Demo 1

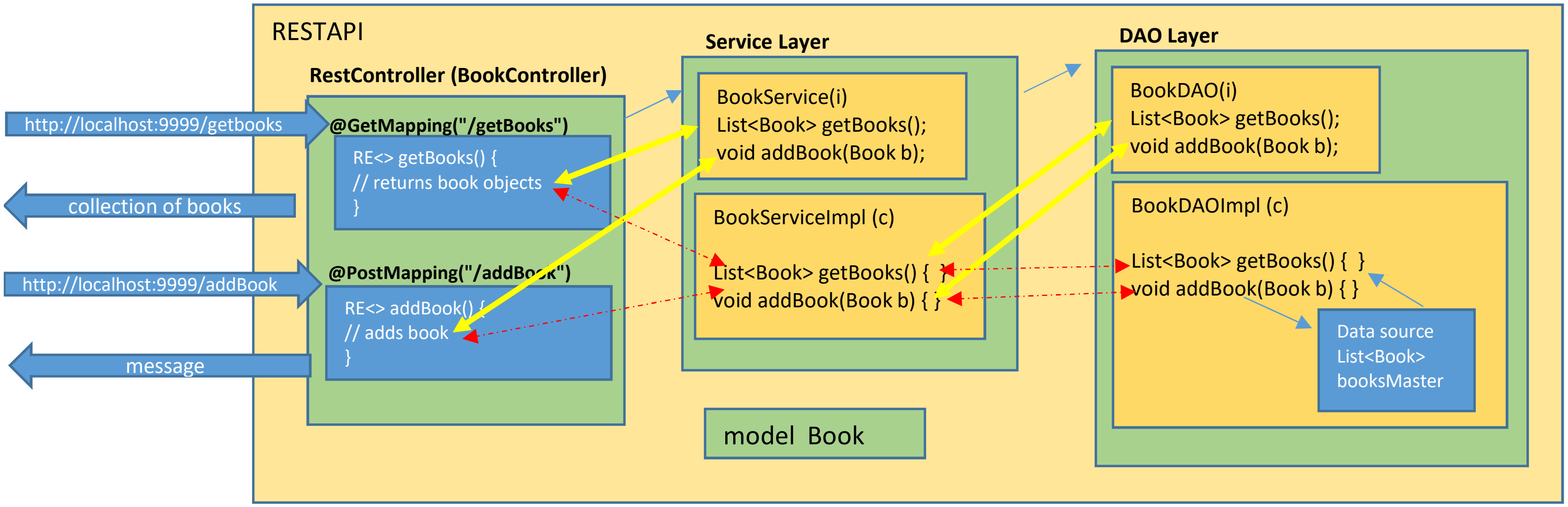
Controller (source)

Demo 2

Controller <-> Service (source)

Demo 3

Controller <-> Service <-> Dao (source)



RESTAPI

RestController (BookController)

http://localhost:9999/getbooks

@GetMapping("/getBooks")

RE<> getBooks() {
// returns book objects
}

collection of books

http://localhost:9999/addBook

@PostMapping("/addBook")

RE<> addBook() {
// adds book
}

message

Service Layer

BookService(i)
List<Book> getBooks();
void addBook(Book b);

BookServiceImpl (c)

List<Book> getBooks() { }
void addBook(Book b) { }

model Book

DAO Layer

BookDAO(i)
List<Book> getBooks();
void addBook(Book b);

BookDAOImpl (c)

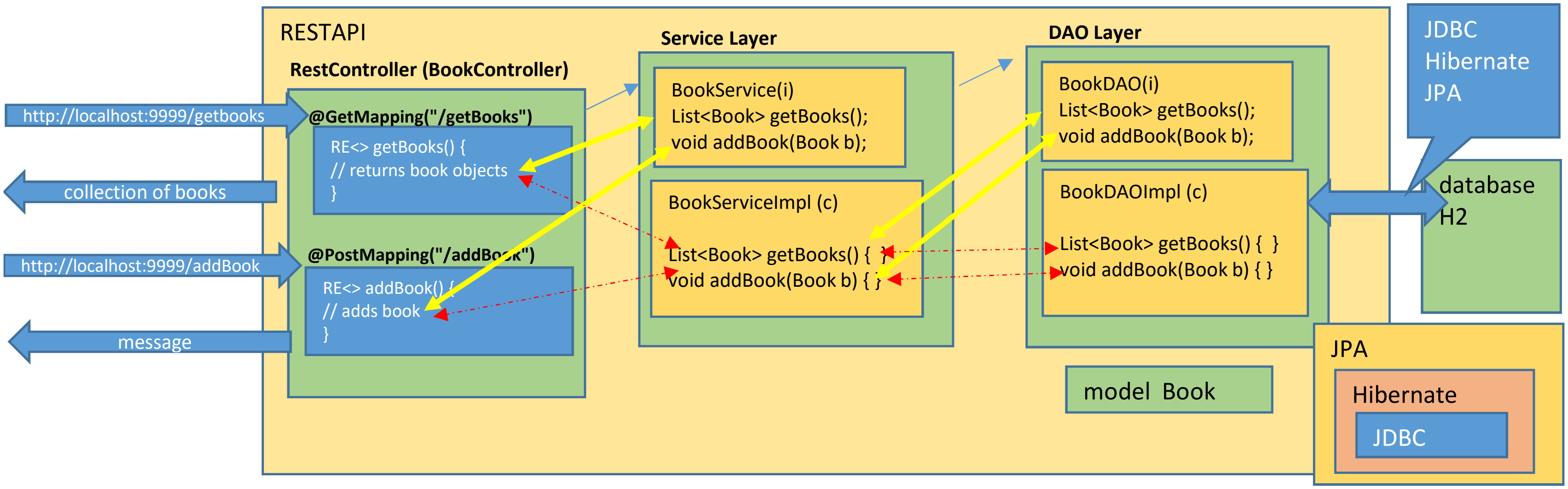
List<Book> getBooks() { }
void addBook(Book b) { }

Data source
List<Book>
booksMaster

Here, data managed in collection, collection is part of application only

get all record	get record by id
add new record	update record
	delete record

How to store/get data from database



JPA
Java Persistence API
Includes collection of interfaces and classes to perform DB operations

Three diff interfaces available in JPA framework to make java application to interact with DB

CrudRepository
JpaRepository
MongoRepository

Why to use JPA

There are 2 major limitations when programmer directly use JDBC

classes in java not mapped with Table in DB
objects in java not mapped with records in table

1 Java class ---> Table in DB
java object this mapping to be done manually in JDBC

2 programmer must know queries

in JDBC
java object convert to record
record convert to java object

both above to convert manually

Java
object

java obj

PreparedSt
st

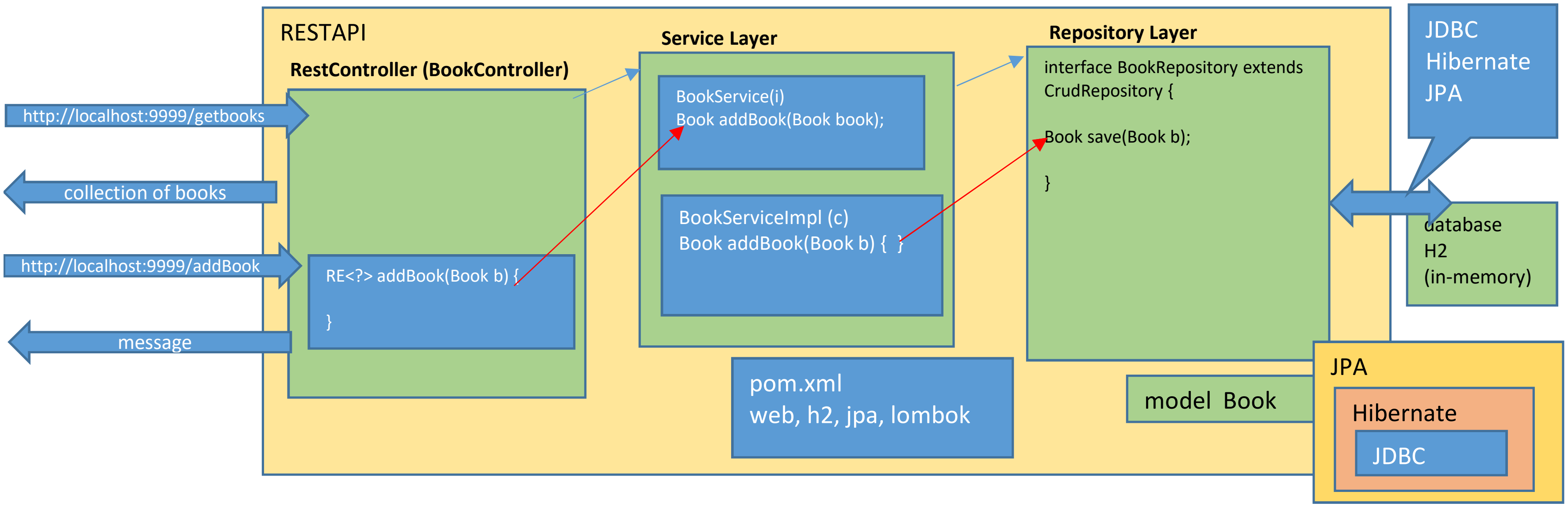
executeUpdate()

DB
records in table

bkid	bkname	bksubject
B001	Abcd	Xyz
B002	Mnop	Pqr

DML insert/update/delete
DQL select (sorting/where/betwe..)

How to store/get data from database



RESTAPI

RestController (BookController)

http://localhost:9999/getbooks

collection of books

http://localhost:9999/addBook

message

RE<?> addBook(Book b) {

}

Service Layer

BookService(i)
Book addBook(Book book);

BookServiceImpl (c)
Book addBook(Book b) { }

Repository Layer

interface BookRepository extends
CrudRepository {

Book save(Book b);

}

JDBC
Hibernate
JPA

database
H2
(in-memory)

pom.xml
web, h2, jpa, lombok

model Book

JPA

Hibernate

JDBC

JPA maps model class with table automatically
JPA comes with many pre-defined methods to perform data opertions on database

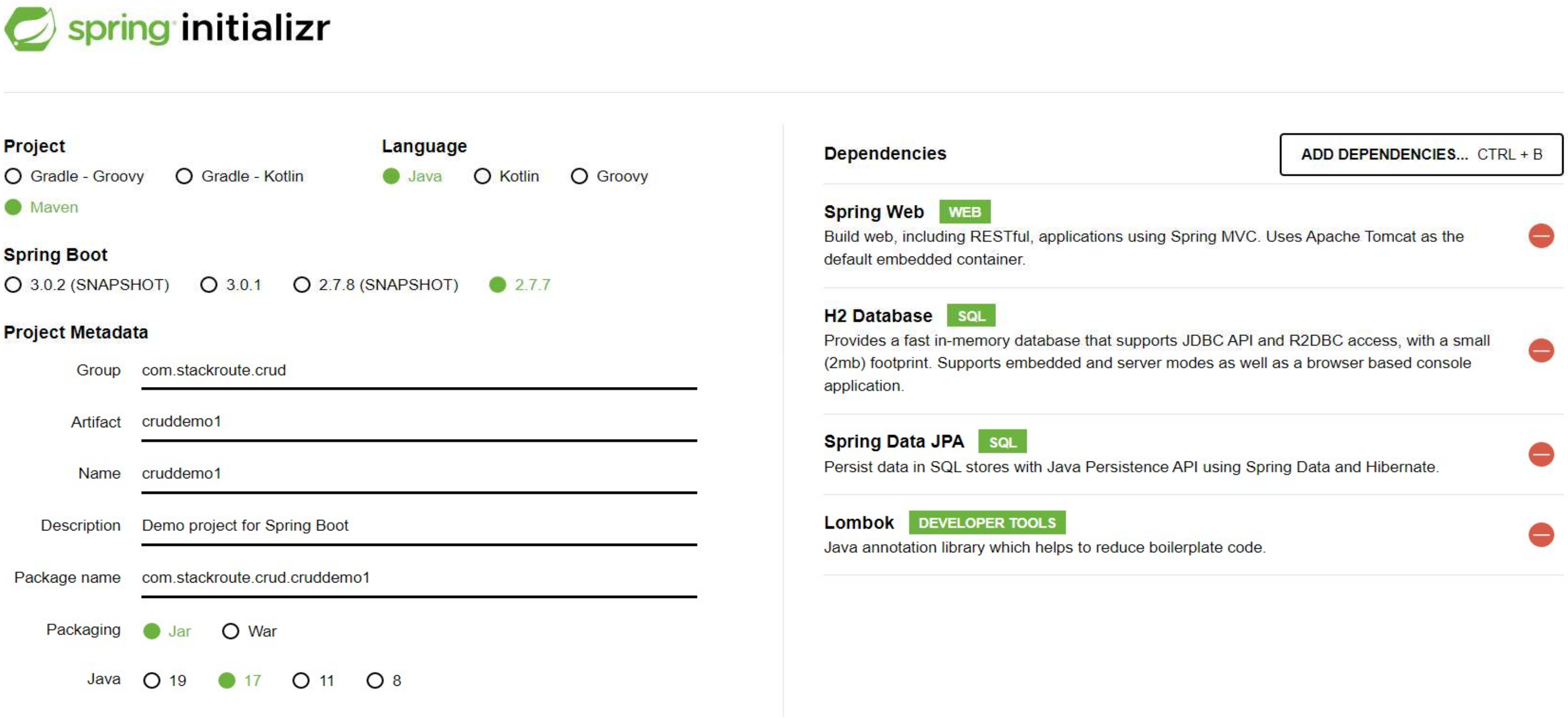
RESTFULLAPI using sringboot

Book data

get all books
add new book

get book by id
update book
delete book

Step 1 Create new application in spring.initializer
add required dependencies web, h2, jpa, lombok
download, extract, copy to workspace, open in intellij
check settings if required
create required packages


spring initializr

Project
○ Gradle - Groovy ○ Gradle - Kotlin
● Maven

Language
● Java ○ Kotlin ○ Groovy

Spring Boot
○ 3.0.2 (SNAPSHOT) ○ 3.0.1 ○ 2.7.8 (SNAPSHOT) ● 2.7.7

Project Metadata

Group	com.stackroute.crud
Artifact	cruddemo1
Name	cruddemo1
Description	Demo project for Spring Boot
Package name	com.stackroute.crud.cruddemo1
Packaging	● Jar ○ War
Java	○ 19 ● 17 ○ 11 ○ 8

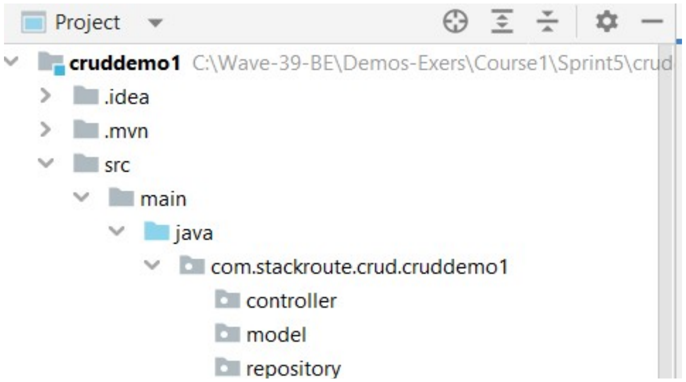
Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

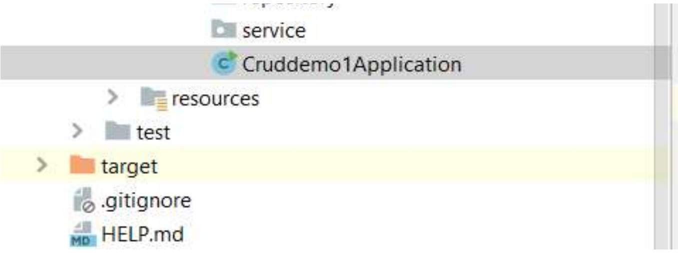
H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

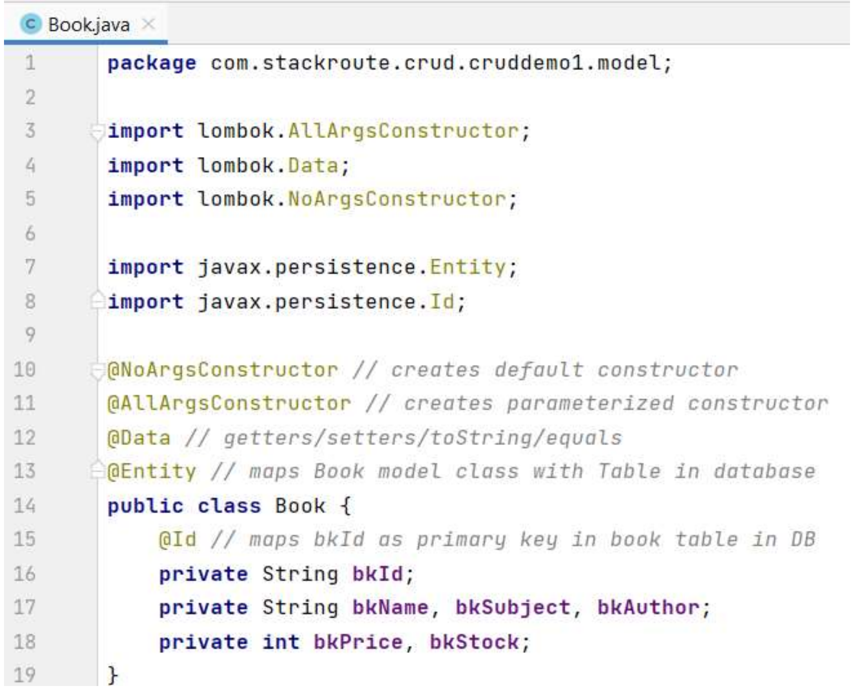


```
cruddemo1 C:\Wave-39-BE\Demos-Exers\Course1\Sprint5\crud
  .idea
  .mvn
  src
    main
      java
        com.stackroute.crud.cruddemo1
          controller
          model
          repository
```



Step 2 Create model class

Book

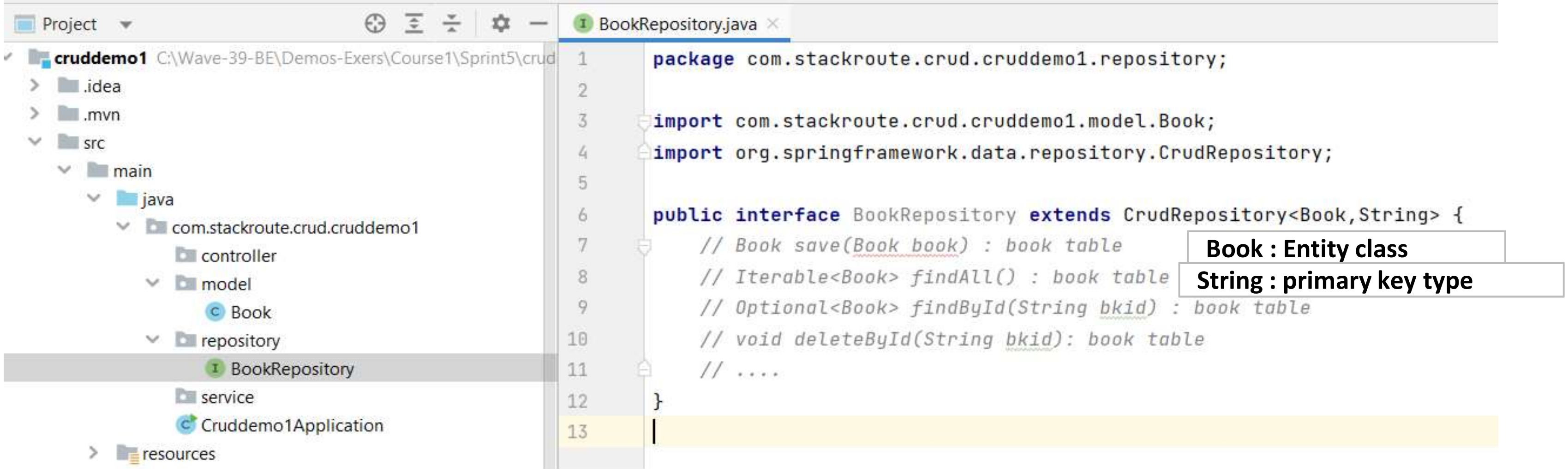


```java
package com.stackroute.crud.cruddemo1.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.Id;

@NoArgsConstructor // creates default constructor
@AllArgsConstructor // creates parameterized constructor
@Data // getters/setters/toString/equals
@Entity // maps Book model class with Table in database
public class Book {
    @Id // maps bkId as primary key in book table in DB
    private String bkId;
    private String bkName, bkSubject, bkAuthor;
    private int bkPrice, bkStock;
}
```

Step 3 Define repository layer

JPA framework comes with pre-defined interfaces with generalized methods to perform data operations in DB

CrudRepository	insert record	save()
JpaRepository	get all records	findAll()
	get record by id	findById()
	delete record	deleteById()
	update record	save()

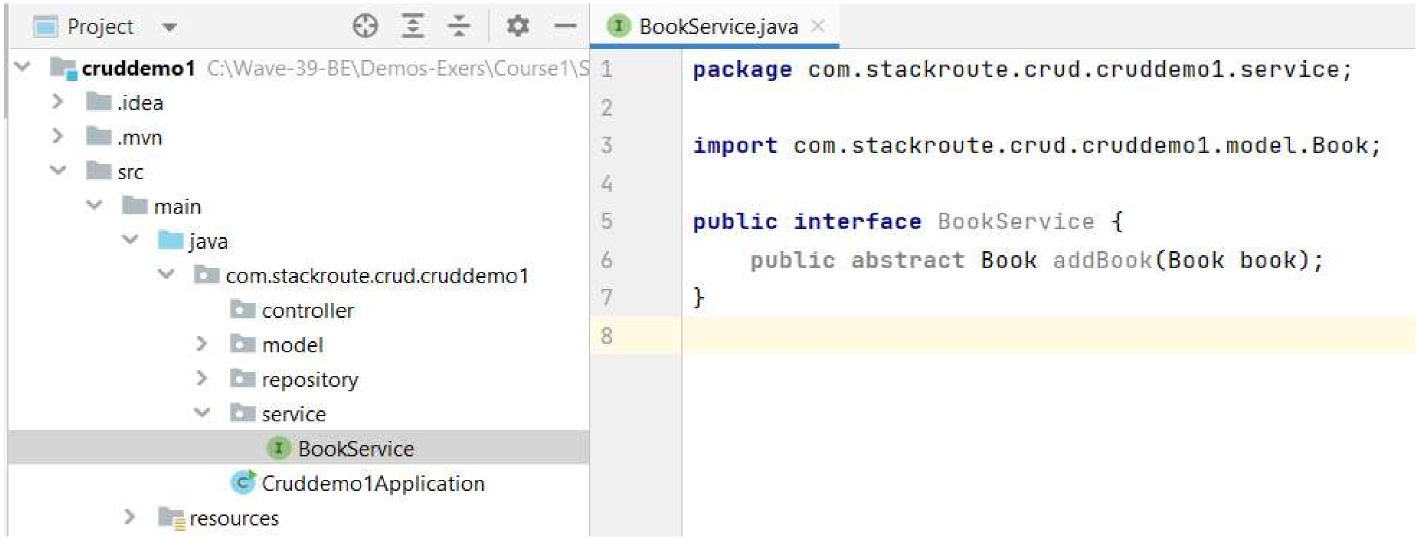


```java
package com.stackroute.crud.cruddemo1.repository;

import com.stackroute.crud.cruddemo1.model.Book;
import org.springframework.data.repository.CrudRepository;

public interface BookRepository extends CrudRepository<Book,String> {
    // Book save(Book book) : book table
    // Iterable<Book> findAll() : book table
    // Optional<Book> findById(String bkid) : book table
    // void deleteById(String bkid): book table
    // ....
}
```

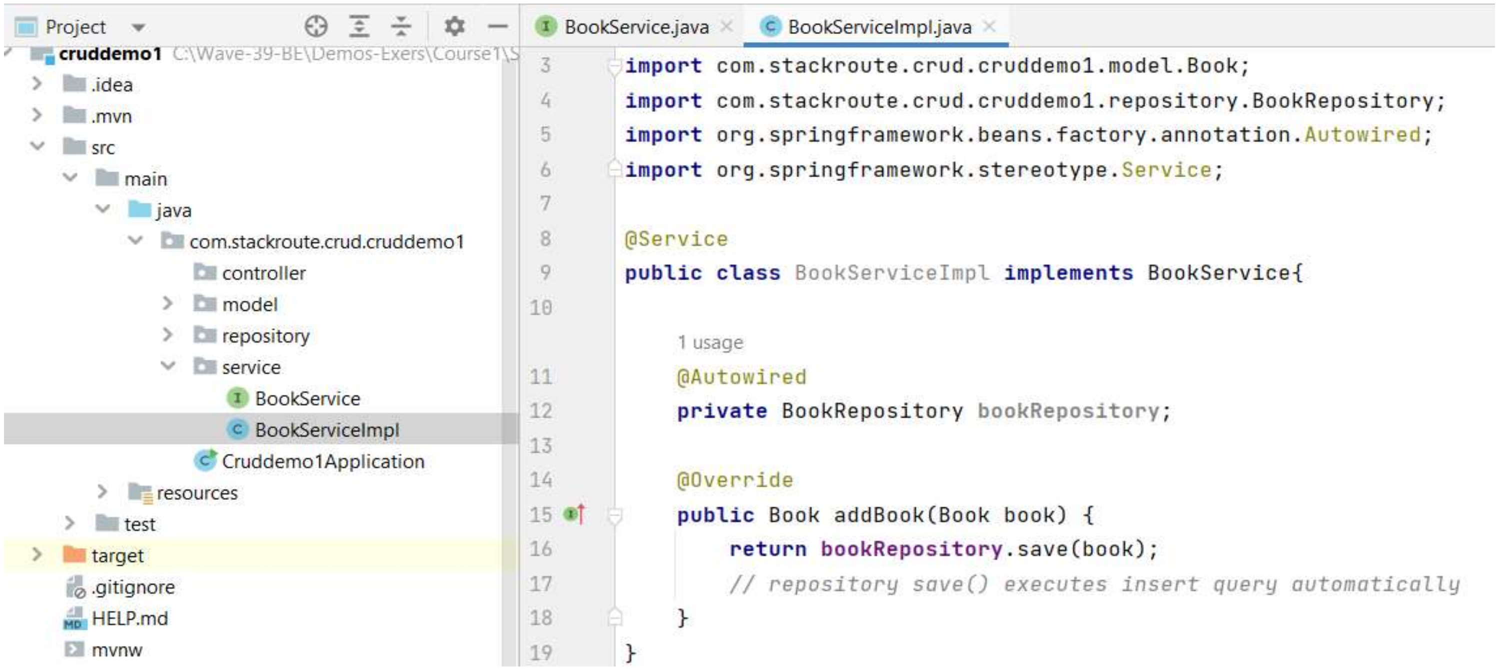
Book : Entity class

String : primary key type

Step 4 Define service layer

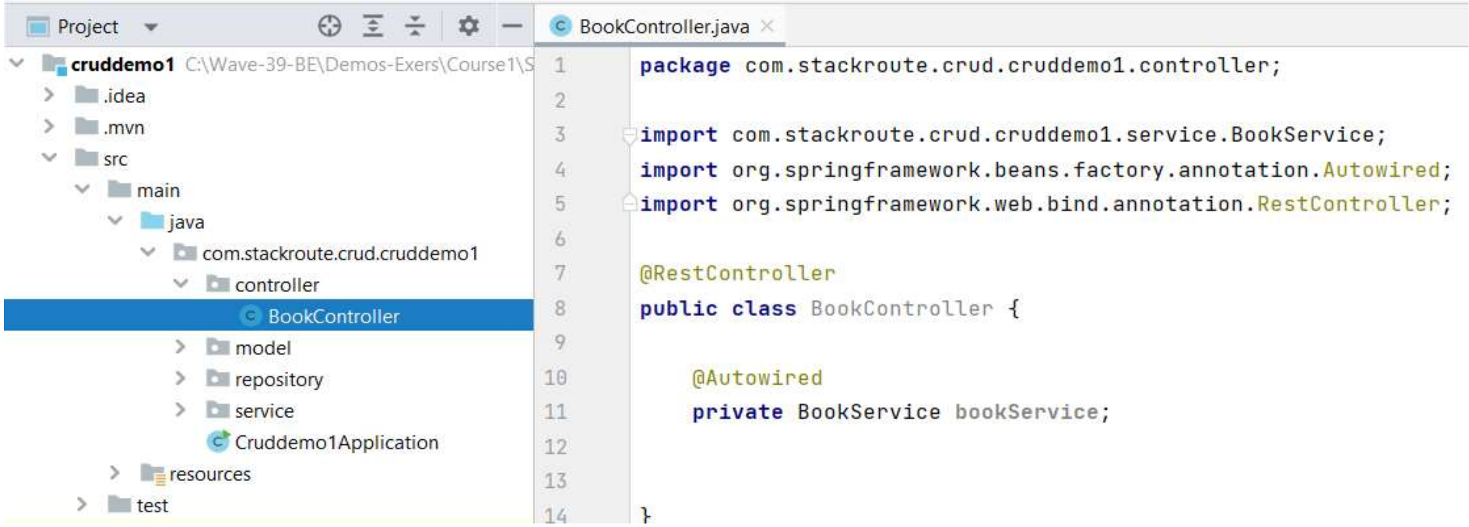


```java
package com.stackroute.crud.cruddemo1.service;

import com.stackroute.crud.cruddemo1.model.Book;

public interface BookService {
    public abstract Book addBook(Book book);
}
```



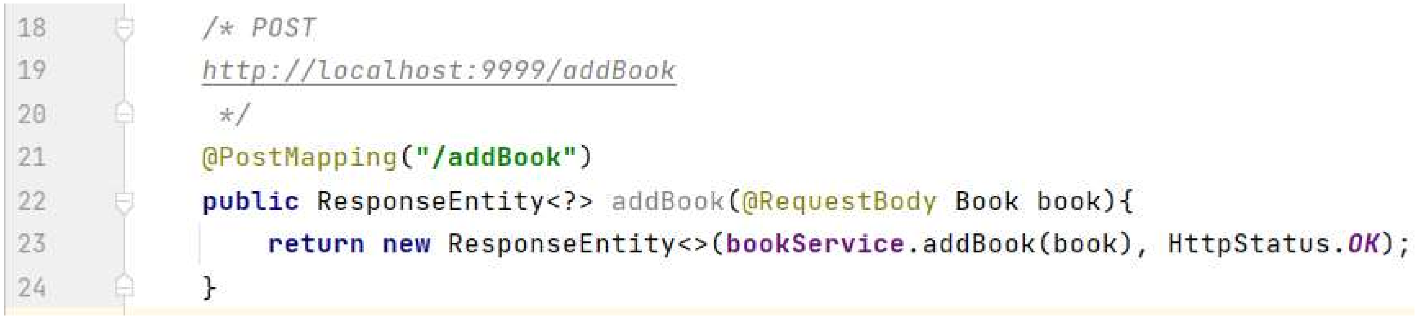
```java
import com.stackroute.crud.cruddemo1.model.Book;
import com.stackroute.crud.cruddemo1.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class BookServiceImpl implements BookService{

    @Autowired
    private BookRepository bookRepository;

    @Override
    public Book addBook(Book book) {
        return bookRepository.save(book);
        // repository save() executes insert query automatically
    }
}
```

Step 5 Define controller
inject service dependency



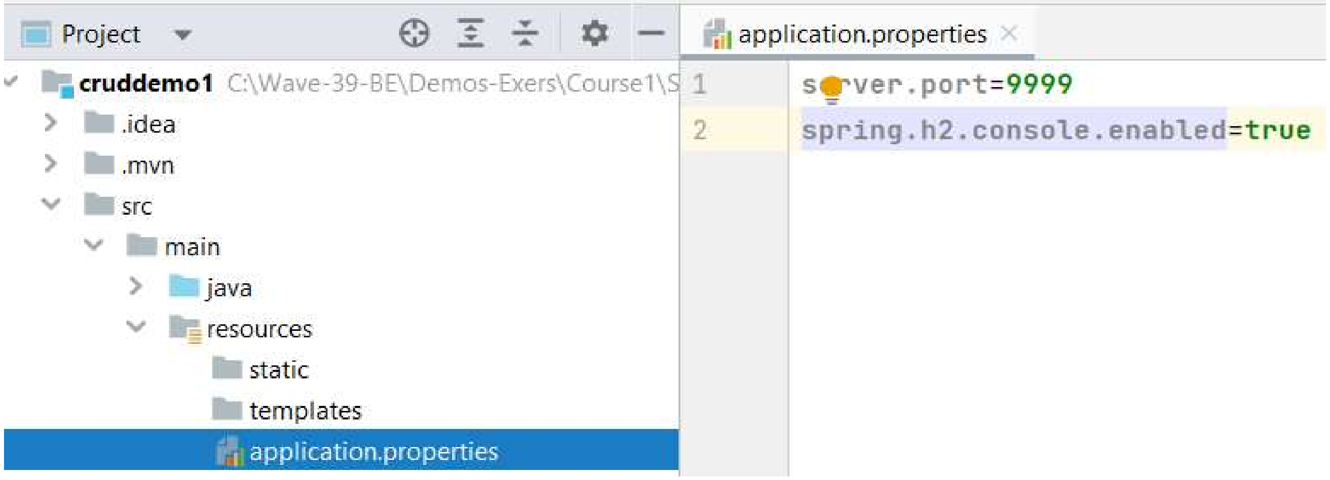
```java
package com.stackroute.crud.cruddemo1.controller;

import com.stackroute.crud.cruddemo1.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class BookController {

    @Autowired
    private BookService bookService;


}
```

Step 6 Define request hanlder methods in controller
to add book record



```
/* POST
http://localhost:9999/addBook
 */
@PostMapping("/addBook")
public ResponseEntity<?> addBook(@RequestBody Book book){
    return new ResponseEntity<>(bookService.addBook(book), HttpStatus.OK);
}
```

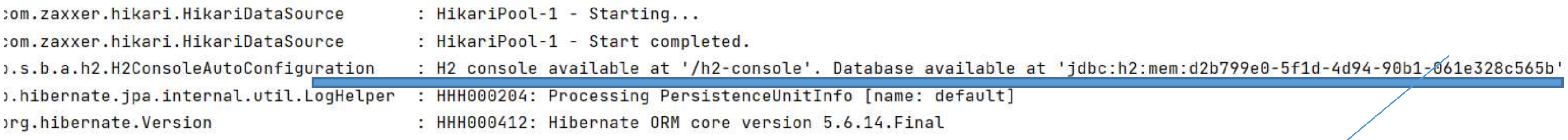
Step 7 application.properies
port no
enable in-memory h2



```
server.port=9999
spring.h2.console.enabled=true
```

run application
check in postman

make sure in-memory h2 is running

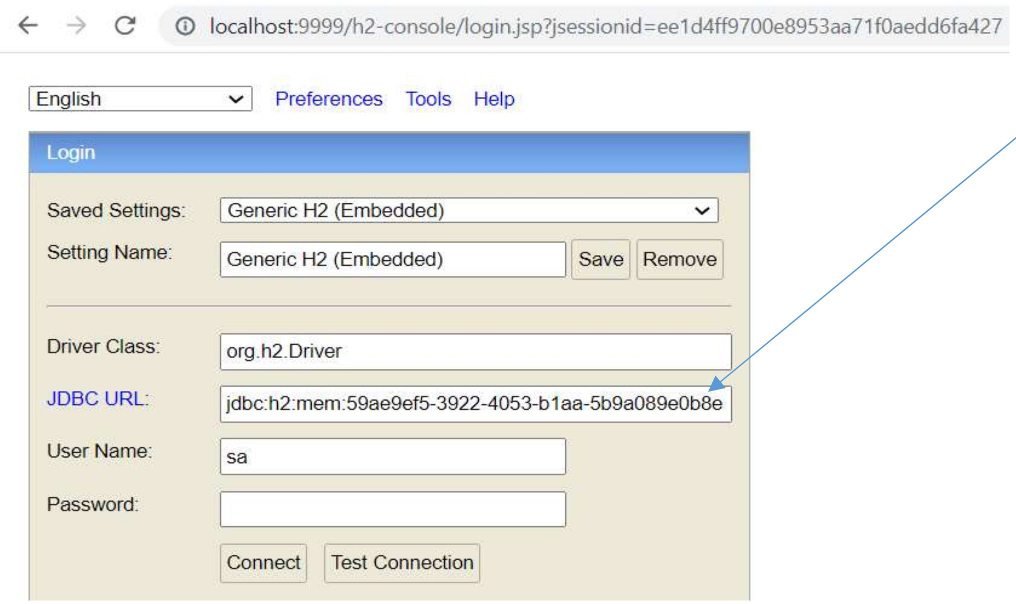


```
:om.zaxxer.hikari.HikariDataSource        : HikariPool-1 - Starting...
:om.zaxxer.hikari.HikariDataSource        : HikariPool-1 - Start completed.
).s.b.a.h2.H2ConsoleAutoConfiguration     : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:d2b799e0-5f1d-4d94-90b1-061e328c565b'
).hibernate.jpa.internal.util.LogHelper   : HHH000204: Processing PersistenceUnitInfo [name: default]
org.hibernate.Version                     : HHH000412: Hibernate ORM core version 5.6.14.Final
```

to check h2

open browser

http://localhost:xxxx/h2-console



localhost:9999/h2-console/login.jsp?jsessionid=ee1d4ff9700e8953aa71f0aedd6fa427

Saved Settings: Generic H2 (Embedded)

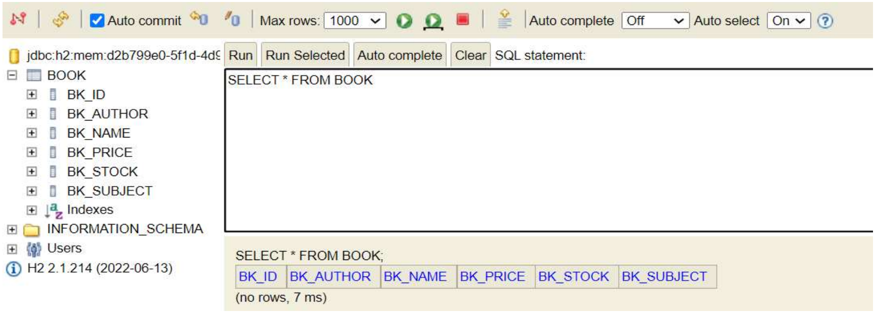
Setting Name: Generic H2 (Embedded)

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:59ae9ef5-3922-4053-b1aa-5b9a089e0b8e

User Name: sa

Password:



```
SELECT * FROM BOOK
```

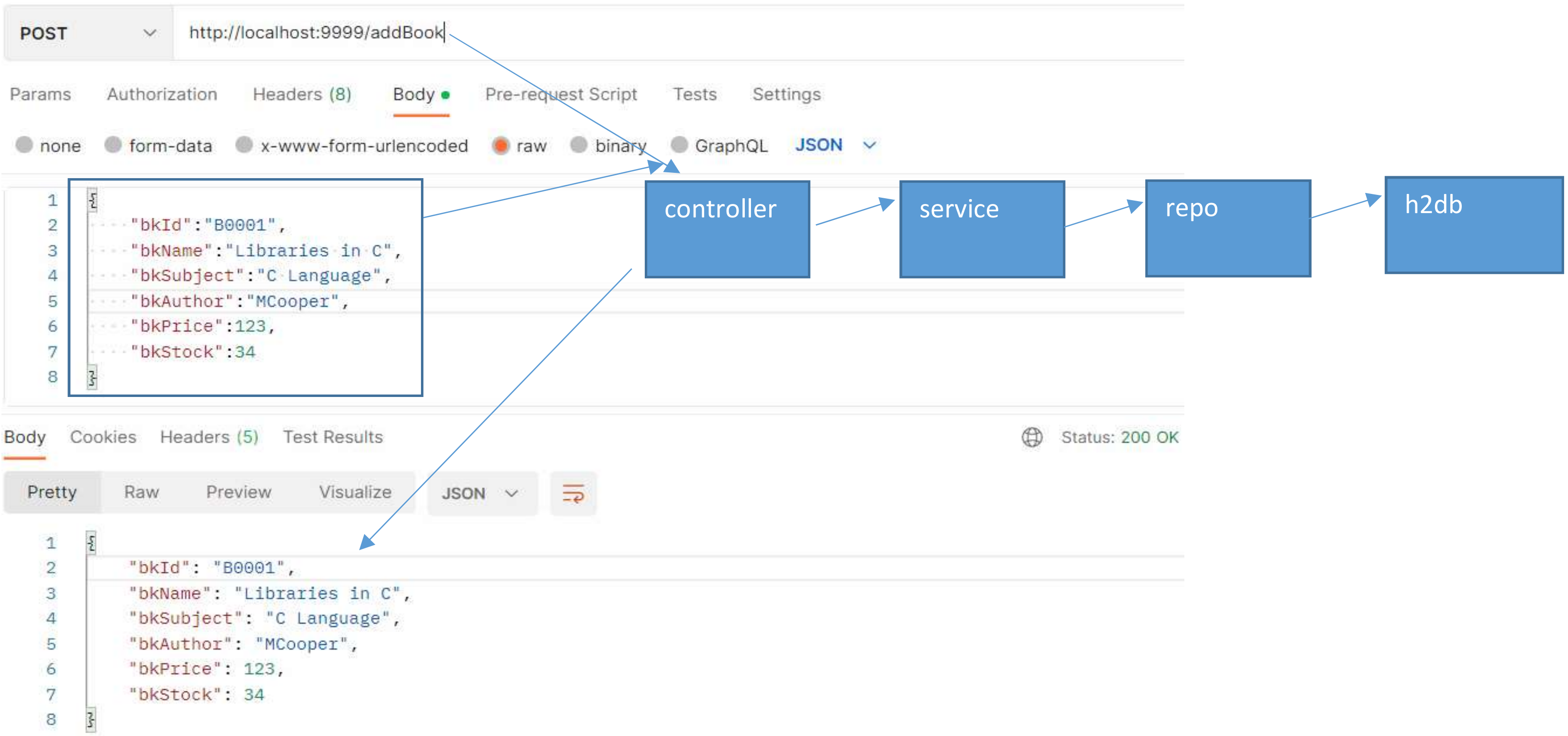
SELECT * FROM BOOK;

BK_ID	BK_AUTHOR	BK_NAME	BK_PRICE	BK_STOCK	BK_SUBJECT

(no rows, 7 ms)

in-memory h2 is working fine

check in post man by adding one record



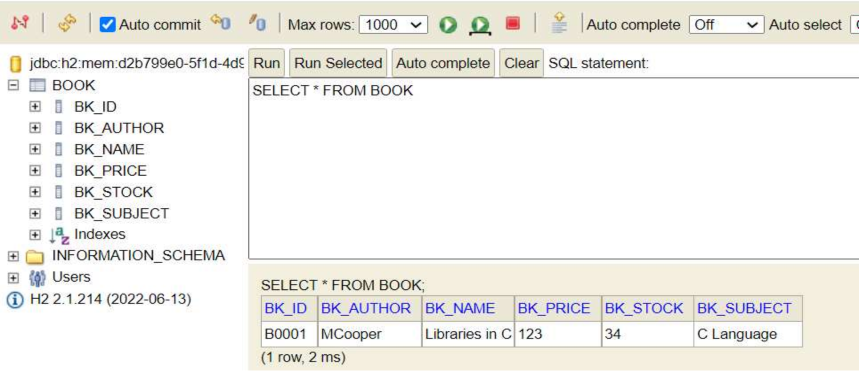
http://localhost:9999/addBook

```
{
    "bkId":"B0001",
    "bkName":"Libraries in C",
    "bkSubject":"C Language",
    "bkAuthor":"MCooper",
    "bkPrice":123,
    "bkStock":34
}
```

Status: 200 OK

```
{
    "bkId": "B0001",
    "bkName": "Libraries in C",
    "bkSubject": "C Language",
    "bkAuthor": "MCooper",
    "bkPrice": 123,
    "bkStock": 34
}
```

check in h2,make sure record inserted



```
SELECT * FROM BOOK
```

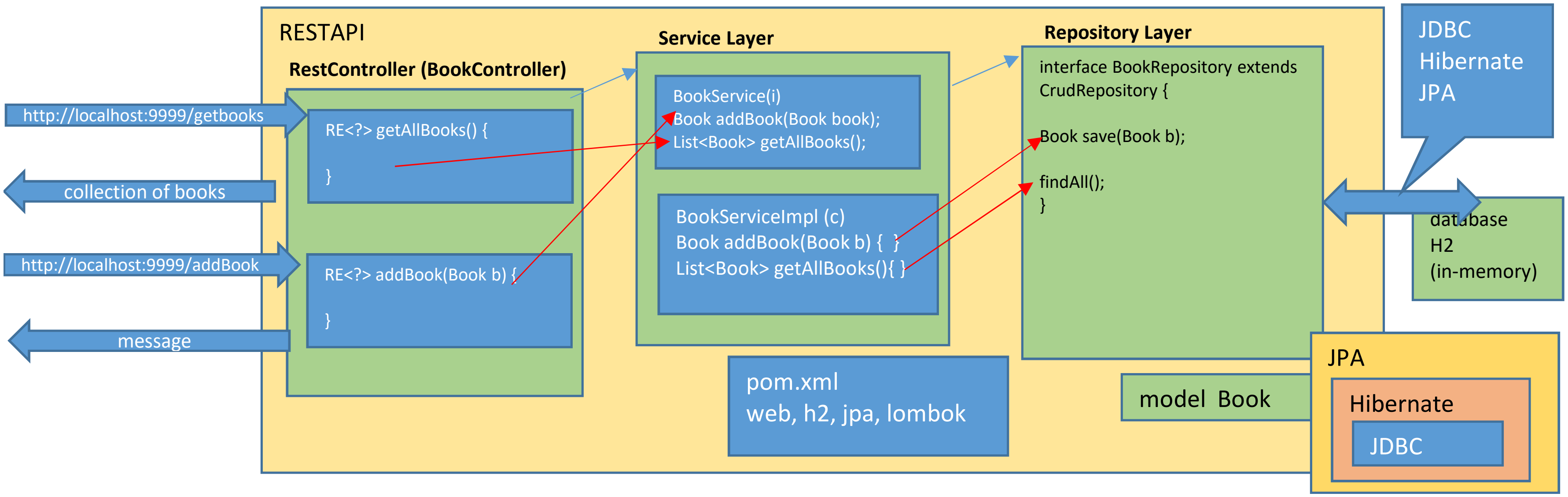
SELECT * FROM BOOK;

BK_ID	BK_AUTHOR	BK_NAME	BK_PRICE	BK_STOCK	BK_SUBJECT
B0001	MCooper	Libraries in C	123	34	C Language

(1 row, 2 ms)

Adding record is done

Demo to get records



RESTAPI

RestController (BookController)

http://localhost:9999/getbooks

collection of books

http://localhost:9999/addBook

message

RE<?> getAllBooks() {

}

RE<?> addBook(Book b) {

}

Service Layer

BookService(i)
Book addBook(Book book);
List<Book> getAllBooks();

BookServiceImpl (c)
Book addBook(Book b) { }
List<Book> getAllBooks(){ }

Repository Layer

interface BookRepository extends CrudRepository {

Book save(Book b);

findAll();
}

JDBC
Hibernate
JPA

database
H2
(in-memory)

pom.xml
web, h2, jpa, lombok

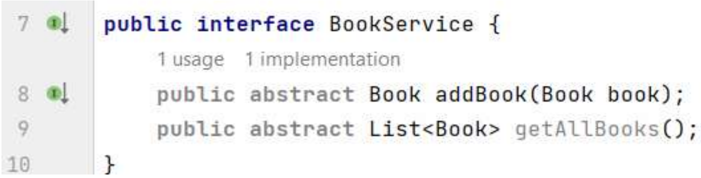
model Book

JPA

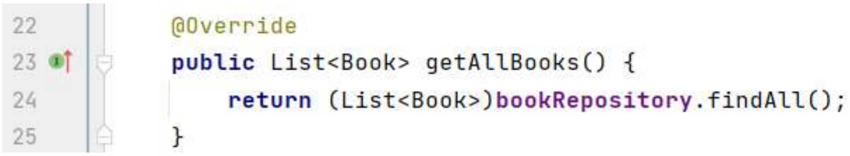
Hibernate

JDBC

step 1 define service layer

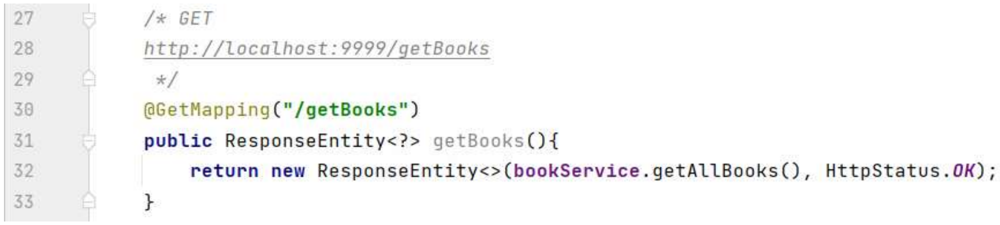


```
public interface BookService {
    1 usage   1 implementation
    public abstract Book addBook(Book book);
    public abstract List<Book> getAllBooks();
}
```



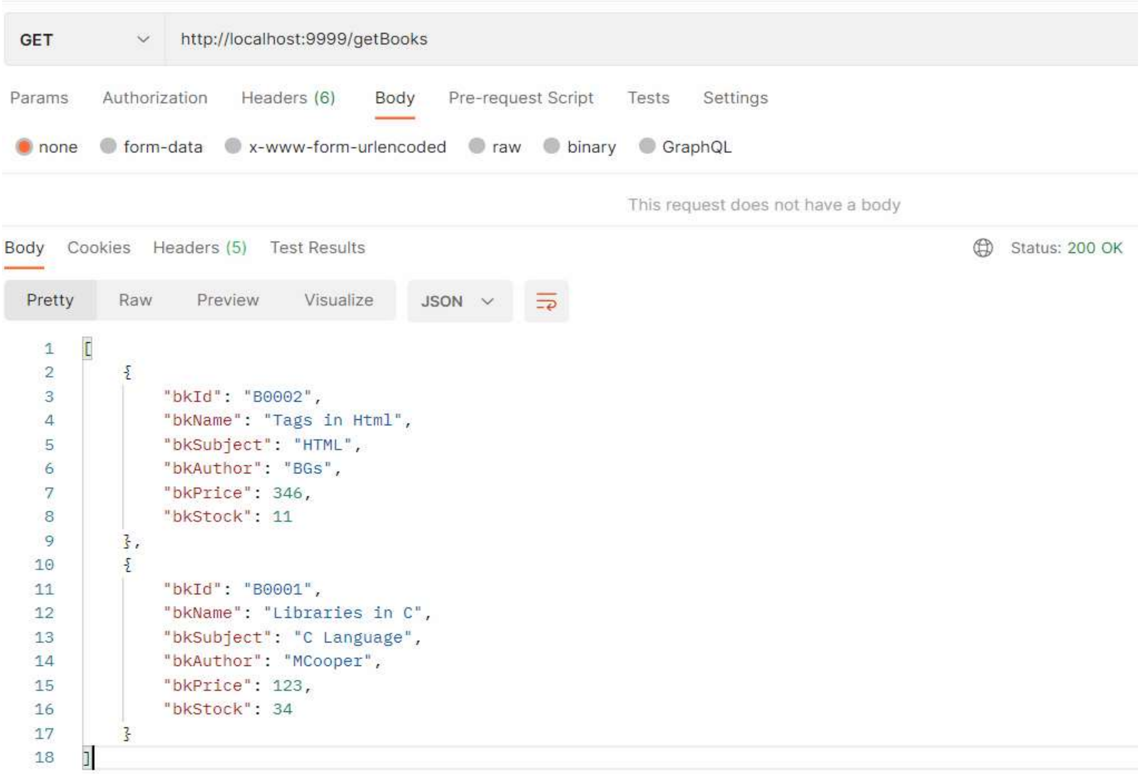
```
    @Override
    public List<Book> getAllBooks() {
        return (List<Book>)bookRepository.findAll();
    }
```

step 2 define request handler method in controller



```
    /* GET
    http://localhost:9999/getBooks
     */
    @GetMapping("/getBooks")
    public ResponseEntity<?> getBooks(){
        return new ResponseEntity<>(bookService.getAllBooks(), HttpStatus.OK);
    }
```

run and test in postman
insert few records



GET http://localhost:9999/getBooks

● none ○ form-data ○ x-www-form-urlencoded ○ raw ○ binary ○ GraphQL

This request does not have a body

Status: 200 OK

```
[
    {
        "bkId": "B0002",
        "bkName": "Tags in Html",
        "bkSubject": "HTML",
        "bkAuthor": "BGs",
        "bkPrice": 346,
        "bkStock": 11
    },
    {
        "bkId": "B0001",
        "bkName": "Libraries in C",
        "bkSubject": "C Language",
        "bkAuthor": "MCooper",
        "bkPrice": 123,
        "bkStock": 34
    }
]
```