

# ReadingsGood Documentation

## About

ReadingsGood is a stand-alone application that was built using Spring-Boot and it has REST Api.

System tools:

- Java: 11
- Maven: 4.0.0
- Spring-boot: 2.7.5
- Database: MongoDB (v4.4.6)

System has the basic models needed to meet the requirements: **User**, **Book**, **Order**, **Stats** and **Role** (ERole enum class also exists to provide accessing role types easily). There are also **BookDao**, **CustomerDao** and **OrderDao** classes in the Data Access Layer to achieve application-based operations in the Persistence Layer.

Whole application is secured by **Bearer** (JWT) authentication except **login** and **signup** requests and also Preauthorized endpoints with admin and user roles.

**BookController**, **OrderController**, **CustomerController** and **StatisticsController** handle the incoming requests and return meaningful response messages. With using these controllers, below executions can be processed.

1. Customer Controller:
  - new user can sign up
  - user can bring all her/his orders
  - user can log in to system
2. Book Controller:
  - admin can add new book
  - user can update the existing book stock (increase or decrease)
3. Order Controller:
  - user can order
  - user can bring order information with it's id
  - user or admin can get orders by giving date interval
4. Statistics Controller:
  - user can bring monthly order statistics

## Run Project

Application runs on port **9090** and main MongoDB runs on **27017 (in Docker network)**. Mongo data can be tracked on port **8081** via Mongo Express which is a third-party tool.

**Mongo Express credentials:** username: **admin**, password: **admin**

There are two ways to start (in the main project path):

- Important Note:** For local execution, please remove **rig** service (short of the readingisgood) from (or comment in) ***docker-compose.yml*** file otherwise application may throw ***port already in use error***.

## 1. Sign Up Request

**POST** - <http://localhost:9090/api/customer/signup> with body



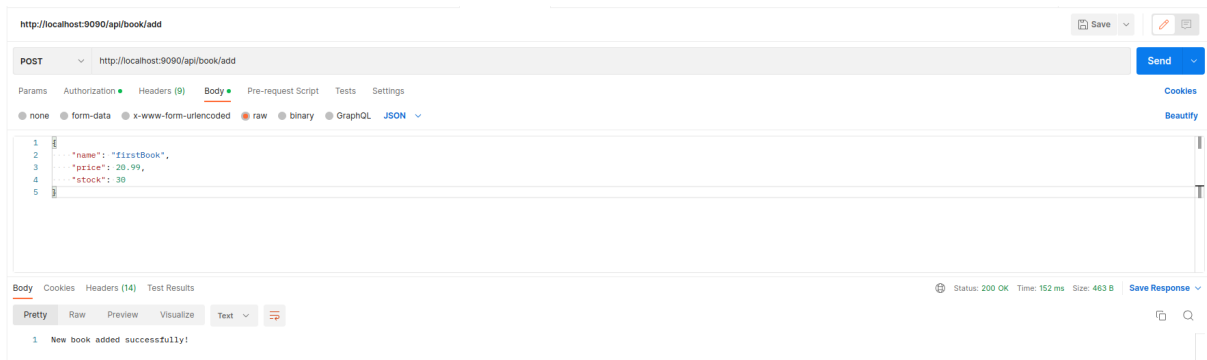
**POST** - <http://localhost:9090/api/customer/login> with body



### 3. New Book Request

**Usage:** Name and price should be given in a proper way. If stock is not given, it can be updated later but price not. Stock must not be negative, name and price must not be null.

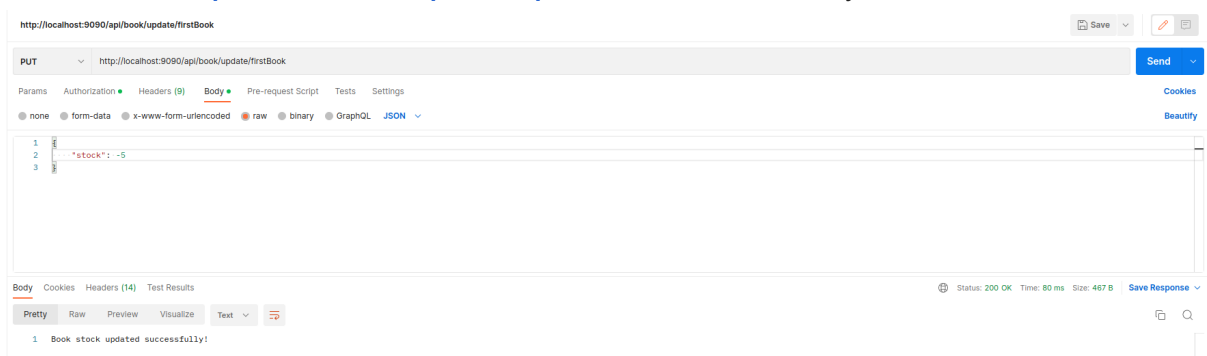
**POST** - <http://localhost:9090/api/book/add> with body



### 4. Book Update Request

**Usage:** Book name for update must be given in a URL path. Stock should be given for decreasing (negative) or increasing (positive).

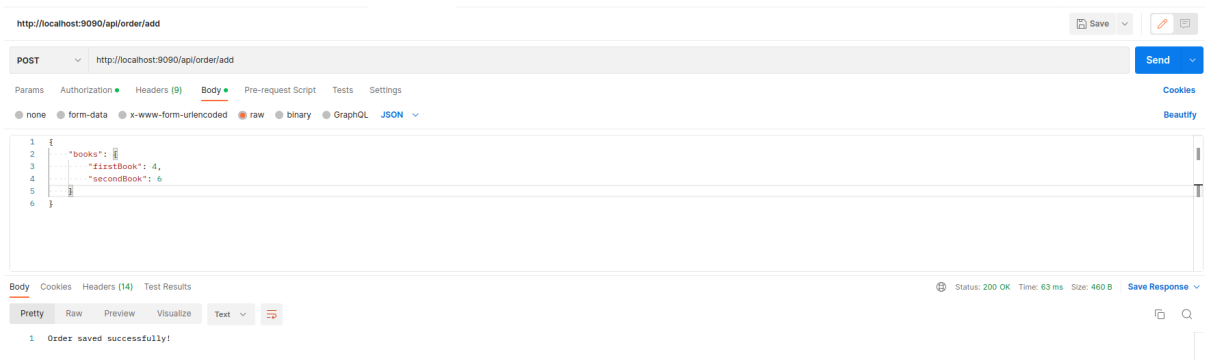
**PUT** - <http://localhost:9090/api/book/update/<bookName>> with body



## 5. Add Order Request

**Usage:** Books for order must be given in the below format. Book count should be a positive number.

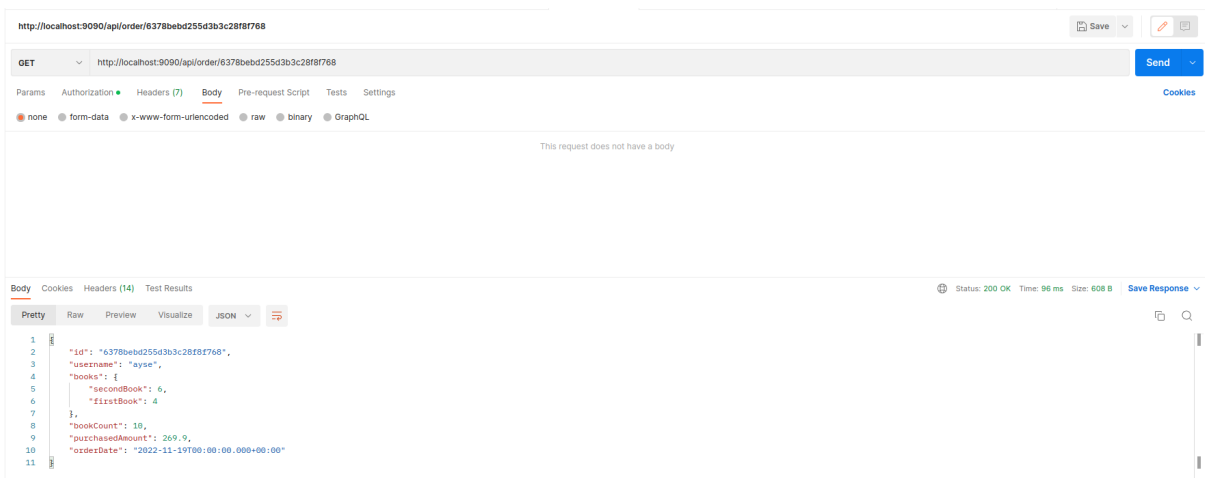
**POST** - <http://localhost:9090/api/order/add> with body



## 6. Get Order By Id Request

**Usage:** Order ID is given automatically from MongoDB so it should be brought from the database. Mongo Express can be used for this operation.

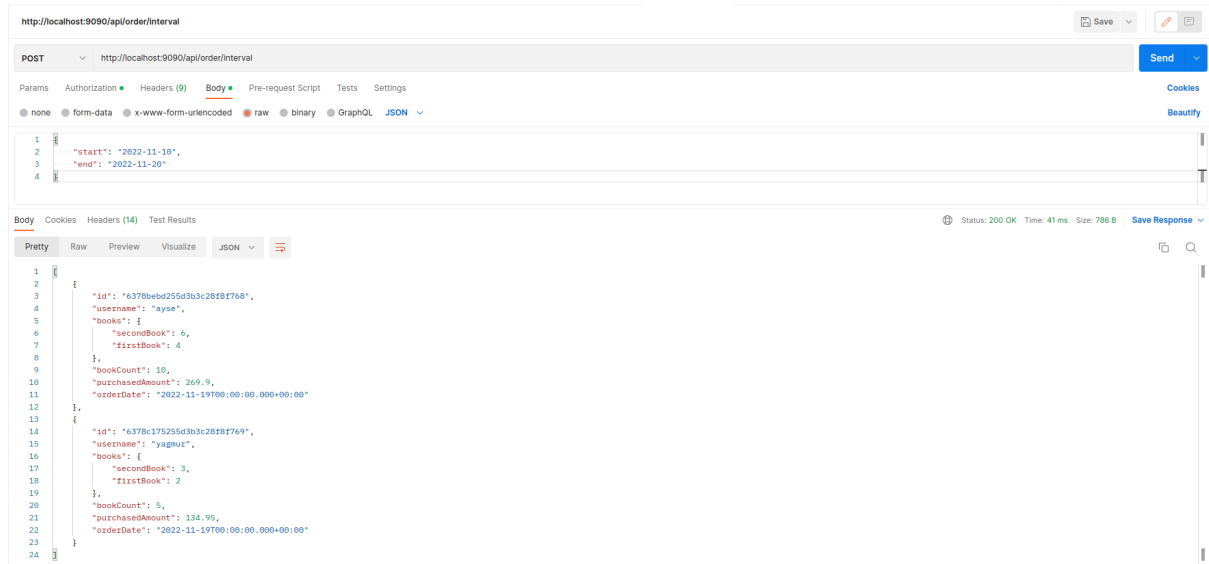
**GET** - <http://localhost:9090/api/order/<orderId>> without body



## 7. Get Orders By Interval Request

**Usage:** Start and end dates must be given in a proper format as below.

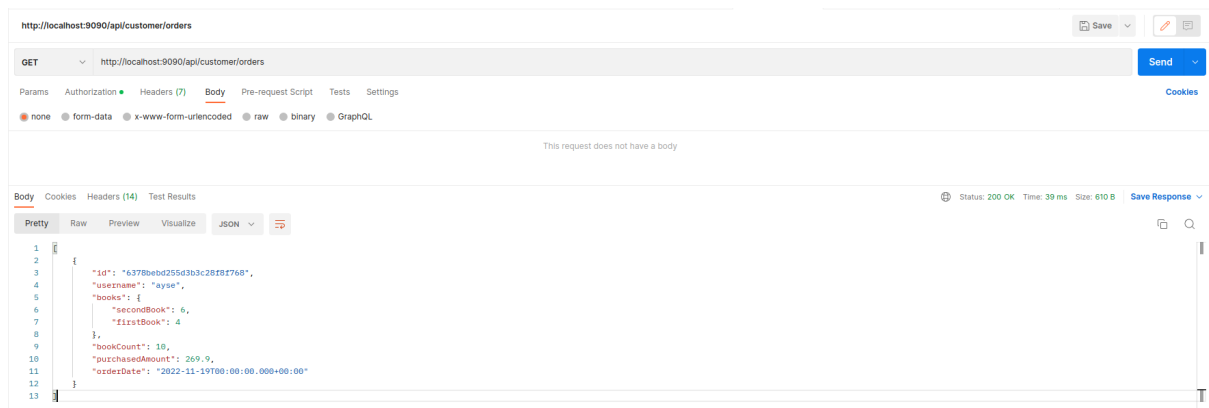
**POST** - <http://localhost:9090/api/order/interval> with body



## 8. Get Customer Orders Request

**Usage:** Bearer token for this request is enough.

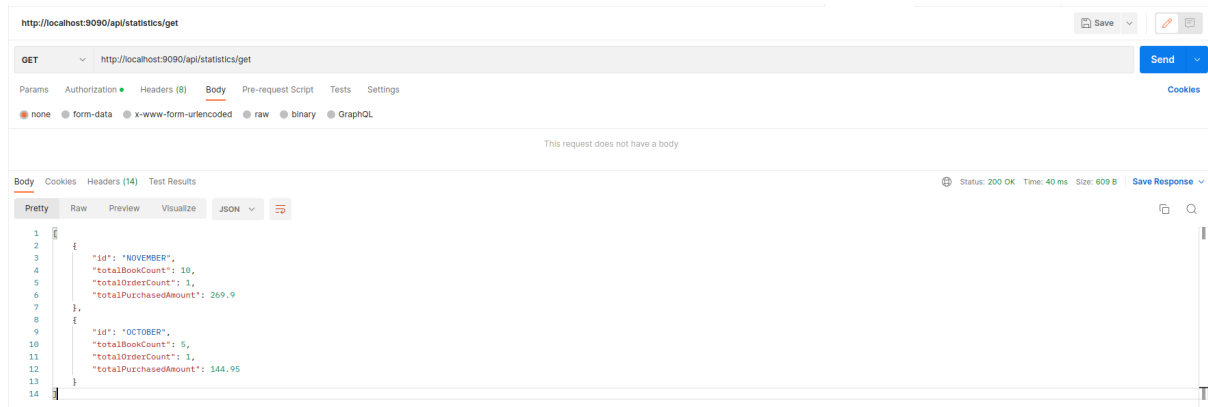
**GET** - <http://localhost:9090/api/customer/orders> without body



## 9. Get Customer Statistics Request

**Usage:** Bearer token for this request is enough.

**GET** - <http://localhost:9090/api/statistics/get> without body



If it is desired to test that all the requirements are fulfilled, the data can be manipulated via Mongo Express.

## Tests

**Assumption:** Test environment must be ready. Tests will be run independently and they have different Mongo database so please run the

1. **`docker-compose -f docker-compose-test.yml up -d --build`** command.

If you get **WARNING: Found orphan containers**, there is no problem. This is because there are two different `docker-compose.yml` files and they try to run in the same project directory but this does not prevent the application and the tests from running separately.

In the test environment, MongoDB also stays on port **27017** and Mongo data for testing can be tracked on port **8082**.

**Mongo Express credentials:** username: admin, password: admin

## Run Tests

Please run the following command to start all tests (in main project path):

1. **`./mvnw test`**

While tests are running, they will create their own data and after tests finish, they will delete all test data from MongoDB to avoid conflict. You can not see the test data because of this situation.

## Coverage Results

For coverage results, tests are run with coverage thanks to IDE.

Coverage: All in ReadingIsGood x			
Element ^	Class, %	Method, %	Line, %
com	100% (23/23)	80% (108/134)	70% (273/385)
example	100% (23/23)	80% (108/134)	70% (273/385)
readingisgood	100% (23/23)	80% (108/134)	70% (273/385)
controller	100% (4/4)	92% (13/14)	60% (48/80)
dao	100% (3/3)	90% (19/21)	79% (69/87)
model	100% (6/6)	64% (32/50)	65% (66/101)
payload	100% (3/3)	100% (20/20)	100% (23/23)
repository	100% (0/0)	100% (0/0)	100% (0/0)
security	100% (6/6)	85% (24/28)	71% (66/92)
ReadingIsGoodApplication	100% (1/1)	0% (0/1)	50% (1/2)

Selcan Yağmur Atak  
+90 (506) 417 10 40  
selcanyagmur.atak@gmail.co