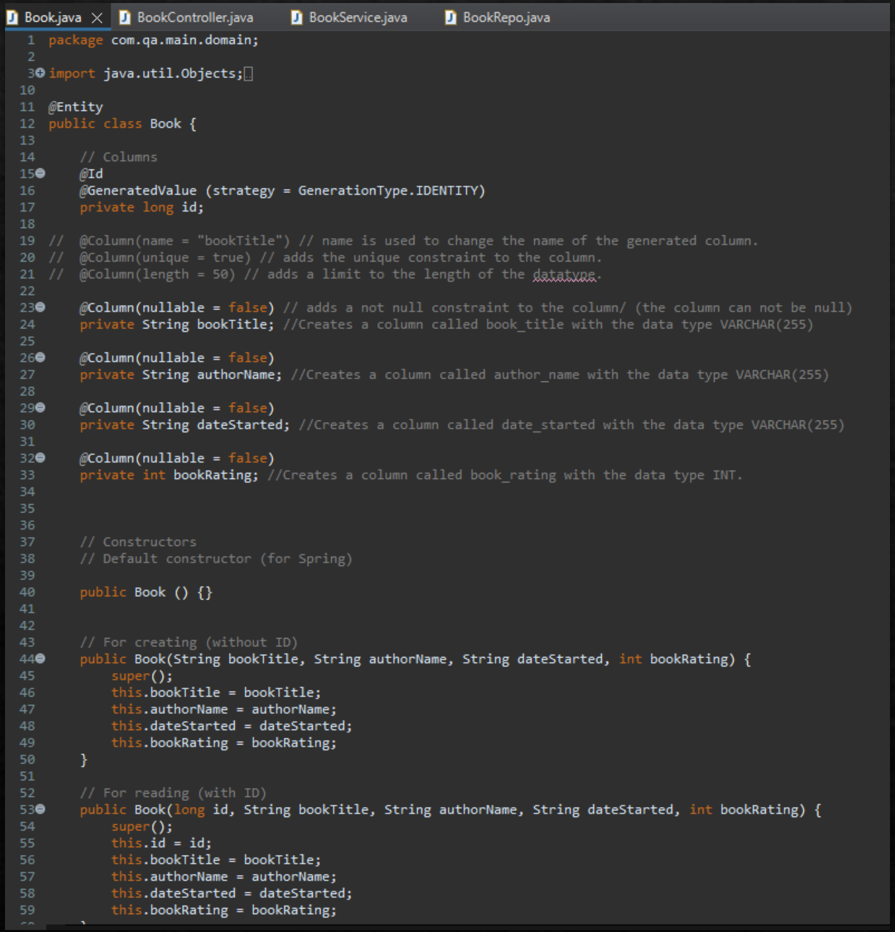
Title

Author

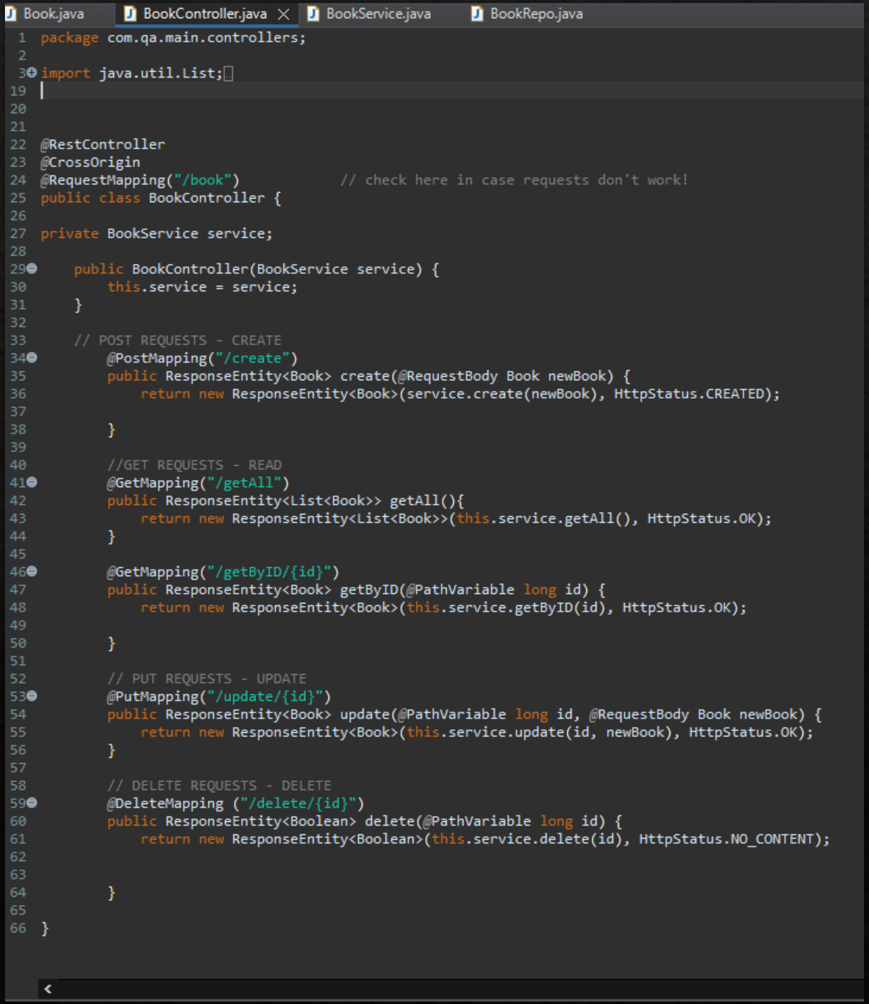
Date Started

Rating

Building the API

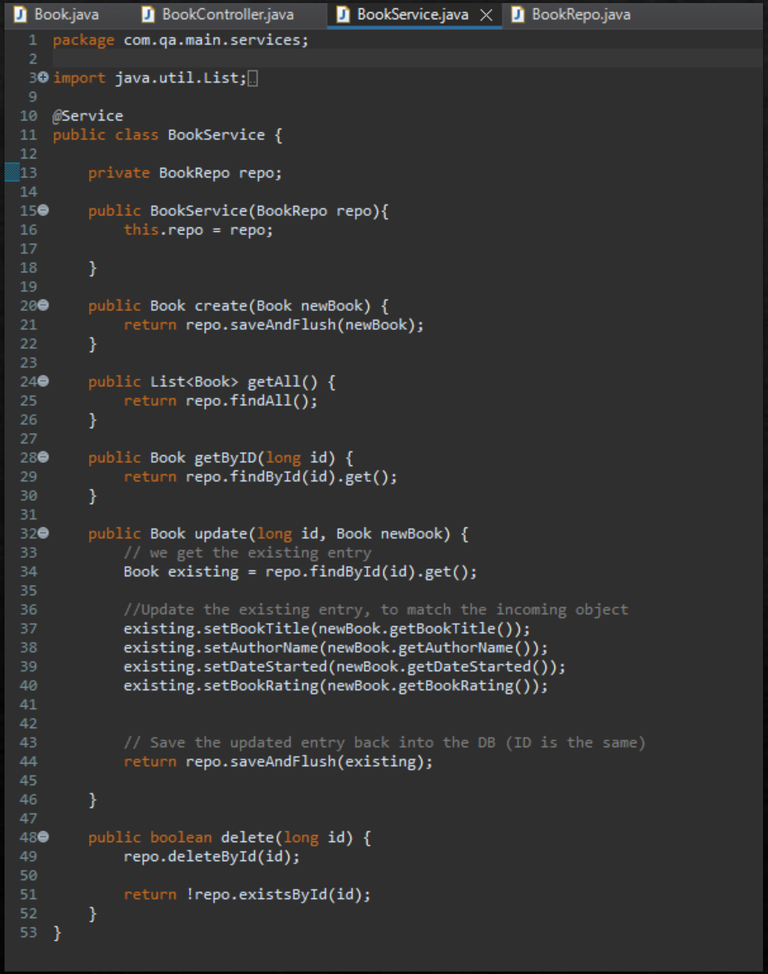


```java
package com.qa.main.domain;

import java.util.Objects;

@Entity
public class Book {

    // Columns
    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    private long id;

//  @Column(name = "bookTitle") // name is used to change the name of the generated column.
//  @Column(unique = true) // adds the unique constraint to the column.
//  @Column(length = 50) // adds a limit to the length of the datatype.

    @Column(nullable = false) // adds a not null constraint to the column/ (the column can not be null)
    private String bookTitle; //Creates a column called book_title with the data type VARCHAR(255)

    @Column(nullable = false)
    private String authorName; //Creates a column called author_name with the data type VARCHAR(255)

    @Column(nullable = false)
    private String dateStarted; //Creates a column called date_started with the data type VARCHAR(255)

    @Column(nullable = false)
    private int bookRating; //Creates a column called book_rating with the data type INT.



    // Constructors
    // Default constructor (for Spring)

    public Book () {}


    // For creating (without ID)
    public Book(String bookTitle, String authorName, String dateStarted, int bookRating) {
        super();
        this.bookTitle = bookTitle;
        this.authorName = authorName;
        this.dateStarted = dateStarted;
        this.bookRating = bookRating;
    }

    // For reading (with ID)
    public Book(long id, String bookTitle, String authorName, String dateStarted, int bookRating) {
        super();
        this.id = id;
        this.bookTitle = bookTitle;
        this.authorName = authorName;
        this.dateStarted = dateStarted;
        this.bookRating = bookRating;
```

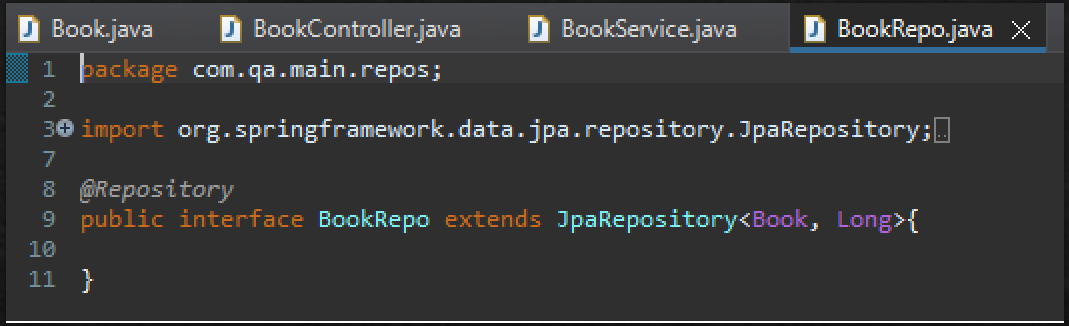


```java
package com.qa.main.controllers;

import java.util.List;



@RestController
@CrossOrigin
@RequestMapping("/book")            // check here in case requests don't work!
public class BookController {

private BookService service;

    public BookController(BookService service) {
        this.service = service;
    }

    // POST REQUESTS - CREATE
        @PostMapping("/create")
        public ResponseEntity<Book> create(@RequestBody Book newBook) {
            return new ResponseEntity<Book>(service.create(newBook), HttpStatus.CREATED);

        }

        //GET REQUESTS - READ
        @GetMapping("/getAll")
        public ResponseEntity<List<Book>> getAll(){
            return new ResponseEntity<List<Book>>(this.service.getAll(), HttpStatus.OK);
        }

        @GetMapping("/getByID/{id}")
        public ResponseEntity<Book> getByID(@PathVariable long id) {
            return new ResponseEntity<Book>(this.service.getByID(id), HttpStatus.OK);

        }

        // PUT REQUESTS - UPDATE
        @PutMapping("/update/{id}")
        public ResponseEntity<Book> update(@PathVariable long id, @RequestBody Book newBook) {
            return new ResponseEntity<Book>(this.service.update(id, newBook), HttpStatus.OK);
        }

        // DELETE REQUESTS - DELETE
        @DeleteMapping ("/delete/{id}")
        public ResponseEntity<Boolean> delete(@PathVariable long id) {
            return new ResponseEntity<Boolean>(this.service.delete(id), HttpStatus.NO_CONTENT);


        }

}
```

Building the API Cont.



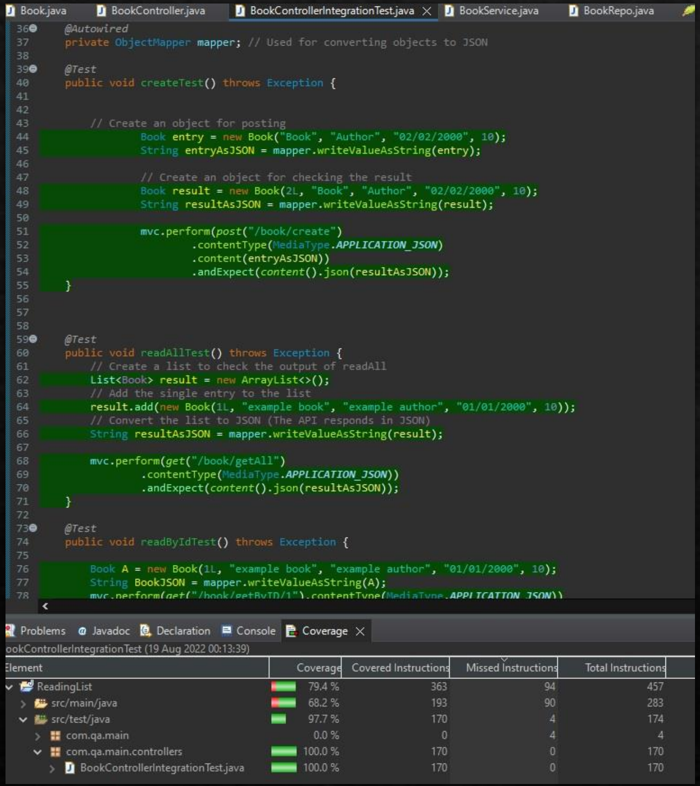
```java
package com.qa.main.services;

import java.util.List;

@Service
public class BookService {

    private BookRepo repo;

    public BookService(BookRepo repo){
        this.repo = repo;

    }

    public Book create(Book newBook) {
        return repo.saveAndFlush(newBook);
    }

    public List<Book> getAll() {
        return repo.findAll();
    }

    public Book getByID(long id) {
        return repo.findById(id).get();
    }

    public Book update(long id, Book newBook) {
        // we get the existing entry
        Book existing = repo.findById(id).get();

        //Update the existing entry, to match the incoming object
        existing.setBookTitle(newBook.getBookTitle());
        existing.setAuthorName(newBook.getAuthorName());
        existing.setDateStarted(newBook.getDateStarted());
        existing.setBookRating(newBook.getBookRating());


        // Save the updated entry back into the DB (ID is the same)
        return repo.saveAndFlush(existing);

    }

    public boolean delete(long id) {
        repo.deleteById(id);

        return !repo.existsById(id);
    }
}
```



```java
package com.qa.main.repos;

import org.springframework.data.jpa.repository.JpaRepository;

@Repository
public interface BookRepo extends JpaRepository<Book, Long>{

}
```

- API Layers
- Entity – Book
- Controller – Functions that call on the logic
- Service – Business Logic
- Repo

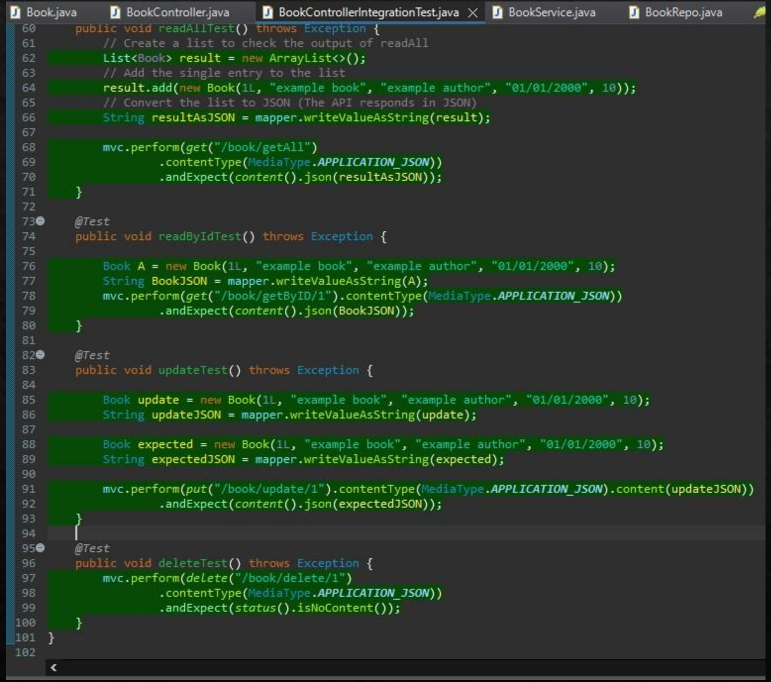
Testing



```java
    @Autowired
    private ObjectMapper mapper; // Used for converting objects to JSON

    @Test
    public void createTest() throws Exception {


        // Create an object for posting
                Book entry = new Book("Book", "Author", "02/02/2000", 10);
                String entryAsJSON = mapper.writeValueAsString(entry);

                // Create an object for checking the result
                Book result = new Book(2L, "Book", "Author", "02/02/2000", 10);
                String resultAsJSON = mapper.writeValueAsString(result);

                mvc.perform(post("/book/create")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(entryAsJSON))
                        .andExpect(content().json(resultAsJSON));
    }



    @Test
    public void readAllTest() throws Exception {
        // Create a list to check the output of readAll
        List<Book> result = new ArrayList<>();
        // Add the single entry to the list
        result.add(new Book(1L, "example book", "example author", "01/01/2000", 10));
        // Convert the list to JSON (The API responds in JSON)
        String resultAsJSON = mapper.writeValueAsString(result);

        mvc.perform(get("/book/getAll")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(content().json(resultAsJSON));
    }

    @Test
    public void readByIdTest() throws Exception {

        Book A = new Book(1L, "example book", "example author", "01/01/2000", 10);
        String BookJSON = mapper.writeValueAsString(A);
        mvc.perform(get("/book/getByID/1").contentType(MediaType.APPLICATION_JSON))
```

ookControllerIntegrationTest (19 Aug 2022 00:13:39)

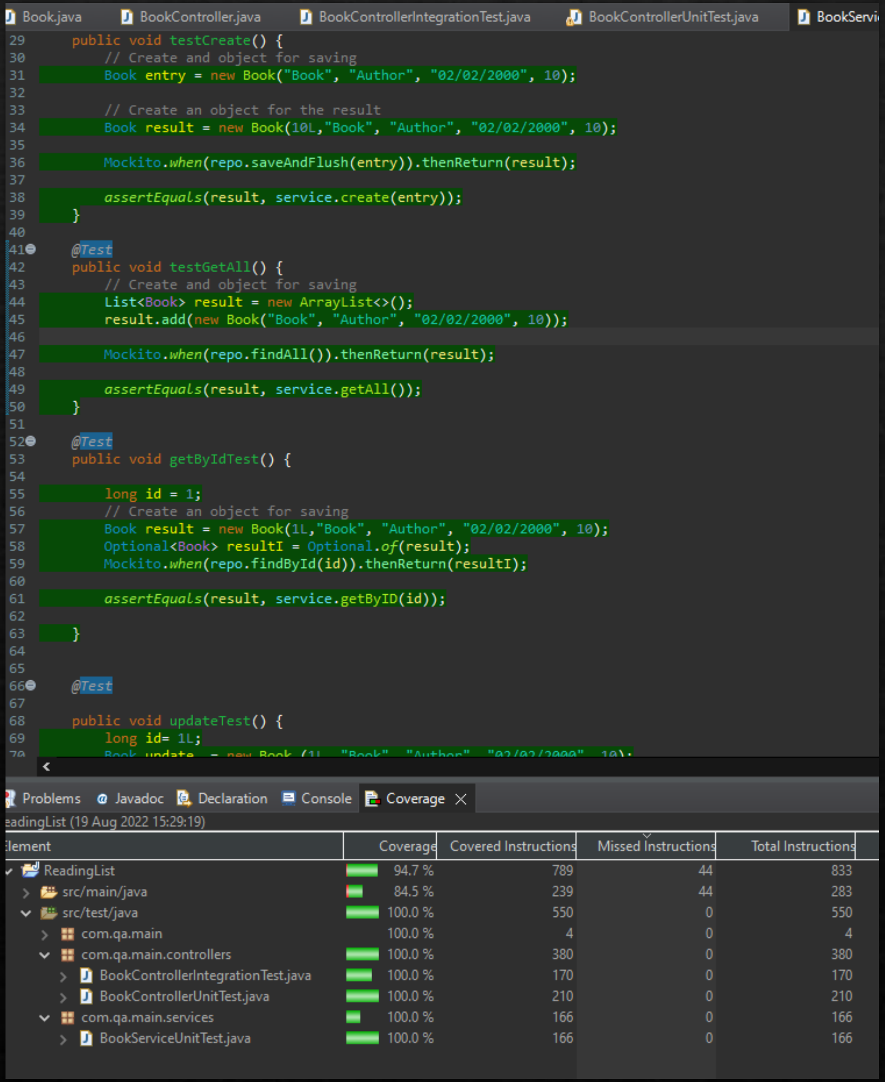
lement	Coverage	Covered Instructions	Missed Instructions	Total Instructions
ReadingList	79.4 %	363	94	457
src/main/java	68.2 %	193	90	283
src/test/java	97.7 %	170	4	174
com.qa.main	0.0 %	0	4	4
com.qa.main.controllers	100.0 %	170	0	170
BookControllerIntegrationTest.java	100.0 %	170	0	170



```java
    public void readAllTest() throws Exception {
        // Create a list to check the output of readAll
        List<Book> result = new ArrayList<>();
        // Add the single entry to the list
        result.add(new Book(1L, "example book", "example author", "01/01/2000", 10));
        // Convert the list to JSON (The API responds in JSON)
        String resultAsJSON = mapper.writeValueAsString(result);

        mvc.perform(get("/book/getAll")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(content().json(resultAsJSON));
    }

    @Test
    public void readByIdTest() throws Exception {

        Book A = new Book(1L, "example book", "example author", "01/01/2000", 10);
        String BookJSON = mapper.writeValueAsString(A);
        mvc.perform(get("/book/getByID/1").contentType(MediaType.APPLICATION_JSON))
                .andExpect(content().json(BookJSON));
    }

    @Test
    public void updateTest() throws Exception {

        Book update = new Book(1L, "example book", "example author", "01/01/2000", 10);
        String updateJSON = mapper.writeValueAsString(update);

        Book expected = new Book(1L, "example book", "example author", "01/01/2000", 10);
        String expectedJSON = mapper.writeValueAsString(expected);

        mvc.perform(put("/book/update/1").contentType(MediaType.APPLICATION_JSON).content(updateJSON))
                .andExpect(content().json(expectedJSON));
    }

    @Test
    public void deleteTest() throws Exception {
        mvc.perform(delete("/book/delete/1")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isNoContent());
    }
}
```

- Integration and Unit Testing
- Some errors initially – testdata/testschema

Unit Testing



```java
    public void testCreate() {
        // Create and object for saving
        Book entry = new Book("Book", "Author", "02/02/2000", 10);

        // Create an object for the result
        Book result = new Book(10L,"Book", "Author", "02/02/2000", 10);

        Mockito.when(repo.saveAndFlush(entry)).thenReturn(result);

        assertEquals(result, service.create(entry));
    }

    @Test
    public void testGetAll() {
        // Create and object for saving
        List<Book> result = new ArrayList<>();
        result.add(new Book("Book", "Author", "02/02/2000", 10));

        Mockito.when(repo.findAll()).thenReturn(result);

        assertEquals(result, service.getAll());
    }

    @Test
    public void getByIdTest() {

        long id = 1;
        // Create an object for saving
        Book result = new Book(1L,"Book", "Author", "02/02/2000", 10);
        Optional<Book> resultI = Optional.of(result);
        Mockito.when(repo.findById(id)).thenReturn(resultI);

        assertEquals(result, service.getByID(id));

    }


    @Test

    public void updateTest() {
        long id= 1L;
        Book update  = new Book (1L, "Book", "Author", "02/02/2000", 10);
```

eadingList (19 Aug 2022 15:29:19)

lement	Coverage	Covered Instructions	Missed Instructions	Total Instructions
ReadingList	94.7 %	789	44	833
src/main/java	84.5 %	239	44	283
src/test/java	100.0 %	550	0	550
com.qa.main	100.0 %	4	0	4
com.qa.main.controllers	100.0 %	380	0	380
BookControllerIntegrationTest.java	100.0 %	170	0	170
BookControllerUnitTest.java	100.0 %	210	0	210
com.qa.main.services	100.0 %	166	0	166
BookServiceUnitTest.java	100.0 %	166	0	166

- Overall Project coverage – 94.7%

Demonstration

Sprint Review/Retrospective

In hindsight :-

- Shorter sprints– compensate for risks
- Break user stories down further into minutae
- Achieved CRUD – However Front end visually could be more cohesive.
- Greater variables - Page Number, Started reading/finished.

Thank You for listening!