Spring Boot Lab:

Simple Multi Layer Application:

Development of a simple CRUD using MongoDB

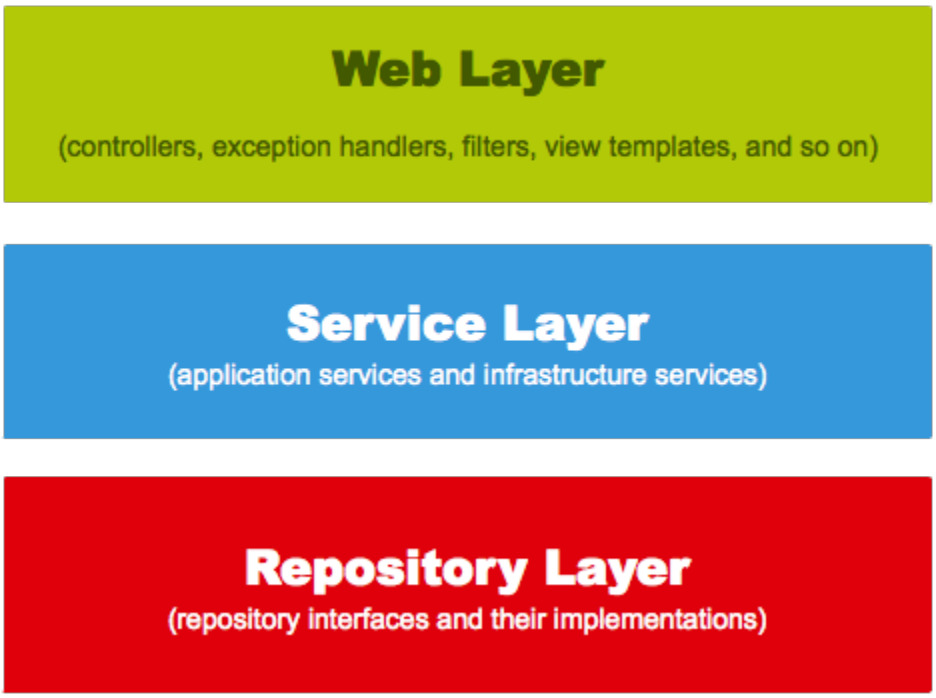
DESCRIPTION

The application is a simple application to save and query a collection of books. Each book has an:

i) Id (integer and unique)
ii) Title (text)
iii) Author (text)

OBJECTIVES

Create a multi-layer application that uses MongoDB and performs some basic CRUD application (in this lab you will do GET and PUT). The architecture of the application is illustrated by the following diagram:



Tasks:

1- Development of API Layer to do CRUD
2- Development of Service Layer to get and create book items from database
3- To create and access MongoDB database
4- Wiring up all layers and annotating classes and methods
5- Testing GET and PUT methods using a client (we will use Postman)

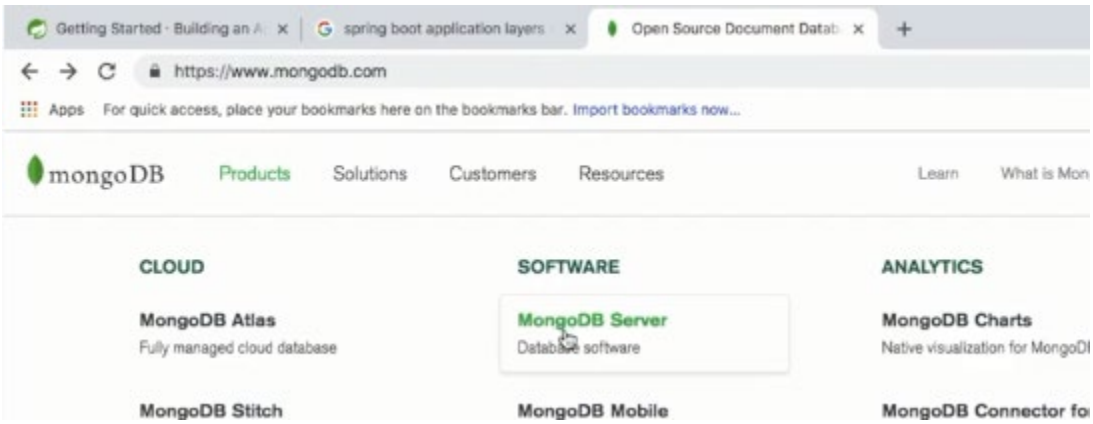
You have 30 minutes time to work on this Lab. At the end of the Lab, the solutions will be discussed.

Follow the following steps and build the application.

1- Check if you have MongoDB

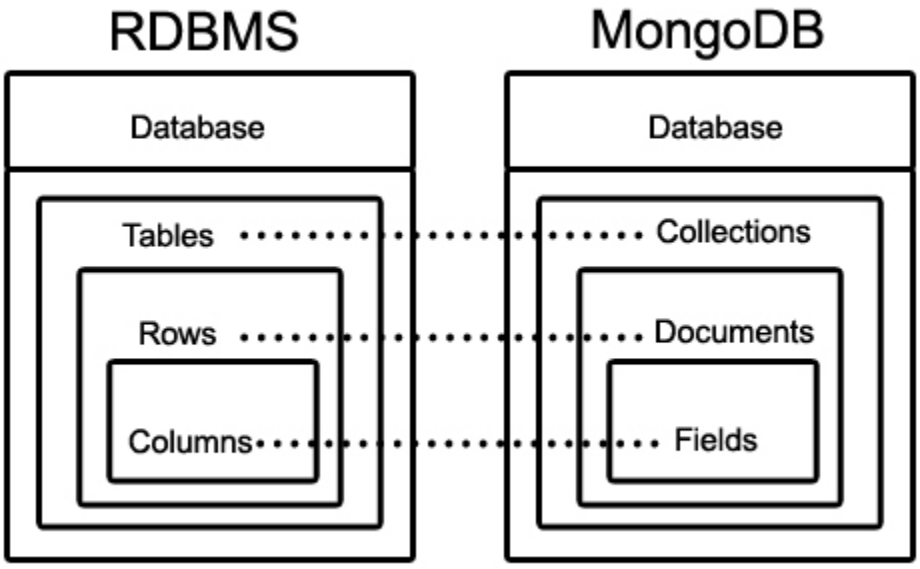
```
➜  ~
➜  ~
➜  ~
➜  ~ mongod -version
db version v4.0.4
git version: f288a3bdf201007f3693c58e140056adf8b04839
allocator: system
modules: none
build environment:
    distarch: x86_64
    target_arch: x86_64
➜  ~
```

Or install



2- Create a Spring Boot App with Web and MongoDB dependencies
3- Create each of the following packages
    - controller
    - service
    - DAO
4- Inside the controller package create a class for the controller, name it “bookController”
5- Annotage bookConroller class with @RestController
6- Add endpoint using annotation @RequestMapping("/books")
        First test:
            Add getBooks method that returns String (e.g. “hello world”)
            Test the server working
            Annotate method with @GetMapping to identify it is for GET
            Now you can test (need have MongoDB running or for now disable it in pom.xml
7- In service package called BookService

8- Annotate BookService with @Service to identify it is sercice layer
9- Create a getBooks method that returns Collection<Book>
    - For Book you need to create an Entity
10- Create another package named “Entity”
    - Inside Entity create a new class called “Book.java”
    - Annotate it with @Document(collection ="Book") (this MongoDB will use)
        Collection will be named Book that is a Table in MongoDB



11- Create some properties for Book according to the above mentioned types
    - Id,
    - Title,
    - Author
12- Generate getters and setters (remember Id will be unique so we won’t provide it)
13- Inside “BookService” class, reference the Book Entity by importing the Entity
14- Create a method “getBooks()” that will return Collection<Books>
    - (return should be from Database, so for now use return bookDAO.getBooks (error will be fixed in next steps)
15- Create DAO package
16- Inside DAO package, create BookDAO class
17- Inside the DAO package:
    - Create an Interface, BookRepository that extends MongoRepository and will use <Book, Integer> that implements the type of data (Book) and ID (Integer)
      (inor: an empty Class is just fine for now, but later if you need to add like find by title or other methods, you can define inside the class)
18- Inside DAO create a Class, named BookDAO
19- Annotate the class with @Component (so spring can scan it)

20- Create property: private BookRepository repository

21- Need to create two methods:
    - getBooks() that will return Collection<book> by implanting repository.findAll()
    - createBooks() that will save a book by repository.insert(book)
22- add @Autowire to BookRepository repository
23- at BookService.java, we still need to define bookDAO and Autowire it
    - create a property
        - private BookDAO bookDAO
        - don't forget to Autowire
24- Still we need to fix BookController, to return Collection<Book> instead of String so we get getBooks() from Service
    - Add return bookService.getBooks()
        i. For this to work you need to create BookService bookService and Autowire it

Now you can run but you get empty JASON as there is no record.

Now use Postman to create some data, (If you don't have Postman, install it. https://www.postman.com/downloads/)

For this we need to have POST method to create data

25- Add @PostMapping() below @GetMapping in BookController
    - Define the method name it postBook() that returns Book and parameter is Book book.

```
@PostMapping()
public Book postBook(@RequestBody Book book) {
    return bookService.createBook(book);
```

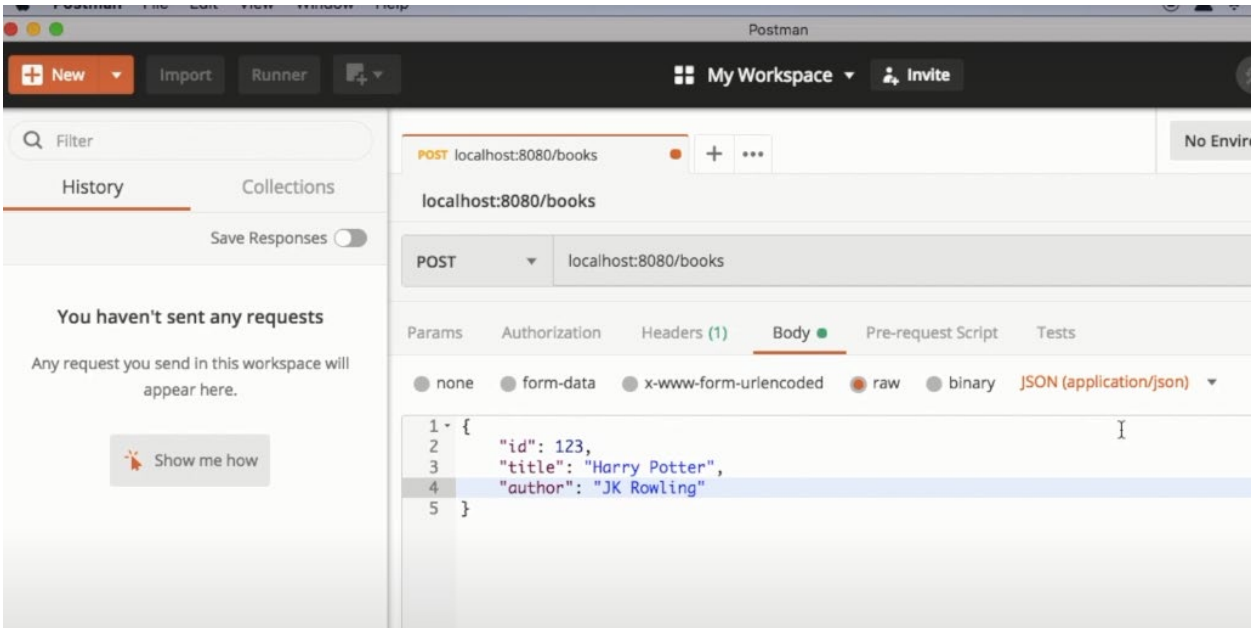
For this we need to add createBook() method in bookService

```
public Book createBook(Book book) {
    return bookDAO.insert(book);
}
```

26- This method needs to be implemented at DAO Layer too.

```
public Book createBook(Book book) {
    return repository.insert(book);
}
```

this should work as end to end solution, and we can test is with postman.



Use postman to GET and test if your POST was registered.

The only problem so far is that we can't see the database in MongoDB (can check with Compass Community)

For that there are some steps:

27- In the resources/application.propeties add:

```
Spring.data.mongodb.database=Library
```

This gives the name of the database as Library

28- You need to post again and now you can check if the database is by posting again through Postman.

Work on it in your groups, and we will revisit it during the rest of the class.